HARDWARE FOR ARTIFICIAL INTELLIGENCE

EDITED BY: Alexantrou Serb, Melika Payvand, Irem Boybat and Oliver Rhodes

PUBLISHED IN: Frontiers in Neuroscience, Frontiers in Artificial Intelligence and Frontiers in Computational Neuroscience

frontiers Research Topics

frontiers

ISSN 1664-8714
ISBN 978-2-88976-398-6
DOI 10.3389/978-2-88976-398-6

About Frontiers

Frontiers is more than just an open-access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers Journal Series

The Frontiers Journal Series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the Frontiers Journal Series operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to Quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews.
Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the Frontiers Journals Series: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area! Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers Editorial Office: frontiersin.org/about/contact

HARDWARE FOR ARTIFICIAL INTELLIGENCE

Topic Editors:
Alexantrou Serb, University of Southampton, United Kingdom
Melika Payvand, University of Zurich, Switzerland
Irem Boybat, IBM Research - Zurich, Switzerland
Oliver Rhodes, The University of Manchester, United Kingdom

Citation: Serb, A., Payvand, M., Boybat, I., Rhodes, O., eds. (2022). Hardware for Artificial Intelligence. Lausanne: Frontiers Media SA.
doi: 10.3389/978-2-88976-398-6

Table of Contents

frontiers | Frontiers in Neuroscience

TYPE Editorial
PUBLISHED 09 August 2022
DOI 10.3389/fnins.2022.979495

OPEN ACCESS

EDITED AND REVIEWED BY
André van Schaik,
Western Sydney University, Australia

*CORRESPONDENCE
Oliver Rhodes
oliver.rhodes@manchester.ac.uk

SPECIALTY SECTION
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

RECEIVED 27 June 2022
ACCEPTED 15 July 2022
PUBLISHED 09 August 2022

CITATION
Boybat I, Payvand M, Rhodes O and Serb A (2022) Editorial: Hardware for artificial intelligence.
Front. Neurosci. 16:979495.
doi: 10.3389/fnins.2022.979495

Editorial: Hardware for artificial intelligence

Irem Boybat[1], Melika Payvand[2], Oliver Rhodes[3]* and Alexander Serb[4]

[1]IBM Research Europe, Rüschlikon, Switzerland, [2]Institute for Neuroinformatics, University of Zurich, Zurich, Switzerland, [3]Advanced Processor Technologies Group, Department of Computer Science, The University of Manchester, Manchester, United Kingdom, [4]Institute for Integrated Micro and Nano Systems, School of Engineering, The University of Edinburgh, Edinburgh, United Kingdom

KEYWORDS

artificial intelligence, in memory computing (IMC), noisy hardware, low energy computing, neural networks, neuromorphic computing

Editorial on the Research Topic
Hardware for artificial intelligence

The remarkable progress in Artificial Intelligence (AI) has been made possible by a "perfect storm" emerging over the past two decades, bringing together tremendous progress in neuroscience, availability of massive amounts of data, and advents in scalable computer and software systems such as Central Processing Units (CPUs) and Graphical Processing Units (GPUs). However, as AI is increasingly integrated into our daily lives and models grow in scale for solving more complex tasks, its required energy and memory footprint is growing unsustainably. Significant research and development effort is centered around custom hardware solutions targeting low latency, high throughput and better energy savings. Complementary to this, are endeavors toward designing algorithms that are best suited for the underlying hardware. This issue aims to bring together novel research on hardware and algorithms for AI, spanning across a range of applications.

Hardware accelerators for AI

Memory is the centerpiece of AI hardware research and development, since it occupies the largest area, and is the dominant source of energy consumption in AI systems. The largest energy contribution is related to the data movement between the memory and processing units, known as the von Neumann bottleneck. To minimize this data movement, there is substantial interest in bringing these two units together, known as In-Memory Computing (IMC). Certain computational operations, such as the matrix-vector multiplication that lie at the heart of all neural network operations, can be performed by leveraging Ohm's and Kirchoff's current laws on custom-designed memory arrays, e.g., using Static Random Access Memory (SRAM) and resistive memory. IMC is thus one of the main themes within this issue, featuring also benchmarking efforts for the hardware, including comparative studies across conventional hardware (e.g., CPUs and GPUs) (Steffen et al.) and IMC solutions from a range of hardware/application

perspectives (Bagheriye and Kwisthout, Dazzi et al., Kim et al., Lu et al.). Specifically, NeuroSim, a simulator for compute-in-memory hardware accelerators, is presented in Lu et al. The simulator provides design tools for a range of IMC architectures, and linking device, circuit and algorithmic level performance. The simulator is validated against post-layout simulation of an actual 40 nm RRAM-based IMC macro design.

Additional techniques for improving performance of AI algorithms are also presented within the Research Topic. For example, low-power systems exploiting event-driven architectures for in-sensor compute (Stuijt et al.), and always-on systems for edge implementations of AI algorithms (Chundi et al.). Insights on optoelectronic platforms are also provided, leveraging the complementary properties of optics and electronics (Primavera and Shainline).

Advances in algorithms for AI hardware

While the novel AI hardware solutions proposed above promise significant gains in energy efficiency, it remains a relatively big challenge to map conventional AI algorithms and workflows onto systems with novel substrates and hybrid bit-precision support. Conventional CPU/GPU-based hardware typically makes use of shared memory and message passing to allow implementation of algorithms such as stochastic gradient descent (SGD) for training, and the underlying floating point number representations enable precise and repeatable computation. Systems based on on-chip low-precision memory omit these features by design, thus requiring different hardware-aware algorithms for training and mapping, to realize their full potential.

The TTv2 algorithm is proposed by Gokmen, which builds on previous work to improve the noise tolerance by 100×, and reduce the number of device conductance states from 1,000s to 10s (100×). The noise tolerance of matrix-vector multiplication is also improved (10×), resulting in an algorithm capable of optimizing DNNs close to their ideal accuracy even at extremely noisy hardware settings. Meanwhile neural network training with asymmetric cross-point elements is investigated by Onen et al.. This work demonstrates how device asymmetry can be exploited, rather than updating model parameters in the direction of negative gradients, the total energy of the system incorporating the effects of device asymmetry is minimized, enabling realization of analog deep learning accelerators. Laborieux et al. adapt equilibrium propagation applied to deep conv nets by reducing gradient estimator bias. This allows local learning in systems such as recurrent neural networks, and the ability to unlock the potential of the IMC devices explored in the remainder of the Research Topic. Zhao et al. employ minibatch-SGD to train memristive devices. The research harnesses gradient averaging across the minibatch and stochastic rounding to overcome device non-idealities and vanishing gradients. Memory overheads are kept low through the use of decomposition methods, and the task of reconstructing gradient matrices internally and externally to memristor arrays is explored. The use of a streaming co-processor for training the memristor hardware is investigated, demonstrating the potential to scale up from small proof-of-concept demonstrators to the large-scale AI workflows. The approach to compress gradient information provides an important step toward biologically-plausible batch averaging during long-term learning, and avoids the poor performance experienced when training non-ideal hysteric devices with small batch sizes.

Mapping to hardware Wang et al. explore the mapping of Bayesian Confidence Propagation Neural Networks (BCPNNs) to memristor-based architecture, overcoming the von Neumann bottleneck which limits access to synaptic storage in conventional digital implementations. The implementation harnesses characteristics of the underlying hardware, e.g., using the dopant drift phenomenon of the memristor to simulate the exponential decay of the synaptic state in the BCPNN learning rule. Consistency between the memristor-based solution and software simulations in Matlab is verified, demonstrating the potential of in-circuit analog computation as a route toward real-time brain emulation. Spoon et al. explore the use of Phase Change Memory (PCM) as a substrate for transformer-based deep neural networks (BERT). The work combines noise-aware training to overcome the drift and noise inherent to PCM devices, together with reduced precision (INT6) digital computation in the attention block. By combining these techniques, software-equivalent accuracy is demonstrated, along with a prospective 11.3× reduction in energy. Overall these works demonstrate that through application of noise-aware training, non-ideal low-precision devices can be trained to produce software-equivalent performance, highlighting the potential of these emerging technologies.

Outlook

Overall, we observe that regardless of its growth in recent years, the field of AI hardware is still developing at breakneck pace and seems to have a long technological runway yet ahead of it. We note the substantial interest toward building widespread accelerators and general-purpose platforms, while automating the design process of hardware. This is a constantly evolving strand of research targeting mitigation and then ideally exploitation of hardware non-idealities in the pursuit of efficient AI computation. It is with great pleasure that we present a fleeting, yet highly interesting snapshot of the field in this issue

and we sincerely hope that you, the reader, finds it instructive, exciting and inspiring for your own future efforts.

Author contributions

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

Conflict of interest

Author IB was employed by IBM Research Europe, Rüschlikon, Switzerland.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 18 February 2021
doi: 10.3389/fnins.2021.633674

Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing Its Gradient Estimator Bias

Axel Laborieux[1]*, *Maxence Ernoult*[1,2,3]*, *Benjamin Scellier*[3†], *Yoshua Bengio*[3,4], *Julie Grollier*[2] *and Damien Querlioz*[1]

[1] Université Paris-Saclay, CNRS, Centre de Nanosciences et de Nanotechnologies, Palaiseau, France, [2] Unité Mixte de Physique, CNRS, Thales, Université Paris-Saclay, Palaiseau, France, [3] Mila, Université de Montréal, Montreal, QC, Canada, [4] Canadian Institute for Advanced Research, Toronto, ON, Canada

Equilibrium Propagation is a biologically-inspired algorithm that trains convergent recurrent neural networks with a local learning rule. This approach constitutes a major lead to allow learning-capable neuromorphic systems and comes with strong theoretical guarantees. Equilibrium propagation operates in two phases, during which the network is let to evolve freely and then "nudged" toward a target; the weights of the network are then updated based solely on the states of the neurons that they connect. The weight updates of Equilibrium Propagation have been shown mathematically to approach those provided by Backpropagation Through Time (BPTT), the mainstream approach to train recurrent neural networks, when nudging is performed with infinitely small strength. In practice, however, the standard implementation of Equilibrium Propagation does not scale to visual tasks harder than MNIST. In this work, we show that a bias in the gradient estimate of equilibrium propagation, inherent in the use of finite nudging, is responsible for this phenomenon and that canceling it allows training deep convolutional neural networks. We show that this bias can be greatly reduced by using symmetric nudging (a positive nudging and a negative one). We also generalize Equilibrium Propagation to the case of cross-entropy loss (by opposition to squared error). As a result of these advances, we are able to achieve a test error of 11.7% on CIFAR-10, which approaches the one achieved by BPTT and provides a major improvement with respect to the standard Equilibrium Propagation that gives 86% test error. We also apply these techniques to train an architecture with unidirectional forward and backward connections, yielding a 13.2% test error. These results highlight equilibrium propagation as a compelling biologically-plausible approach to compute error gradients in deep neuromorphic systems.

Keywords: equilibrium propagation, energy based models, biologically plausible deep learning, neuromorphic computing, on-chip learning, deep convolutional neural network, learning algorithms

OPEN ACCESS

Edited by:
Melika Payvand,
ETH Zurich, Switzerland

Reviewed by:
Christian Pehle,
Heidelberg University, Germany
Bruno Umbria Pedroni,
University of California, San Diego, United States

***Correspondence:**
Axel Laborieux
axel.laborieux@c2n.upsaclay.fr
Maxence Ernoult
ernoultm@mila.quebec

***Present address:**
Benjamin Scellier,
Google, Zurich, Switzerland

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *25 November 2020*
Accepted: *26 January 2021*
Published: *18 February 2021*

Citation:
Laborieux A, Ernoult M, Scellier B, Bengio Y, Grollier J and Querlioz D (2021) Scaling Equilibrium Propagation to Deep ConvNets by Drastically Reducing Its Gradient Estimator Bias. Front. Neurosci. 15:633674. doi: 10.3389/fnins.2021.633674

1. INTRODUCTION

How synapses in hierarchical neural circuits are adjusted throughout learning a task remains a challenging question called the credit assignment problem (Richards et al., 2019). Equilibrium Propagation (EP) (Scellier and Bengio, 2017) provides a biologically plausible solution to this problem in artificial neural networks. EP is an algorithm for convergent recurrent neural networks

(RNNs) which, by definition, are given a static input and whose recurrent dynamics converge to a steady state corresponding to the prediction of the network. EP proceeds in two phases, bringing the network to a first steady state, then nudging the output layer of the network toward a ground-truth target until reaching a second steady state. During the second phase of EP, the perturbation originating from the output layer propagates forward in time to upstream layers, creating local error signals that match exactly those that are computed by Backpropagation Through Time (BPTT), the canonical approach for training RNNs (Ernoult et al., 2019). We refer to Scellier and Bengio (2019) for a comparison between EP and recurrent backpropagation (Almeida, 1987; Pineda, 1987). Owing to this strong theoretical guarantee, EP can provide leads for understanding biological learning (Lillicrap et al., 2020). Moreover, the spatial locality of the learning rule prescribed by EP and the possibility to make it also local in time (Ernoult et al., 2020) is highly attractive for designing energy-efficient neuromorphic hardware implementations of gradient-based learning algorithms (Ernoult et al., 2020; Foroushani et al., 2020; Ji and Gross, 2020; Kendall et al., 2020; Martin et al., 2020; Zoppo et al., 2020).

To meet these expectations, however, EP should be able to scale to complex tasks. Until now, works on EP (Scellier and Bengio, 2017; O'Connor et al., 2018, 2019; Ernoult et al., 2019, 2020) limited their experiments to the MNIST classification task and shallow network architectures. Despite the theoretical guarantees of EP, the literature suggests that no implementation of EP has thus far succeeded to match the performance of standard deep learning approaches to train deep networks on hard visual tasks. This problem is even more challenging when using a more bio-plausible topology where the synaptic connections of the network are unidirectional: existing proposals of EP in this situation (Scellier et al., 2018; Ernoult et al., 2020) lead to a degradation of accuracy on MNIST compared to standard EP. In this work, we show that performing the second phase of EP with nudging strength of constant sign induces a systematic first order bias in the EP gradient estimate which, once canceled, unlocks the training of deep convolutional neural networks (ConvNets), with bidirectional or unidirectional connections and with performance closely matching that of BPTT on CIFAR-10. We also propose to implement the neural network predictor as an external softmax readout. This modification preserves the local nature of EP and allows us to use the cross-entropy loss, contrary to previous approaches using the squared error loss, and where the predictor takes part in the free dynamics of the system.

Other biologically plausible alternatives to backpropagation (BP) have attempted to scale to hard vision tasks. Bartunov et al. (2018) investigated the use of feedback alignment (Lillicrap et al., 2016) and variants of target propagation (Lecun, 1987; Bengio, 2014) on CIFAR-10 and ImageNet, showing that they perform significantly worse than backpropagation. When the alignment between forward and backward weights is enhanced with extra mechanisms (Akrout et al., 2019), feedback alignment performs better on ImageNet than sign-symmetry (Xiao et al., 2018), where feedback weights are taken to be the sign of the forward weights, and almost as well as backpropagation. However, in feedback alignment and target propagation, the error feedback does not affect the forward neural activity and is instead routed through a distinct backward pathway, an issue that EP avoids. Payeur et al. (2020) proposed a burst-dependent learning rule that also addresses this problem and whose rate-based equivalent, relying on the use of specialized synapses and complex network topology, has been benchmarked against CIFAR-10 and ImageNet. Related works on implicit models (Bai et al., 2019) have shown that training deep networks can be framed as solving a fixed point (steady state) equation, leading to an analytical backward pass. This framework was shown to solve challenging vision tasks (Bai et al., 2020). While the use of a steady state is common with EP, the process to reach the steady state as well as the learning rule are different. In comparison with these approaches, EP offers a minimalistic circuit requirement to handle both inference and gradient computation, which makes it an outstanding candidate for energy-efficient neuromorphic learning hardware design.

More specifically, the contributions of this work are the following:

- We introduce a new method to estimate the gradient of the loss based on three steady states instead of two (section 3.1). This approach enables us to achieve 11.68% test error on CIFAR-10, with 0.6% performance degradation only with respect to BPTT. Conversely, we show that using a nudging strength of constant sign yields 86.64% test error.
- We propose to implement the output layer of the neural network as a softmax readout, which subsequently allows us to optimize the cross-entropy loss function with EP. This method improves the classification performance on CIFAR-10 with respect to the use of the squared error loss and is also closer to the one achieved with BPTT (section 3.2).
- Finally, based on ideas of Scellier et al. (2018) and Kolen and Pollack (1994), we adapt the learning rule of EP for architectures with distinct (unidirectional) forward and backward connections, yielding only 1.5% performance degradation on CIFAR-10 compared to bidirectional connections (section 2.4).

2. BACKGROUND

2.1. Convergent RNNs With Static Input

We consider the setting of supervised learning where we are given an input x (e.g., an image) and want to predict a target y (e.g., the class label of that image). To solve this type of task, Equilibrium Propagation (EP) relies on convergent RNNs, where the input of the RNN at each time step is static and equal to x, and the state s of the neural network converges to a steady-state s_*. EP applies to a wide class of convergent RNNs, where the transition function derives from a scalar primitive[1] Φ (Ernoult et al., 2019). In this situation, the dynamics of a network with parameters θ, usually

[1] In the original version of EP for real-time dynamical systems (Scellier and Bengio, 2017), the dynamics derive from an energy function E, which plays a similar role to the primitive function Φ in the discrete-time setting studied here.

synaptic weights, is given by

$$s_{t+1} = \frac{\partial \Phi}{\partial s}(x, s_t, \theta), \tag{1}$$

where s_t is the state of the RNN at time step t. After the dynamics have converged at some time step T, the network reaches the steady state $s_T = s_*$, which, by definition, satisfies:

$$s_* = \frac{\partial \Phi}{\partial s}(x, s_*, \theta). \tag{2}$$

Formally, the goal of learning is to optimize θ to minimize the loss at the steady state $\mathcal{L}^* = \ell(s_*, y)$, where ℓ is a differentiable cost function. While we did not investigate theoretical guarantees ensuring the convergence of the dynamics, we refer the reader to Scarselli et al. (2008) for sufficient conditions on the transition function to ensure convergence. In practice, we always observe the convergence to a steady-state.

2.2. Training Procedures for Convergent RNNs

2.2.1. Equilibrium Propagation (EP)

Scellier and Bengio (2017) introduced Equilibrium Propagation in the case of real time dynamics. Subsequent work adapted it to discrete-time dynamics, bringing it closer to conventional deep learning (Ernoult et al., 2019). EP consists of two distinct phases. During the first ("free") phase, the RNN evolves according to Equation (1) for T time steps to ensure convergence to a first steady state s_*. During the second ("nudged") phase of EP, a nudging term $-\beta \frac{\partial \ell}{\partial s}$ is added to the dynamics, with β a small scaling factor. Denoting s_0^β, s_1^β, s_2^β... the states during the second phase, the dynamics reads

$$s_0^\beta = s_*, \quad \text{and} \quad \forall t > 0, \quad s_{t+1}^\beta = \frac{\partial \Phi}{\partial s}(x, s_t^\beta, \theta) - \beta \frac{\partial \ell}{\partial s}(s_t^\beta, y). \tag{3}$$

The RNN then reaches a new steady state denoted s_*^β. Scellier and Bengio (2017) proposed the EP learning rule, denoting η the learning rate applied:

$$\Delta\theta = \eta \widehat{\nabla}^{\text{EP}}(\beta), \qquad \text{where}$$
$$\widehat{\nabla}^{\text{EP}}(\beta) \triangleq \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_*^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_*, \theta)\right). \tag{4}$$

They proved that this learning rule performs stochastic gradient descent in the limit $\beta \to 0$:

$$\lim_{\beta \to 0} \widehat{\nabla}^{\text{EP}}(\beta) = -\frac{\partial \mathcal{L}^*}{\partial \theta}. \tag{5}$$

2.2.2. Equivalence of Equilibrium Propagation and Backpropagation Through Time (BPTT)

The convergent RNNs considered by EP can also be trained by Backpropagation Through Time (BPTT). At each BPTT training iteration, the first phase is performed for T time steps until the network reaches the steady state $s_T = s_*$. The loss at the final time step is computed and the gradients are subsequently backpropagated through the computational graph of the first phase, backward in time.

Let us denote $\nabla^{\text{BPTT}}(t)$ the gradient computed by BPTT truncated to the last t time steps $(T-t, \ldots, T)$, which we define formally in **Supplementary Material** (section 1).

A theorem derived by Ernoult et al. (2019), inspired from Scellier and Bengio (2019), shows that, provided convergence in the first phase has been reached after $T - K$ time steps (i.e., $s_{T-K} = s_{T-K+1} = \ldots = s_T = s_*$), the gradients of EP match those computed by BPTT in the limit $\beta \to 0$, in the first K time steps of the second phase for fully connected and convolutional architectures including pooling operations:

$$\forall t = 1, 2, \ldots, K,$$
$$\widehat{\nabla}^{\text{EP}}(\beta, t) \triangleq \frac{1}{\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s_t^\beta, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s_*, \theta)\right) \xrightarrow[\beta \to 0]{} \nabla^{\text{BPTT}}(t). \tag{6}$$

2.3. Convolutional Architectures for Convergent RNNs

A convolutional architecture for convergent RNNs with static input was introduced by Ernoult et al. (2019) and successfully trained with EP on the MNIST dataset. In this architecture, presented in **Figure 1**, we define N^{conv} and N^{fc} the number of convolutional and fully connected layers respectively, and $N^{\text{tot}} \triangleq N^{\text{conv}} + N^{\text{fc}}$. w_{n+1} denotes the weights connecting s^n to s^{n+1}, with $s_0 = x$. To simplify notations, we use distinct operators to differentiate whether w_n is a convolutional layer or a fully connected layer: respectively $\star$ for convolutions and $\cdot$ for linear layers. The primitive function can therefore be defined as:

$$\Phi(x, \{s^n\}) = \sum_{n=0}^{N^{\text{conv}}-1} s^{n+1} \bullet \mathcal{P}\left(w_{n+1} \star s^n\right) + \sum_{n=N^{\text{conv}}}^{N^{\text{tot}}-1} s^{n+1\top} \cdot w_{n+1} \cdot s^n, \tag{7}$$

where $\bullet$ is the Euclidean scalar product generalized to pairs of tensors with same arbitrary dimension, and $\mathcal{P}$ is a pooling operation. Combining Equations (1) and (7), and restricting the space of the state variables to $[0, 1]$, yield the dynamics:

$$\begin{cases} s_{t+1}^n = \sigma\left(\mathcal{P}\left(w_n \star s_t^{n-1}\right) + \tilde{w}_{n+1} \star \mathcal{P}^{-1}\left(s_t^{n+1}\right)\right), 1 \le n \le N^{\text{conv}} \\ s_{t+1}^n = \sigma\left(w_n \cdot s_t^{n-1} + w_{n+1}^\top \cdot s_t^{n+1}\right), N^{\text{conv}} < n < N^{\text{tot}} \end{cases} \tag{8}$$

where σ is an activation function bounded between 0 and 1. Transpose convolution and inverse pooling are respectively defined through the convolution by the flipped kernel $\tilde{w}$ and $\mathcal{P}^{-1}$. Plugging Equation (7) into Equation (4) yields the local learning rule $\Delta\theta_{ij} = \eta(s_{i,*}^\beta s_{j,*}^\beta - s_{i,*} s_{j,*})/\beta$ for a parameter θ_{ij} linking neurons i and j. **Supplementary Material** (section 4) provides the implementation details of this model.

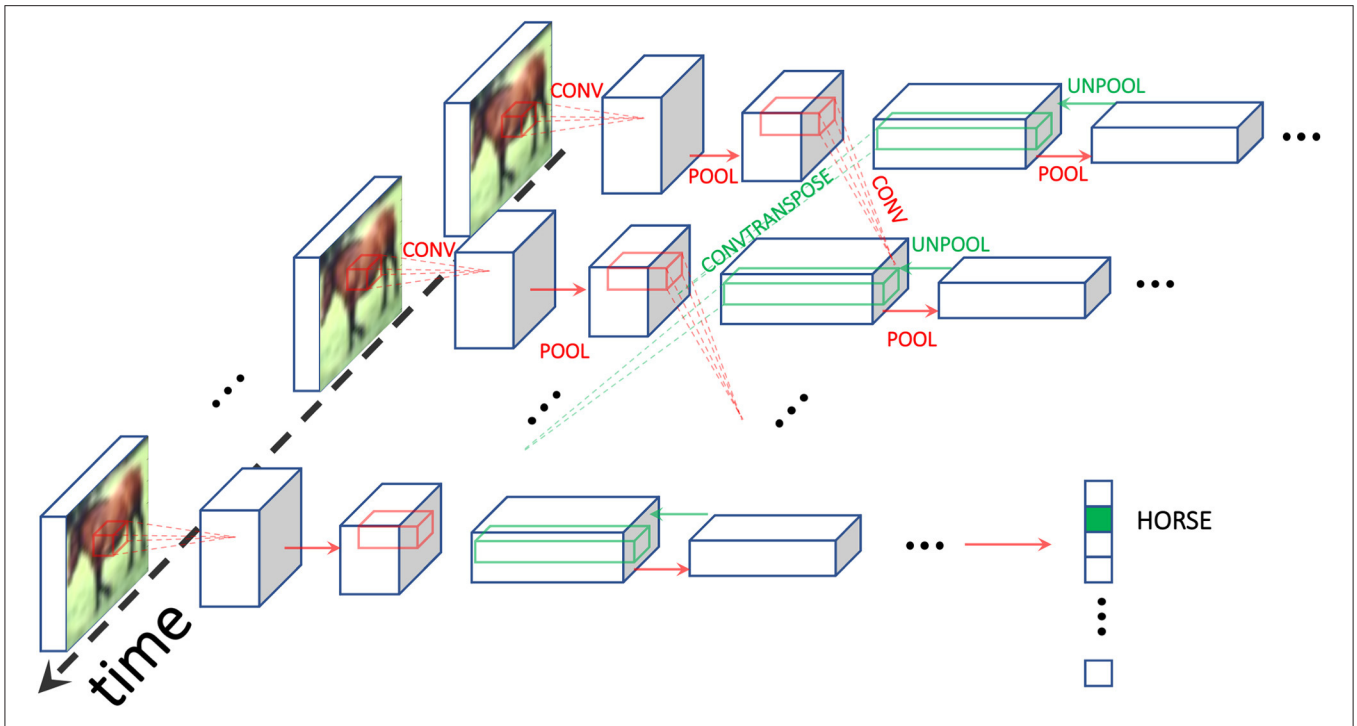


FIGURE 1 | Schematic of the architecture used. We use Equilibrium Propagation (EP) to train a recurrent ConvNet receiving a static input. Red (resp. green) arrows depict forward (resp. backward) operations, with convolutions and transpose convolutions happening through time. At the final time step, the class prediction is carried out. The use of RNNs is inherent in the credit assignment of EP which uses of the temporal variations of the system as error signals for the gradient computation.

2.4. Equilibrium Propagation With Unidirectional Synaptic Connections

We have seen that in the standard formulation of EP, the dynamics of the neural network derive from a function Φ (Equation 1) called the primitive function. This formulation implies the existence of bidirectional synaptic connections between neurons. For better biological plausibility, a more general formulation of EP circumvents this requirement and allows training networks with distinct (unidirectional) forward and backward connections (Scellier et al., 2018; Ernoult et al., 2020). This feature is also desirable for hardware implementations of EP. Although some analog implementations of EP naturally lead to symmetric weights (Kendall et al., 2020), neural networks with unidirectional weights are in general easier to implement in neuromorphic hardware.

In this setting, the dynamics of Equation (1) is changed into the more general form:

$$s_{t+1} = F(x, s_t, \theta), \tag{9}$$

and the conventionally proposed learning rule reads:

$$\Delta\theta = \eta \widehat{\nabla}^{\mathrm{VF}}(\beta), \qquad \text{where}$$
$$\widehat{\nabla}^{\mathrm{VF}}(\beta) \triangleq \frac{1}{\beta}\frac{\partial F}{\partial \theta}(x, s_*, \theta)^\top \cdot \left(s_*^\beta - s_*\right), \tag{10}$$

where VF stands for Vector Field (Scellier et al., 2018). If the transition function F derives from a primitive function Φ (i.e., if $F = \frac{\partial \Phi}{\partial s}$), then $\widehat{\nabla}^{\mathrm{VF}}(\beta)$ is equal to $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ in the limit $\beta \to 0$ (i.e., $\lim_{\beta\to 0} \widehat{\nabla}^{\mathrm{VF}}(\beta) = \lim_{\beta\to 0} \widehat{\nabla}^{\mathrm{EP}}(\beta)$).

3. IMPROVING EP TRAINING

We have seen in Equation (6) that the temporal variations of the network over the second phase of EP exactly compute BPTT gradients in the limit $\beta \to 0$. This result appears to underpin the use of two phases as a fundamental element of EP, but is it really the case? In this section, we revisit EP as a gradient estimation procedure and propose an implementation in three phases instead of two. Moreover, we show how to optimize the cross-entropy loss function with EP. Combining these two new techniques enabled us to achieve the best performance on CIFAR-10 by EP, on architectures with bidirectional and unidirectional forward and backward connections (section 4).

3.1. Reducing Bias and Variance in the Gradient Estimate of the Loss Function

In the foundational work on EP, Scellier and Bengio (2017) demonstrate that:

$$\left.\frac{d}{d\beta}\right|_{\beta=0} \frac{\partial \Phi}{\partial \theta}(x, s_*^\beta, \theta) = -\frac{\partial \mathcal{L}^*}{\partial \theta}. \tag{11}$$

The traditional implementation of EP evaluates the left-hand side of Equation (11) using the estimate $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ with two points $\beta = 0$ and $\beta > 0$, thereby calling for the need of two

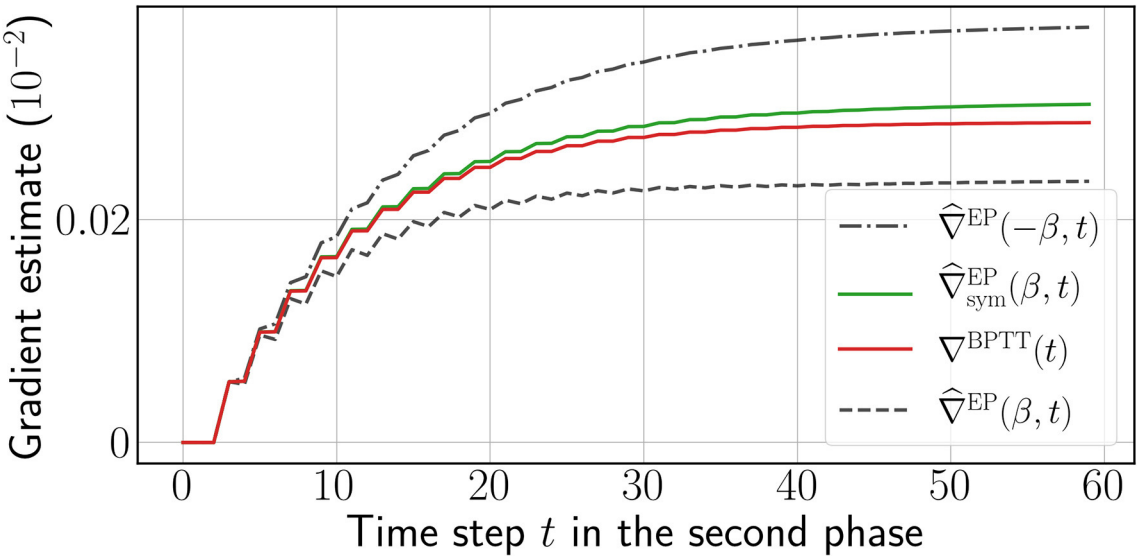


FIGURE 2 | One-sided EP gradient estimate for opposite values of $\beta = 0.1$ (black dashed curves), symmetric EP gradient estimate (green curve), and reference gradients computed by BPTT (red curve) computed over the second phase, for a single weight chosen at random. The time step t is defined for BPTT and EP according to Equation (6). More instances can be found in **Supplementary Material** (section 6).

phases—the free phase and the nudged phase. However, the use of $\beta > 0$ in practice induces a systematic first order bias in the gradient estimation provided by EP. In order to eliminate this bias, we propose to perform a third phase with $-\beta$ as the nudging factor, keeping the first and second phases unchanged. We then estimate the gradient of the loss using the following symmetric difference estimate:

$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) \triangleq \frac{1}{2\beta}\left(\frac{\partial \Phi}{\partial \theta}(x, s^{\beta}_{*}, \theta) - \frac{\partial \Phi}{\partial \theta}(x, s^{-\beta}_{*}, \theta)\right). \tag{12}$$

Indeed, under mild assumptions on the function $\beta \mapsto \frac{\partial \Phi}{\partial \theta}(x, s^{\beta}_{*}, \theta)$, we can show that, as $\beta \to 0$:

$$\frac{\widehat{\nabla}^{\mathrm{EP}}(\beta) + \widehat{\nabla}^{\mathrm{EP}}(-\beta)}{2} = -\frac{\partial \mathcal{L}^{*}}{\partial \theta} + O(\beta^2), \tag{13}$$

$$\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta) = -\frac{\partial \mathcal{L}^{*}}{\partial \theta} + O(\beta^2). \tag{14}$$

This result is proved in Lemma 2 of the **Supplementary Material** (section 2). Equation (13) shows that the estimate $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ possesses a first-order error term in β which the symmetric estimate $\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta)$ eliminates (Equation 14). Note that the first-order term of $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ could also be canceled out on average by choosing the sign of β at random with even probability (so that $\mathbb{E}(\beta) = 0$, see Algorithm 1 of the **Supplementary Material**, section 3.1). Although not explicitly stated in this purpose, the use of such randomization has been reported in some earlier publications on the MNIST task (Scellier and Bengio, 2017; Ernoult et al., 2020). However, in this work, we show that this method exhibits high variance in the training procedure.

We call $\widehat{\nabla}^{\mathrm{EP}}(\beta)$ and $\widehat{\nabla}^{\mathrm{EP}}_{\mathrm{sym}}(\beta)$ the one-sided and symmetric EP gradient estimates, respectively. The qualitative difference between these estimates is depicted in **Figure 2**, and the full training procedure is depicted in Algorithm 2 of the **Supplementary Material** (section 3.2).

Finally, this technique can also be applied to the Vector Field setting introduced in section 2.4 and we denote $\widehat{\nabla}^{\mathrm{VF}}_{\mathrm{sym}}(\beta)$ the resulting symmetric estimate—see the **Supplementary Material** (section 4.3) for details.

3.2. Changing the Loss Function

We also introduce a novel architecture to optimize the cross-entropy loss with EP, narrowing the gap with conventional deep learning architectures for classification tasks. In the next paragraph, we denote $\widehat{y}$ the set of neurons that carries out the prediction of the neural network.

3.2.1. Squared Error Loss Function

Previous implementations of EP used the squared error loss. Using this loss function for EP is natural, as in this setting, the output $\widehat{y}$ is viewed as a part of s (the state variable of the network), which can influence the state of the network through bidirectional synaptic connections (see **Figure 3**). Moreover, the nudging term in this case can be physically interpreted since it reads as an elastic force. The state of the network is of the form $s = (s^1, \ldots, s^N, \widehat{y})$ where $h = (s^1, \ldots, s^N)$ represent the "hidden layers," and the corresponding cost function is

$$\ell(\widehat{y}, y) = \frac{1}{2}\left\|\widehat{y} - y\right\|^2. \tag{15}$$

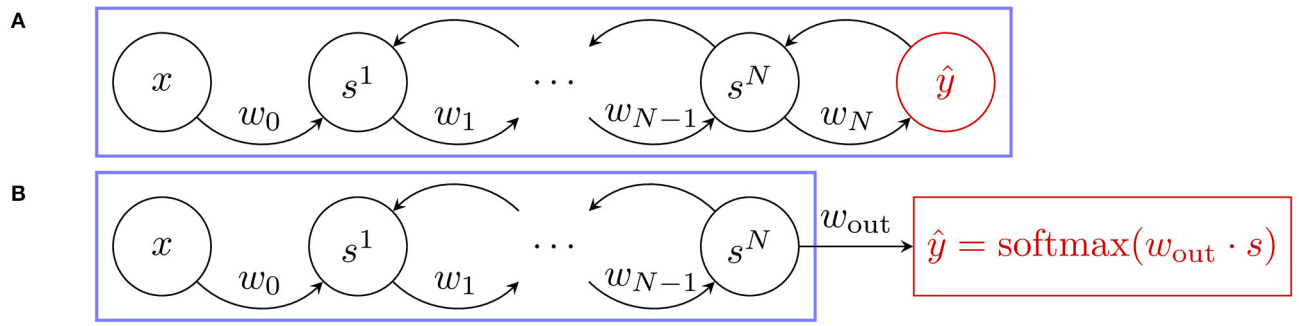


FIGURE 3 | Free dynamics of the architectures used for the two loss functions where the blue frame delimits the system. **(A)** Squared Error loss function. The usual setting where the predictor $\hat{y}$ (in red) takes part in the free dynamics of the neural network through bidirectional synaptic connections. **(B)** Cross-entropy loss function. The new approach proposed in this work where the predictor $\hat{y}$ (also in red) is no longer involved in the system free dynamics and is implemented as a softmax readout.

The second phase dynamics of the hidden state and output layer given by Equation (3) read, in this context:

$$\begin{aligned} h^\beta_{t+1} &= \frac{\partial \Phi}{\partial h}(x, h^\beta_t, \widehat{y}^\beta_t, \theta), \\ \widehat{y}^\beta_{t+1} &= \frac{\partial \Phi}{\partial \widehat{y}}(x, h^\beta_t, \widehat{y}^\beta_t, \theta) + \beta\ (y - \widehat{y}^\beta_t). \end{aligned} \tag{16}$$

3.2.2. Softmax Readout, Cross-Entropy Loss Function

In this paper, we propose an alternative approach, where the output $\widehat{y}$ is not a part of the state variable s but is instead implemented as a read-out (see **Figure 3**), which is a function of s and of a weight matrix w_{out} of size $\dim(y) \times \dim(s)$. In practice, w_{out} reads out the last convolutional layer. At each time step t we define:

$$\widehat{y}_t = \text{softmax}(w_{\text{out}} \cdot s_t). \tag{17}$$

The cross-entropy cost function associated with the softmax readout is then:

$$\ell(s, y, w_{\text{out}}) = -\sum_{c=1}^{C} y_c \log(\text{softmax}_c(w_{\text{out}} \cdot s)). \tag{18}$$

Using $\frac{\partial \ell}{\partial s}(s, y, w_{\text{out}}) = w_{\text{out}}^\top \cdot \big(\text{softmax}(w_{\text{out}} \cdot s) - y\big)$, the second phase dynamics given by Equation (3) read in this context:

$$s^\beta_{t+1} = \frac{\partial \Phi}{\partial s}(x, s^\beta_t, \theta) + \beta\ w_{\text{out}}^\top \cdot \left(y - \widehat{y}^\beta_t\right). \tag{19}$$

Note here that the loss $\mathcal{L}^* = \ell(s_*, y, w_{\text{out}})$ also depends on the parameter w_{out}. The **Supplementary Material** (section 4.2.2) provides the learning rule applied to w_{out}.

3.3. Changing the Learning Rule of EP With Unidirectional Synaptic Connections

In the case of architectures with unidirectional connections, applying the traditional EP learning rule directly, as given by Equation (10), prescribes different forward and backward weights updates, resulting in significantly different forward and backward weights throughout learning. However, the theoretical equivalence between EP and BPTT only holds for bidirectional connections. Until now, training experiments of unidirectional weights EP have performed worse than bidirectional weights EP (Ernoult et al., 2020). In this work, therefore, we tailor a new learning rule for unidirectional weights, described in detail the **Supplementary Material** (section 4.3), where the forward and backward weights undergo the same weight updates, incorporating an equal leakage term. This way, forward and backward weights, although they are independently initialized, naturally converge to identical values throughout the learning process. A similar methodology, adapted from Kolen and Pollack (1994), has been shown to improve the performance of Feedback Alignment in Deep ConvNets (Akrout et al., 2019).

Assuming general dynamics of the form of Equation (9), we distinguish forward connections θ_{f} from backward connections θ_{b} so that $\theta = \{\theta_{\text{f}}, \theta_{\text{b}}\}$, with θ_{f} and θ_{b} having same dimension. Assuming a first phase, a second phase with $\beta > 0$ and a third phase with $-\beta$, we define:

$$\forall \text{i} \in \{\text{f}, \text{b}\},$$
$$\overline{\nabla^{\text{VF}}_{\theta_{\text{i}}}}(\beta) = \frac{1}{2\beta}\left(\frac{\partial F}{\partial \theta_{\text{i}}}^\top (x, s^\beta_*, \theta) \cdot s^\beta_* - \frac{\partial F}{\partial \theta_{\text{i}}}^\top (x, s^{-\beta}_*, \theta) \cdot s^{-\beta}_*\right) \tag{20}$$

and we propose the following update rules:

$$\begin{cases} \Delta\theta_{\text{f}} = \eta\left(\widehat{\nabla}^{\text{KP}-\text{VF}}_{\text{sym}}(\beta) - \lambda\theta_{\text{f}}\right) \\ \Delta\theta_{\text{b}} = \eta\left(\widehat{\nabla}^{\text{KP}-\text{VF}}_{\text{sym}}(\beta) - \lambda\theta_{\text{b}}\right) \end{cases}, \quad \text{with} \quad \widehat{\nabla}^{\text{KP}-\text{VF}}_{\text{sym}}(\beta) = \frac{1}{2}(\overline{\nabla^{\text{VF}}_{\theta_{\text{f}}}}(\beta) + \overline{\nabla^{\text{VF}}_{\theta_{\text{b}}}}(\beta)) \tag{21}$$

where η is the learning rate and λ a leakage parameter. The estimate $\widehat{\nabla}^{\text{KP}-\text{VF}}_{\text{sym}}(\beta)$ can be thought of a generalization of Equation (12), as highlighted in the **Supplementary Material** (section 4.3) with an explicit application of Equation (21) to a ConvNet. In the case of a fully connected layer, both terms in

TABLE 1 | Hyper-parameters used for the CIFAR-10 experiments.

Hyper-parameter	Squared error	Cross-entropy
T	250	250
K	30	25
β	0.5	1.0
Batch size	128	128
Initial learning rates (Layer-wise)	0.25 - 0.15 - 0.1 - 0.08 - 0.05	0.25 - 0.15 - 0.1 - 0.08 - 0.05
Final learning rates	10^{-5}	10^{-5}
Weight decay (All layers)	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
Momentum	0.9	0.9
Epoch	120	120
Cosine annealing Decay time (epochs)	100	100

TABLE 2 | Performance comparison on CIFAR-10 between BPTT and EP with several gradient estimation schemes.

Loss function	EP gradient estimate	EP error (%) Test	EP error (%) Train	BPTT error (%) Test	BPTT error (%) Train
Squared error	2-Phase/$\hat{\nabla}^{EP}$	86.64 (5.82)	84.90	11.10 (0.21)	3.69
	Random Sign	21.55 (20.00)	20.01		
	3-Phase/$\hat{\nabla}^{EP}_{sym}$	12.45 (0.18)	7.83		
Cross-Ent.	3-Phase/$\hat{\nabla}^{EP}_{sym}$	**11.68** (**0.17**)	**4.98**	11.12 (0.21)	2.19
Cross-Ent. (Dropout)	3-Phase/$\hat{\nabla}^{EP}_{sym}$	11.87 (0.29)	6.46	10.72 (0.06)	2.99
Cross-Ent.	3-Phase/$\widehat{\nabla}^{VF}_{sym}$	75.47 (4.72)	78.04	9.46 (0.17)	0.80
	3-Phase/$\widehat{\nabla}^{KP-VF}_{sym}$	**13.15** (**0.49**)	8.87		

Note that the different gradient estimates only apply to EP. We indicate over five trials the mean and standard deviation in parenthesis for the test error, and the mean train error.

the sum in the right hand side of Equation (21) are equal: $\partial F/\partial\theta_i$ only depends on the neuron activations and not on θ_i, in the same way, as seen at the end of section 2.3, that Equation (8) yields a fully local learning rule. The case of convolutional layers is a little more subtle, due to presence of the maximum pooling operations. The forward weights are involved in a pooling operation while the backward weights are involved in an unpooling operation. However, for the parameter update to be the same, the pooling and unpooling operations need to share information regarding the indices of maxima. Therefore, there is indeed a need for information transfer between backward and forward parameters, but this exchange is limited to the index of the maximum identified in the maximum pooling operation (this can be seen from Equation 24).

4. RESULTS

In this section, we implement EP with the modifications described in section 3 and successfully train deep ConvNets on the CIFAR-10 vision task (Krizhevsky et al., 2009). The convolutional architecture used consists of four 3×3 convolutional layers of respective feature maps 128–256–512–512. We use a stride of one for each convolutional layer, and zero-padding of one for each layer except for the last layer. Each layer is followed by a 2×2 Max Pooling operation with a stride of two. The resulting flattened feature vector is of size 512. The weights are initialized using the default initialization of PyTorch, which is the uniform Kaiming initialization of He et al. (2015). The data is normalized and augmented with random horizontal flips and random crops. The training is performed with stochastic gradient descent with momentum and weight decay. We use the learning rate scheduler introduced by Loshchilov and Hutter (2016) to speed up convergence.

The hyper-parameters are reported in **Table 1**. All experiments are performed using PyTorch 1.4.0. (Paszke et al., 2017). The simulations were carried across several servers consisting of 14 Nvidia GeForce RTX 2080 TI GPUs in total. Each run was performed on a single GPU for an average run time of 2 days.

4.1. ConvNets With Bidirectional Connections

We first consider the bidirectional weight setting of section 2.3. In **Table 2**, we compare the performance achieved by the ConvNet for each EP gradient estimate introduced in section 3.1 with the performance achieved by BPTT.

The one-sided gradient estimate leads to unstable training behavior where the network is unable to fit the data, as shown by the purple curve of **Figure 4A**, with 86.64% test error on CIFAR-10. When the bias in the gradient estimate is averaged out by choosing at random the sign of β during the second phase, the average test error over five runs goes down to 21.55% (see **Table 2**). However, one run among the five yielded instability similar to the one-sided estimate, whereas the four remaining runs lead to 12.61% test error and 8.64% train error. This method for estimating the loss gradient thus presents high variance—further experiments shown in the **Supplementary Material** (section 4.4) confirm this trend.

Conversely, the three-phase symmetric estimate enables EP to consistently reach 12.45% test error, with only 1.35% degradation with respect to BPTT (see **Figure 4A**). Therefore, removing the first-order error term in the gradient estimate is critical for scaling to deeper architectures. Proceeding to this end deterministically (with three phases) rather than stochastically (with a randomized nudging sign) appears more reliable.

The results of **Table 2** also show that the readout scheme introduced in section 3.2 to optimize the cross-entropy loss function enables EP to narrow the performance gap with BPTT down to 0.56% while outperforming the Squared Error setting by 0.77%. However, we observe that the test errors reached by BPTT are similar for the squared error and the cross-entropy loss. The fact that only EP benefits from the cross-entropy loss is due to the output not being part of the dynamics, which reduces the number of layers following the dynamics by one.

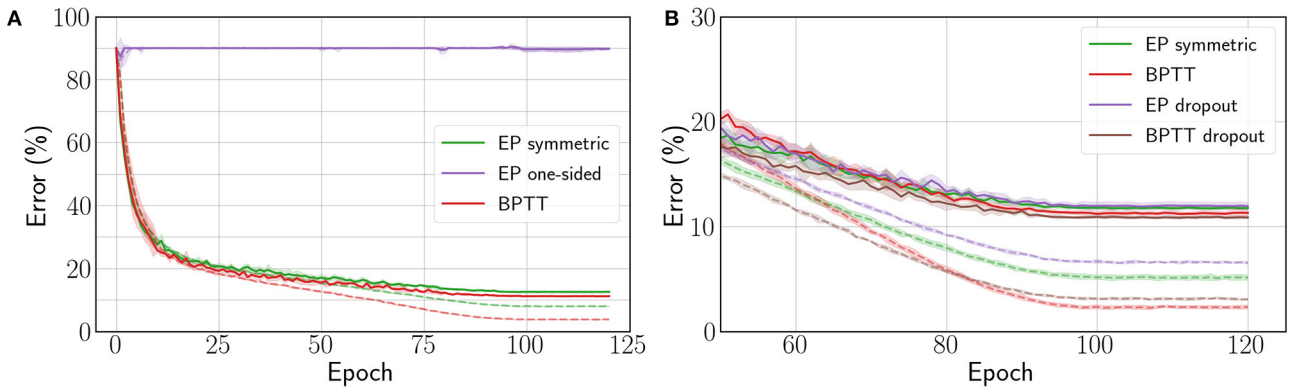


FIGURE 4 | (A) Train (dashed) and test (solid) errors on CIFAR-10 with the Squared Error loss function. **(B)** Train (dashed) and test (solid) errors on CIFAR-10 with the Cross-Entropy loss function. The curves are averaged over 5 runs and shadows stand for $\pm 1 \times$ standard deviation. The change in error rate around epochs 85–90 is due to the end of the learning rate scheduler decay phase (Cosine annealing).

We also adapted dropout (Srivastava et al., 2014) to convergent RNNs (see the **Supplementary Material**, section 4.5 for implementation details) to see if the performance could be improved further. However, we can observe from **Table 2** and **Figure 4B** that contrary to BPTT, the EP test error is not improved by adding a 0.1 dropout probability in the neuron layer after the convolutions.

4.2. ConvNets With Unidirectional Connections

We now present the accuracy achieved by EP when the architecture uses distinct forward and backward weights, using a softmax readout. For this architecture, the backward weights are defined for all convolutional layers, except the first convolutional layer connected to the static input. The forward and backward weights are initialized randomly and independently at the beginning of training. The backward weights have no bias contrary to their forward counterparts. The hyper-parameters such as learning rate, weight decay and momentum are shared between forward and backward weights.

As seen in **Table 2**, we find that the estimate $\widehat{\nabla}^{\mathrm{VF}}_{\mathrm{sym}}(\beta)$ leads to a poor performance with 75.47% test-error. We concomitantly observed that forward and backward weight did not align well, as shown by the dashed curves in **Figure 5**. Conversely, when using our new estimate $\widehat{\nabla}^{\mathrm{KP-VF}}_{\mathrm{sym}}(\beta)$ defined in section 3.3, a good performance is recovered with only 1.5% performance degradation with respect to the architecture with bidirectional connections, and a 3% degradation with respect to BPTT (see **Table 2**). The discrepancy between the BPTT test error achieved by the architecture with bidirectional (11.12%) and unidirectional (9.46%) connections comes from the increase in parameters provided by backward weights. As observed in the weight alignment curves in **Figure 5**, forward and backward weights are well-aligned by epoch 50 when using the new estimate. These results suggest that enhancing forward and backward weights alignment can help EP training in deep ConvNets.

5. DISCUSSION

Our results unveil the necessity, in order to scale EP to deep convolutional neural networks on hard visual tasks, to compute better gradient estimates than the conventional implementation of EP. This traditional implementation incorporates a first order gradient estimate bias, which severely impedes the training of deep architectures. Conversely, we saw that the three-phase EP proposed here removes this bias and brings EP performance on CIFAR-10 close to the one achieved by BPTT. Additionally, our new technique to train EP with softmax readout reduces the gap between EP and BPTT further down to 0.56%, while maintaining the locality of the learning rule of all parameters.

While the test accuracy of BPTT and our adapted EP are very close, we can notice in **Table 2** that BPTT fits the training data better than EP by at least 2.8%. Also, the introduction of dropout improves BPTT performance, while it has no significant effect on the test accuracy of EP. These two insights combined suggest that EP training may have a self-regularizing effect applied throughout the network, similar to the effects of dropout. We hypothesize this effect to be not only due to the residual estimation bias of the BPTT gradients by EP, but also to an additional inherent error term due to the fact that in practice, the fixed point is approached with a precision that depends on the number of time steps at inference. While the exactness of the fixed point is crucial for EP, BPTT computes exact gradients regardless of whether the fixed point is not exactly reached.

We also saw that employing a new training technique that still preserves the spatial locality of EP computations—and therefore its suitability for neuromorphic implementations—our results extend to the case of an architecture with distinct forward and backward synaptic connections.

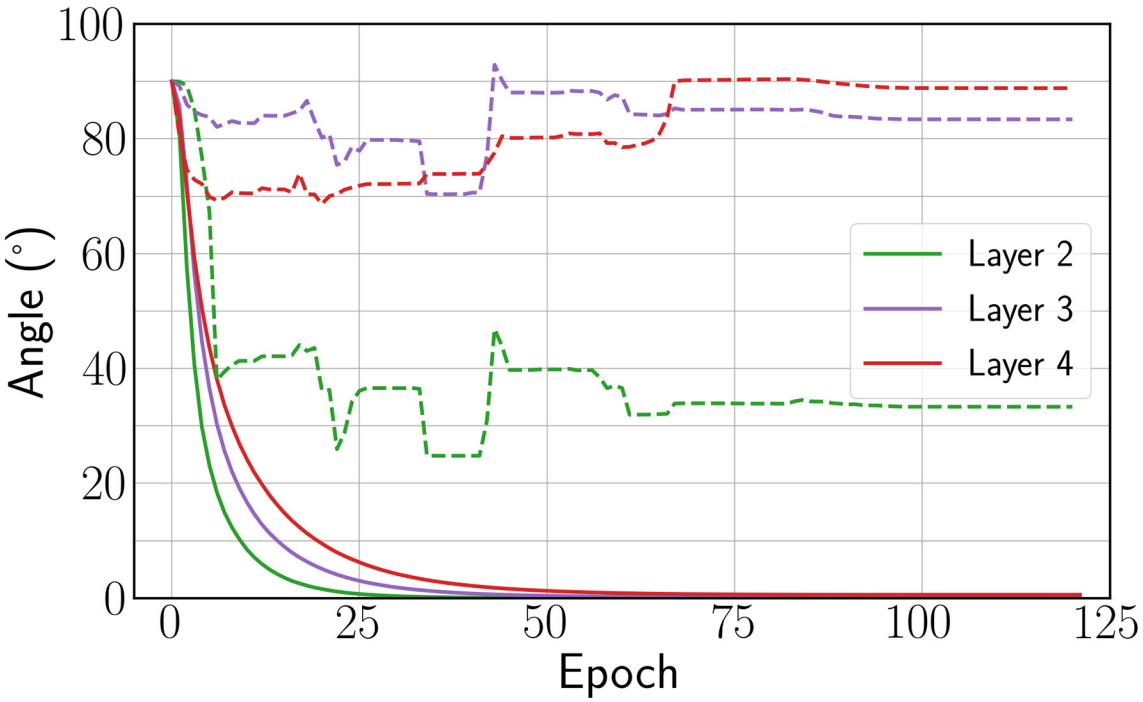


FIGURE 5 | Angle between forward and backward weights for the new estimate $\widehat{\nabla}_{\text{sym}}^{\text{KP}-\text{VF}}$ introduced (solid) and $\widehat{\nabla}_{\text{sym}}^{\text{VF}}$ (dashed). The angle is not defined for the first layer because the input layer is clamped.

We only observe a 1.5% performance degradation with respect to the bidirectional architecture. This result demonstrates the scalability of EP without the biologically implausible requirement of a bidirectional connectivity pattern.

Our three steady states-based gradient estimate comes at a computational cost with regards to the conventional EP implementation, as an additional phase is needed. Even though the steady state of the free phase s_* is not used to compute the gradient estimate in Equation (12), we experimentally found that s_* is needed as a starting point for the second and third phases. In terms of simulation time, EP is 20% slower than BPTT due to the dynamics performed in second and third phases. However, the memory requirement to store the computational graph unfolded in time in the case of BPTT far outweighs the memory needed by EP, which consists only of the steady states reached by the neurons.

The full potential of EP will be best envisioned on neuromorphic hardware. Multiple works have investigated the implementation of EP on such systems (Ernoult et al., 2019, 2020; Foroushani et al., 2020; Ji and Gross, 2020; Zoppo et al., 2020), in both rate based (Kendall et al., 2020) and spiking approaches (Martin et al., 2020). Most of these approaches employ analog circuits that exploit device physics to implement the dynamics of EP intrinsically. The spatially local nature of EP computations, on top of its connection with physical equations, make this mapping between EP and neuromorphic hardware natural. Our prescription to run two nudging phases with opposite nudging strengths could be implemented naturally in neuromorphic systems. In fact, the use of differential operation to cancel inherent biases is a technique widely used in electronics, and in neuromorphic computing in particular (Hirtzlin et al., 2019). Overall, our work provides evidence that EP is a compelling approach to scale neuromorphic on-chip training to real-world tasks in a fully local fashion.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are publicly available. This data can be found here: https://github.com/Laborieux-Axel/Equilibrium-Propagation.

AUTHOR CONTRIBUTIONS

AL developed the PyTorch code for the project and performed the simulations. ME supervised the work, helped debug the code, guided hyperparameter search, and designed the experiments with unidirectional connections. BS proposed the ideas of unbiasing the gradient estimate and of using a softmax readout. DQ, JG, and YB provided additional guidance and support. All

authors participated in data analysis, discussed the results, and co-edited the manuscript.

FUNDING

This work was supported by European Research Council Starting Grant NANOINFER (reference: 715872), European Research Council Grant bioSPINspired (reference: 682955), CIFAR, NSERC, and Samsung.

ACKNOWLEDGMENTS

The authors would like to thank Thomas Fischbacher for useful feedback and discussions.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.633674/full#supplementary-material

REFERENCES

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., and Tweed, D. B. (2019). "Deep learning without weight transport," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 974–982.

Almeida, L. B. (1987). "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *Proceedings of the IEEE First International Conference on Neural Networks (San Diego, CA), Vol. II* (Piscataway, NJ: IEEE), 609–618.

Bai, S., Kolter, J. Z., and Koltun, V. (2019). "Deep equilibrium models," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 690–701.

Bai, S., Koltun, V., and Kolter, J. Z. (2020). Multiscale deep equilibrium models. *arXiv preprint arXiv:2006.08656*.

Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. (2018). "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 9368–9378.

Bengio, Y. (2014). How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*.

Ernoult, M., Grollier, J., Querlioz, D., Bengio, Y., and Scellier, B. (2019). "Updates of equilibrium prop match gradients of backprop through time in an RNN with static input," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 7081–7091.

Ernoult, M., Grollier, J., Querlioz, D., Bengio, Y., and Scellier, B. (2020). Equilibrium propagation with continual weight updates. *arXiv preprint arXiv:2005.04168*.

Foroushani, A. N., Assaf, H., Noshahr, F. H., Savaria, Y., and Sawan, M. (2020). "Analog circuits to accelerate the relaxation process in the equilibrium propagation algorithm," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (Séville), 1–5. doi: 10.1109/ISCAS45731.2020.9181250

He, K., Zhang, X., Ren, S., and Sun, J. (2015). "Delving deep into rectifiers: surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago), 1026–1034. doi: 10.1109/ICCV.2015.123

Hirtzlin, T., Bocquet, M., Penkovsky, B., Klein, J.-O., Nowak, E., Vianello, E., et al. (2019). Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Front. Neurosci.* 13:1383. doi: 10.3389/fnins.2019.01383

Ji, Z., and Gross, W. (2020). "Towards efficient on-chip learning using equilibrium propagation," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (Séville), 1–5. doi: 10.1109/ISCAS45731.2020.9180548

Kendall, J., Pantone, R., Manickavasagam, K., Bengio, Y., and Scellier, B. (2020). Training end-to-end analog neural networks with equilibrium propagation. *arXiv preprint arXiv:2006.01981*.

Kolen, J. F., and Pollack, J. B. (1994). "Backpropagation without weight transport," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94), Vol. 3* (Orlando, FL), 1375–1380. doi: 10.1109/ICNN.1994.374486

Krizhevsky, A., Hinton, G., et al. (2009). *Learning Multiple Layers of Features From Tiny Images*. Available online at: https://www.semanticscholar.org/paper/Learning-Multiple-Layers-of-Features-from-Tiny-Krizhevsky/5d90f06bb70a0a3dced62413346235c02b1aa086

Lecun, Y. (1987). *Modeles connexionnistes de l'apprentissage (connectionist learning models)* (Ph.D. thesis). IAAI Laboratory, Paris, France.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* 7, 1–10. doi: 10.1038/ncomms13276

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nat. Rev. Neurosci.* 21, 335–346. doi: 10.1038/s41583-020-0277-3

Loshchilov, I., and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Martin, E., Ernoult, M., Laydevant, J., Li, S., Querlioz, D., Petrisor, T., and Grollier, J. (2020). Eqspike: spike-driven equilibrium propagation for neuromorphic implementations. *arXiv preprint arXiv:2010.07859*.

O'Connor, P., Gavves, E., and Welling, M. (2018). "Initialized equilibrium propagation for backprop-free training" in *International Conference on Learning Representations 2019*.

O'Connor, P., Gavves, E., and Welling, M. (2019). "Training a spiking neural network with equilibrium propagation," in *The 22nd International Conference on Artificial Intelligence and Statistics* (Montreal, QC), 1516–1523.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in pytorch," in *NeurIPS 2017 Workshop Autodiff Decision Program*.

Payeur, A., Guerguiev, J., Zenke, F., Richards, B., and Naud, R. (2020). Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv [Preprint]*. doi: 10.1101/2020.03.30.015511

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* 59, 2229–2232. doi: 10.1103/PhysRevLett.59.2229

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., et al. (2019). A deep learning framework for neuroscience. *Nat. Neurosci.* 22, 1761–1770. doi: 10.1038/s41593-019-0520-2

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Trans. Neural Netw.* 20, 61–80. doi: 10.1109/TNN.2008.2005605

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11:24. doi: 10.3389/fncom.2017.00024

Scellier, B., and Bengio, Y. (2019). Equivalence of equilibrium propagation and recurrent backpropagation. *Neural Comput.* 31, 312–329. doi: 10.1162/neco_a_01160

Scellier, B., Goyal, A., Binas, J., Mesnard, T., and Bengio, Y. (2018). Generalization of equilibrium propagation to vector field dynamics. *arXiv preprint arXiv:1808.04873*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1929–1958.

Xiao, W., Chen, H., Liao, Q., and Poggio, T. (2018). Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*.

Zoppo, G., Marrone, F., and Corinto, F. (2020). Equilibrium propagation for memristor-based recurrent neural networks. *Front. Neurosci.* 14:240. doi: 10.3389/fnins.2020.00240

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 19 May 2021
doi: 10.3389/fnins.2021.664208

μBrain: An Event-Driven and Fully Synthesizable Architecture for Spiking Neural Networks

Jan Stuijt, Manolis Sifalakis, Amirreza Yousefzadeh and Federico Corradi**

Ultra-Low-Power Systems for Internet of Things (IoT), Stichting Interuniversitair Micro-Elektronica Centrum (IMEC) Nederland, Eindhoven, Netherlands

The development of brain-inspired neuromorphic computing architectures as a paradigm for Artificial Intelligence (AI) at the edge is a candidate solution that can meet strict energy and cost reduction constraints in the Internet of Things (IoT) application areas. Toward this goal, we present μBrain: the first digital yet fully event-driven without clock architecture, with co-located memory and processing capability that exploits event-based processing to reduce an always-on system's overall energy consumption (μW dynamic operation). The chip area in a 40 nm Complementary Metal Oxide Semiconductor (CMOS) digital technology is 2.82 mm^2 including pads (without pads 1.42 mm^2). This small area footprint enables μBrain integration in re-trainable sensor ICs to perform various signal processing tasks, such as data preprocessing, dimensionality reduction, feature selection, and application-specific inference. We present an instantiation of the μBrain architecture in a 40 nm CMOS digital chip and demonstrate its efficiency in a radar-based gesture classification with a power consumption of 70 μW and energy consumption of 340 nJ per classification. As a digital architecture, μBrain is fully synthesizable and lends to a fast development-to-deployment cycle in Application-Specific Integrated Circuits (ASIC). To the best of our knowledge, μBrain is the first tiny-scale digital, spike-based, fully parallel, non-Von-Neumann architecture (without schedules, clocks, nor state machines). For these reasons, μBrain is ultra-low-power and offers software-to-hardware fidelity. μBrain enables always-on neuromorphic computing in IoT sensor nodes that require running on battery power for years.

Keywords: spiking neural network, neuromorphic computing, radar signal processing, IoT, edge-AI

OPEN ACCESS

Edited by:
Oliver Rhodes,
The University of Manchester,
United Kingdom

Reviewed by:
Alice Mizrahi,
Thales Group, France
Can Li,
The University of Hong Kong,
Hong Kong

****Correspondence:***
Jan Stuijt
jan.stuijt@imec.nl
Federico Corradi
federico.corradi@imec.nl

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *04 February 2021*
Accepted: *15 April 2021*
Published: *19 May 2021*

Citation:
Stuijt J, Sifalakis M, Yousefzadeh A and Corradi F (2021) μBrain: An Event-Driven and Fully Synthesizable Architecture for Spiking Neural Networks. Front. Neurosci. 15:664208. doi: 10.3389/fnins.2021.664208

1. INTRODUCTION

Information processing in the brain has been a topic of active research for decades (Cappy, 2020). As a computing substrate, the brain structure is exciting from an engineering perspective. It is massively parallel, impressively low power, enables scalable operation, and memory and computation are multiplexed together in the same substrate. As a result of the study of the brain, research in neuromorphic computing has been trying to build brain-inspired models of information processing and respective hardware implementations thereof.

Unlike conventional computer architectures designed to perform exact calculations, a biological brain seems optimized for signal processing in the presence of noisy or incomplete inputs. It is very robust to damages and partial failures. As a result, neuromorphic computing offers an alternative for algorithms and compute architectures that perform (statistical) signal processing

and neural processing tasks. Even though we are far from having understood the brain's functioning altogether, the study of its operation leads us to several important architectural features, which we can successfully and effectively adopt in silicon technology of computing machines.

Many of the brain's energy and compute efficiency features come from its asynchronous and event-driven operation (Yu and Yu, 2017), which promotes and simultaneously exploits sparse computations. In conventional processor/accelerator architectures where high-energy consumption is unavoidable, the focus is on maximizing efficiency (and speed) by increasing the number of operations possible per unit of energy consumed. By contrast, in neuromorphic architectures, sparsity exploitation results in skipping redundant operations, and efficiency is achieved by directly reducing both latency and energy consumption. Reducing operations translates to fewer computations and less power density (i.e., power per silicon area) in the neuromorphic processors. Besides, asynchronous event-driven processing allows for theoretically infinite scalability as every neuron can process its inputs independent of other neurons. It also lets the information flow as fast as possible, which results in a low latency response. It is not required to route a dynamic clock pulse to every neuron in a silicon implementation, as each neuron immediately evaluates its membrane potential against the threshold without the need for a global synchronization signal (a clock).

This paper introduces μBrain, a neuromorphic IC for ultra-low power ($<$100 μW) neural network processing for edge AI IoT applications. μBrain exploits low-cost digital technology, but unlike most other digital neuromorphic Integrated Circuits (ICs) (as shown in **Table 2**), it relies on local on-demand oscillators and a novel delay-cell to avoid the use of a global clock and it supports event-driven processing. μBrain, in the absence of input stimuli, only consumes leakage power while maintaining its internal state stored in the neuron's membrane potential, synaptic weights, and network dynamics. Furthermore, μBrain does not exploit separate memory blocks (either on -chip or off-chip memory), but memory and computation are co-localized in the IC area, avoiding the data access and energy overheads of distal memories of conventional Von-Neumann architectures.

The use of digital technology leverages synthesizability, and it provides reliability for use in various IoT applications. Besides, the high area efficiency of digital gates offered in advanced process nodes makes analog neurons less attractive.

The μBrain architecture is based on digital event-based spiking neurons organized in layers (recurrent topologies are also supported). Inputs and outputs are digital pulses (rate- or time-coded), whereas the synaptic weights are programmable and are stored on-chip with a customizable bit-width. Depending on the application requirements, the μBrain architecture can be customized during synthesis for bit precision, network topology (number of neurons in each layer, and number of layers), and connectivity. In contrast, neuron parameters and synaptic weights are runtime programmable.

The niche of μBrain in the landscape of neuromorphic processors and accelerators is ultra-low-power (e.g., hundreds of μW) lightweight machine-learning data processing near- or in-sensor (and by "in-sensor" we mean integration at the IC level). Example target deployments include radar signal classification, biomedical signal analysis on wearable devices, low-dimensional image classification deployed on luminaires, audio analysis and tactile sensing analysis in thin-film electronics, data processing on ingestible sensors, and many other IoT applications.

1.1. Background and Related Literature

Neuromorphic compute accelerator ICs leverage Spiking Neural Network (SNN) processing, using stateful neuron models that exchange information in the form of sparse asynchronous events (spikes). State-of-the-art implementations are based on analog, digital, or hybrid mixed-signal silicon technology (such as Schemmel et al., 2010; Qiao et al., 2015; Furber, 2016; Neckar et al., 2018), often in combination with "exotic" non-volatile memories (NVM) (Zhang et al., 2018), or photonic technology (Prucnal and Shastri, 2017), or spintronic devices (Grollier et al., 2020). This broad range of options accounts for varying degrees of emulation of the real brain structures, integration, and features.

Analog neuromorphic ICs resemble the biological neural cells more than digital ICs (Indiveri et al., 2011). They model potassium and sodium channels and N-methyl-D-aspartate (NMDA) receptors with their intricate dynamics. Yet, they suffer from variability, high design cost, low flexibility, and low neuron density. When implemented in conventional silicon technology, neurons store their membrane potentials (neuron states) in a leaky capacitor, which costs a large area, and analog synaptic circuits mimic adaptation and learning with programmable synaptic weights with low digital resolution (Bartolozzi and Indiveri, 2007). Alternatively, a dense Resistive Random Access Memory (ReRAM) crossbar may be used to build the synaptic connections between neurons (Liu et al., 2015). In ReRAM crossbars, the bit cell's resistance is the programmable synaptic weight that connects a presynaptic with a post-synaptic neuron. Due to process variations, the analog chips are not exactly reproducible and are vulnerable to temperature changes. In theory, it is possible to overcome the variations by using an adaptive self-learning neuron model and efficient on-chip adaptivity/learning mechanism to compensate for the variations and noise (Kuzum et al., 2012). However, such mechanisms make the neuron more complex. Their performance is not yet sufficiently reliable to enable the use of such technology in critical applications (e.g., health care, automotive, safety). The analog approach is not suitable for our work as μBrain targets inference only, IoT use cases, and easy and affordable reproducibility and integrations with other ICs (e.g., sensors) leveraging in-sensor processing.

By contrast to analog circuits, digital ICs rely on logic gates to emulate neurons and synapses and dense memory to store neuron state and synaptic weights (Frenkel et al., 2018). This approach's motivation is to make a synthesizable architecture integrated quickly in a System On a Chip (SoC) and results in a

low-cost implementation. In theory, due to using logic gates, the required area in this approach can be higher than in analog chips. However, it is easier to use state-of-the-art technology nodes (like 7 nm and below) for digital, which offers much better density at reasonable power consumption. One disadvantage of digitally designed chips is the implementation of membrane potential leakage as an additional periodic operation. This disadvantage is not so relevant if the frequency is low enough, i.e., in the same order as the input spike rates. Besides this, since commercial electronic design automation (EDA) tools are optimized for synchronous deployments, it is not straightforward to implement fully event-driven implementations.

Likewise, in μBrain, we followed a fully digital approach. However, our leakage mechanism is event-based and, therefore, does not necessarily need to be periodic. Additionally, we have designed a lightweight local oscillator (a delay cell) that can drive self-timed digital blocks (similar to Davies et al., 2018) to overcome the lack of support in Electronic Design Automation (EDA) tools.

At the intersection of these two approaches, mixed analog and digital neuromorphic ICs may combine analog circuit networks with a digital readout layer (Corradi et al., 2019) or an analog ReRAM crossbar for synaptic connections with digitally implemented neurons (Ni et al., 2017). In this case, at the interfacing between the analog and digital circuit, analog signals are discretized using an analog to digital converter. As activations in SNNs are binary (no multiplication is required), this method's main advantage is the possibility to store multiple bits in one memory cell. Additionally, bio-inspired learning algorithms can be implemented using resistive memory cells' physical characteristics and can facilitate on-chip learning. Even though μBrain is compatible with non-volatile memory technologies as a replacement of the distributed memory (digital flip-flops) for synaptic weights, we ruled out the analog option for the reasons mentioned before.

As electrons' speed is much faster than ions, a silicon neuron can process spikes some orders of magnitude faster than its real-time biological equivalent (nanoseconds switching on/off time for transistors, vs. milliseconds neuronal and synaptic time constant). This fact has motivated neuromorphic digital IC engineers to implement time-multiplexed digital neuromorphic chips (Davies et al., 2018, Merolla et al., 2011). In digital implementations, it is possible to separate the processing part and the memory. For example, one physical neuron core can emulate many (virtual) neurons and one physical link to emulate many (virtual) synaptic connections. Time-multiplexing methods employ fast computations and constantly shuffle neuron's membrane potential from/to neuron memory and their synaptic weights from/to synaptic memory. Furthermore, such an architecture may host multi-neuron cores, each assigned the emulation of a group of neurons, e.g., a layer, which can exchange spikes asynchronously in a packet-switched form through a network-on-chip (NoC); and based on the Address-Event Representation (AER) of spikes in packets. The advantage of the time-multiplexing approach is a higher neuron and synapse density compared to the previous approaches and leveraging of more complex neuron models [or even programmable (Painkras et al., 2013)] at the cost of increased memory access and complex data-shuffling primitives. Time-multiplexing may be disadvantageous for ultra-low-power designs as it requires additional control circuitry, increasing power consumption to manage the core's coherence. Also, contra to biological neurons, the distance between memory and compute cores increases the power consumption. As events inside each core are processed serially, at peak activity times, processing latency also increases or is not guaranteed and may result in event drop out (depending on the depth and occupancy of event queues). Finally, packetization and explicit addressing of events (as in AER protocols) increase communication overhead (power consumption) due to the additional address processing and routing and memory requirements for queueing events in transit (events are not a binary pulse or a direct signal anymore). In the μBrain architecture, we do not time-multiplex the processing of multiple neurons in a core (rather, each core is assigned exclusively to one neuron) because for the size of networks we are considering, the total silicon area of neurons is negligible compared to the total area of synapse memory. In addition, a packet-based event addressing is not required internally among neurons, but we have opted for AER communication at the chip interface with the outside world for ease of integration with existing neuromorphic sensory systems.

The μBrain area is memory dominated, which is not a good characteristic. However, μBrain requires distributed memories and motivates the search of alternative memory technologies to Static Random Access Memory technologies. Many novel memory technologies are currently being investigated as candidate solutions for neuromorphic technologies, such as Phase Change Memories (PCM) (Nandakumar et al., 2018), Resistance switching memory (RRAM) (Indiveri et al., 2013), Electrochemical Metalization Memories (ECM) (Hao et al., 2021). For this reason, our architecture is not focusing on the memory aspect, as it could soon be replaced with some of the novel technologies.

2. MATERIALS AND METHODS

2.1. Event-Based Architecture

An overview of the main building blocks of the μBrain architecture and their interactions is provided in **Figure 1A**. Event-based integrate-and-fire (IF) neurons are arranged in a fully parallel topology of layered populations, which means that each neuron is physically implemented in silicon (not time-multiplexed). Within each layer, there may exist lateral synaptic connections (that can leverage recurrent connectivity). Every neuron independently (no global clock) accumulates weighted incoming synaptic spikes and emits a spike itself when the neuron's accumulator overflows. Input spikes trigger the membrane voltage integration, with immediate threshold evaluation, resulting in distributed granular activations. As input pulses arrive asynchronously before a neuron layer, an event arbiter resolves any ordering conflicts if spikes arrive simultaneously. Synaptic weights have a fixed bit-width (determined at synthesis) representing 2's complement integer quantized values, in the range $[-2^{W-1}-1, +2^{W-1}-1]$, where

W represents the number of bits. For a given bit-width, the range of quantized weight values can be linearly or logarithmically arranged (the latter case has been taken into account since precision is often more critical for smaller weight values).

Note that while the neuron implements an Integrate-and-Fire (IF) neuron model (see **Figure 1C**), a Leaky Integrate and Fire (LIF) model can also be facilitated by using one of the neuron inputs to provide a periodic leakage signal. This will necessitate an external clocked input (see **Figure 1C**).

2.2. Input/Output Interface

Input and output spikes are transmitted to/from μBrain using a simple communication protocol based on the Address Event Representation (AER). Unlike other common neuromorphic AER systems (Boahen, 2000), which rely on a handshake mechanism, μBrain uses only a strobe signal whose rising edge informs when the address data are ready to be parsed (**Figure 1B**). The strobe is then kept high for a few ns to indicate a time duration that the address data remain valid and a spike is propagated throughout the network.

The AER representation allows seamless interfacing with event-based sensors like the silicon retina (Lichtsteiner et al., 2008) and silicon cochlea (Liu et al., 2010), and microcontrollers to perform further downstream spike-based signal analysis (classification, regression, etc.).

2.3. Spike Arbiter

The spike arbiter before each layer of neurons (shown in **Figure 2A**) detects the presence of at least one input spike and dispatches it to the recipient layer neurons. When more than one spikes arrive simultaneously, the spike arbiter takes care of ordering and spacing them in time[1]. The arbitrations delays are in the order of ns, while the incoming spikes arrive with a spacing in the order of μs, or even *ms* (input frequencies range from Hz to hundreds of kHz).

This functionality is implemented as follows (see **Figure 2A**). Incoming spikes trigger an Input Edge Detector (implemented as shown in **Figure 2C**) and are immediately propagated to a spike register before the Priority-Encoder. A round-robin or linear polling algorithm generates a 1-hot encoded mask, which gets applied to the spike register contents to select a single spike for propagation. Suppose there has been registered more than one simultaneous spike in the spike register. In that case, the difference between the spike register contents and the masked output (i.e., remaining spikes) are fed back to the Input Edge Detector for subsequent recursive processing (until all spikes are consumed one-by-one by the Priority-Encoder). The spikes that come out of the arbiter (see **Figure 1C**) activate (index) parts of the post-synaptic weight memory to select weight values from the fan-out synapses into the respective neurons' accumulators; to incrementally implement a weighted spike integration at each downstream IF neuron.

[1]In this respect, input spike arbitration does not preserve the timing of inter-arrivals.

Upon the arrival of incoming spikes and throughout their consumption, the arbiter circuit becomes on-demand self-clocked by means of a multi-phase single-cycle oscillator and a special delay-cell circuit (explained next).

2.4. The Multi-Phase-Oscillator and Delay Cell

In the absence of a global system-clock, the Multi-Phase-Oscillator (**Figure 2B**) is an on-demand activated local clocking circuit at the heart of the arbiter that warrants correct pacing of its phases for ordered propagation of spikes among neurons and across layers; and in this sense, it is the key component for the event-driven operation of μBrain. The primary sophistication that enables this functionality is a delay-cell (within the multi-phase-oscillator).

Whenever (at least) one spike is latched in the arbiter and propagated to the priority encoder, it sets off one oscillation cycle in the multi-phase-oscillator, which by means of the delay cell gets delivered in sequence at different places of the arbiter to activate, temporarily only, first the loading of the spike register in the priority encoder, then trigger the 1-hot masking/selection of a spike, and finally activate the synaptic memory selector. Its operation is depicted in **Figure 2C**.

The delay cell's generated delays are fixed and take into account the maximum input spike frequency, various integration technology variation parameters, and the overall timing constraints of the circuit during synthesis/place-and-route of the IP. The current prototype operates in a few ns (we used 100ns to have a safe margin). This is a substantially large delay given that in standard CMOS technology timing circuits are generally energy-consuming. It is, however, possible to make considerable delays (hundreds of ns to hundreds of μs) without sacrificing power dissipation using CMOS thyristors (Zhang et al., 2004). Our design uses two thyristors in a cross-coupled configuration (see the schematic of **Figure 3B**), in which the current in the delay cell is limited with a near-threshold bias voltage. The final layout of this cell is compact and, in our design, requires 3.0 μm^2. The delay must be within safe margins while its actual value does not need to be precisely tuned. In the face of these challenges, the delay cell's custom design plays a crucial role in μBrain's low power consumption.

The delay generation is explained as follows: assume that $V_n = 0$ and $Vp = Vdd$ such that both transistors are off (see **Figure 3A**). Then, because of the current source I_c, V_n goes up linearly until $V_n = V_{tn}$ during a time t_{d1} when the NMOS transistor starts to conduct:

$$t_{d1} = \frac{C_n V_{tn}}{I_c} \tag{1}$$

Voltage V_n keeps going up linearly:

$$V_n(t) - V_n = \frac{I_c}{C_n} t \tag{2}$$

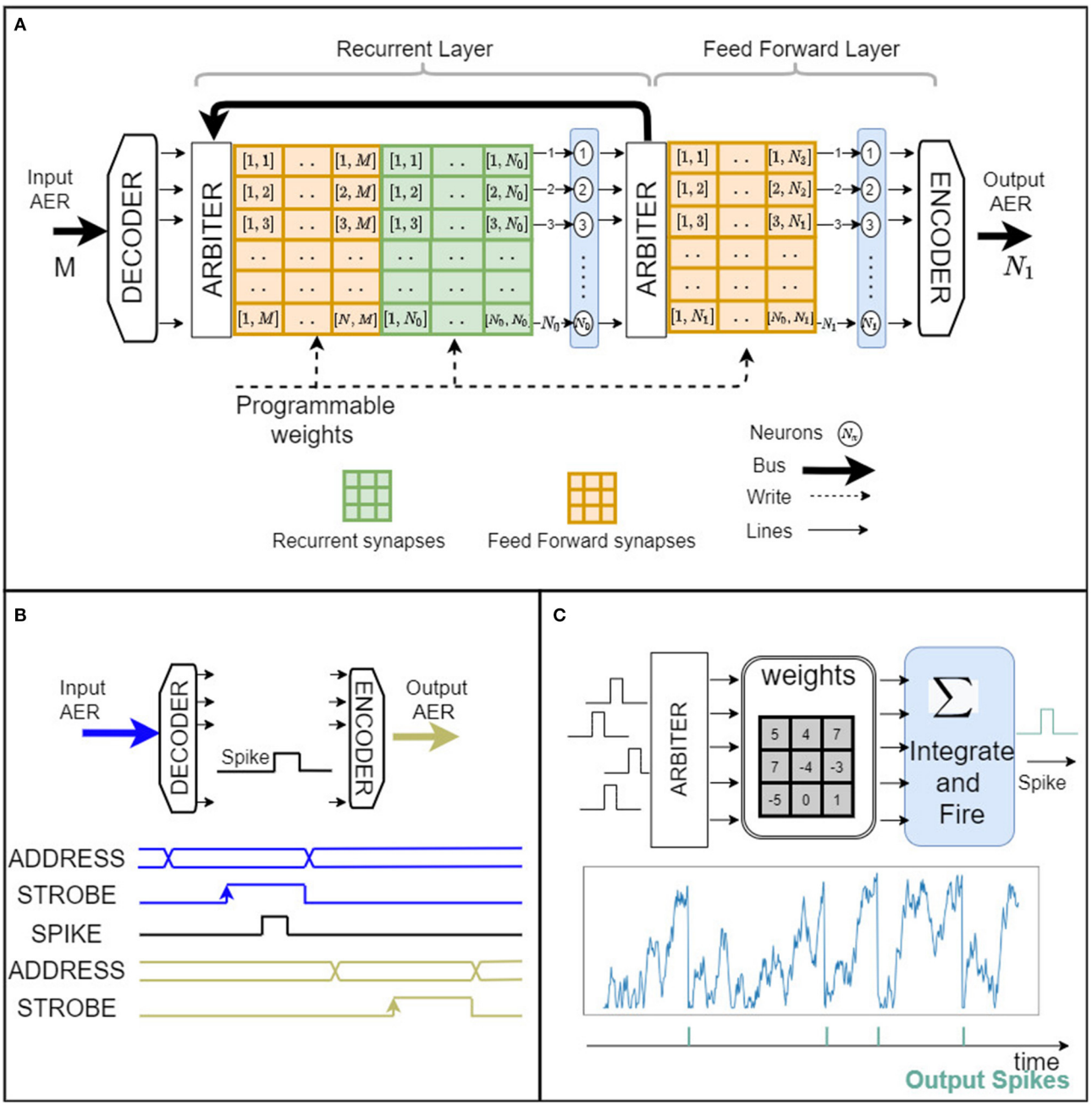


FIGURE 1 | μBrain event-driven architecture. **(A)** The digital architecture is organized in layers. Each layer consists of an arbiter, a weight memory matrix for forward and recurrent connections, and a set of IF neurons. The architecture can be synthesized for an arbitrary number of neurons, weight bit width resolution, and synaptic memory size M, N_x – where M, is the number of inputs and N_x is the number of neurons in layer indexed by x. **(B)** Input/Output address event representation signals and timing. **(C)** Simplified schematic of a digital spiking neuron. Input spikes arriving at random times select corresponding weights, which in turn are added (or subtracted) by an accumulator. Each time the accumulator overflows, the neuron's circuit emits an output spike on the axon output. The graph below shows the time progress of the accumulator value representing the neuron's membrane potential. Output spikes are shown below the neuron's membrane potential.

V_p goes down until $Vdd - V_{tp}$ during a time t_{d2} when the PMOS transistor starts to conduct:

$$I_{dn} = \frac{\beta_n}{2}(V_n - V_{tn})^2 = \frac{\beta_n}{2}\left(\frac{I_c}{C_n}\right)^2 \quad (3)$$

The charge on capacitor C_p is simply the integral in the t_{d2} time interval, as:

$$\int_0^{t_{d2}} I_n dt = C_p V_{tp} \quad (4)$$

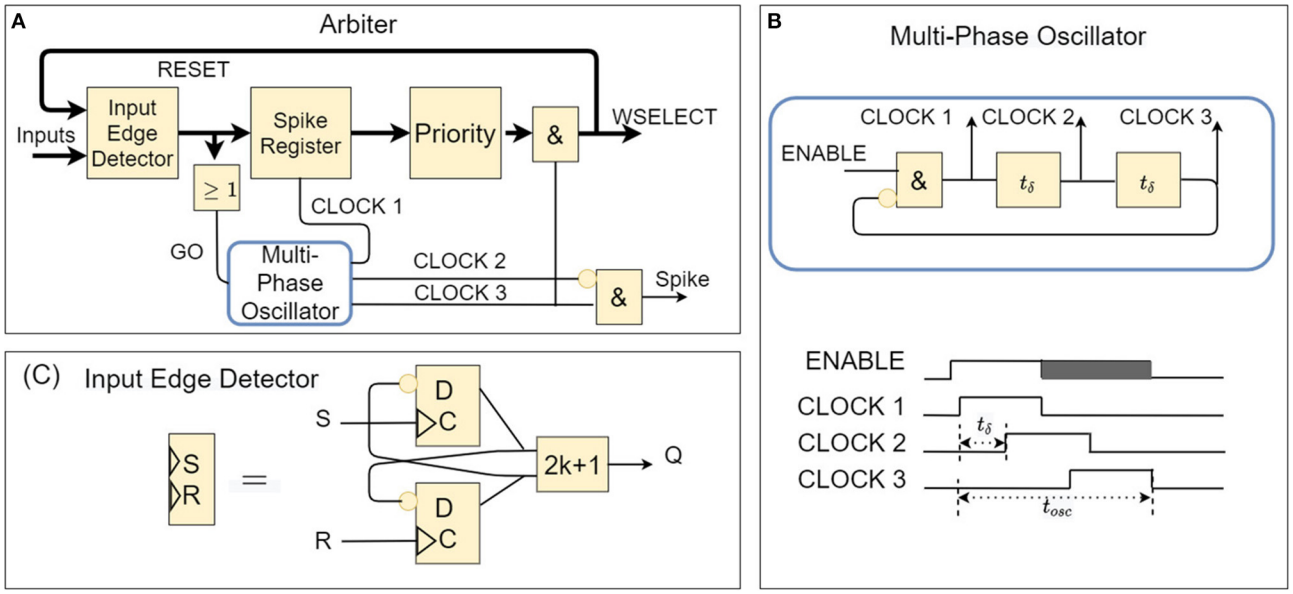


FIGURE 2 | (A) Logic block diagram of spike arbiter (thick lines represent many parallel signals). **(B)** Logic block diagram of the local oscillator and the timing of the self-generated clock pulses. **(C)** Logic block diagram of the input edge detector (edge-triggered) implemented through an S/R circuit. It is parameterizable with a parameter *H* representing the number of states (so it can remember *H-1* spikes). Here we show the case in which (H = 2), i.e., the most straightforward configuration.

Which means that t_{d2} is:

$$t_{d2} = \sqrt[3]{\frac{6C_n^2 C_p}{\beta_n I_c^2}} \tag{5}$$

After, the voltages quickly move to $V_n = V_{dd}$ and $V_p = 0$ Finally the total delay time t_d results in:

$$t_d = t_{d1} + t_{d2} = \left(\frac{V_{tn}}{I_c} + \sqrt[3]{\frac{6V_{tp}}{\beta_n I_c^2}}\right) C_L \tag{6}$$

Where $C_L = C_p = C_n$.

The current in the CMOS delay cell (**Figure 3B**) is limited with a near-threshold bias voltage on node V_N. The delay between node *A* and *X* tracks with process variations, voltage, and temperature (PVT).

3. RESULTS

This section presents an evaluation of an instantiation of μBrain's IP in a 40 nm technology node. For reference comparison of μBrain with other tiny spiking neural network processors, we perform the standard benchmark of handwritten digits recognition (MNIST). We also showcase the capabilities of μBrain while performing a radar-based hand gesture classification task.

3.1. μBrain's ASIC Prototype

We have produced a prototype implementation (see **Figure 4**) consisting of 336 neurons organized in a Recurrent Fully Connect (RFC) layer of 256 neurons, followed by two Fully Connected (FC) layers of 64 and 16 neurons, respectively. The synaptic weights' resolution in all layers has been fixed to 4 bits, representing discrete values from −7 to +7. The weights are runtime re-programmable in local flip-flops, organized via a shift register circuit. The RFC layer has a random connectivity pattern of about 30%, allowing savings in weight memory and using it as a reservoir. After the RFC layer, two FC-connected layers can serve as a second shallow network or can act as a readout classification network. The RFC has 19,878 weight registers (synapses), and the FC has 17,488, which is a total of 37,366. This adds up to 149,464 distributed memory bits (18.2 kB). Both RFC and FC have a global-scale input. When active, the synaptic weights get scaled by a factor of 8 before being accumulated in the neurons. The scaling option sets the threshold to 8 instead of 64. The neuron accumulators' size is 7 bits and can effectively store only positive values from 0 to 63. A neuron will generate an output spike when its accumulator value (i.e., "membrane voltage") overflows. In that case, the accumulator content will not be reset but rather wrapped around. The accumulator's wrapping implies that the neurons reset to the overflow amount after emitting a spike. If a spike causes an underflow, the neuron accumulator is kept to zero. Each FC neuron has a bias input with a corresponding synaptic weight value. The global bias input emulates linear membrane leakage. The reset of the membrane potential at the overflow amount enables to map the behavior of the μBrain neurons to the Rectified Linear Units (ReLU) activations in a mean-rate approximation (to ease ANN to SNN conversion).

μBrain layout area is 2.82 mm^2, we used the 40 nm TSMC technology with I/O voltage of 2.5 V, and a core voltage 1.1 V. A micro-graph picture of the prototype device is shown in **Figure 5**.

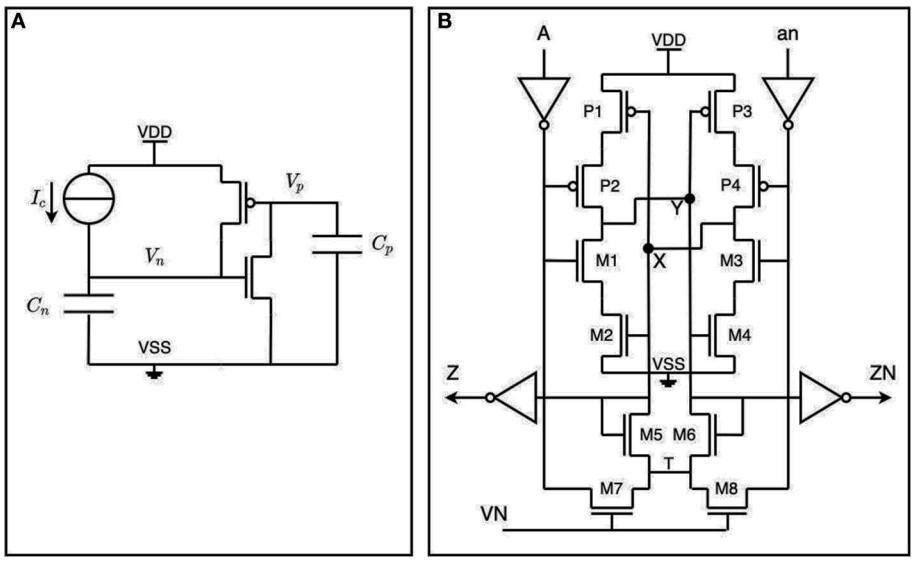


FIGURE 3 | Schematic design of the delay cell. **(A)** A CMOS thyristor is a combination of a PMOS and an NMOS transistor, in which the drain of the PMOS is connected to the gate of the NMOS. **(B)** Two cross-coupled CMOS thyristors implementing a delay-cell.

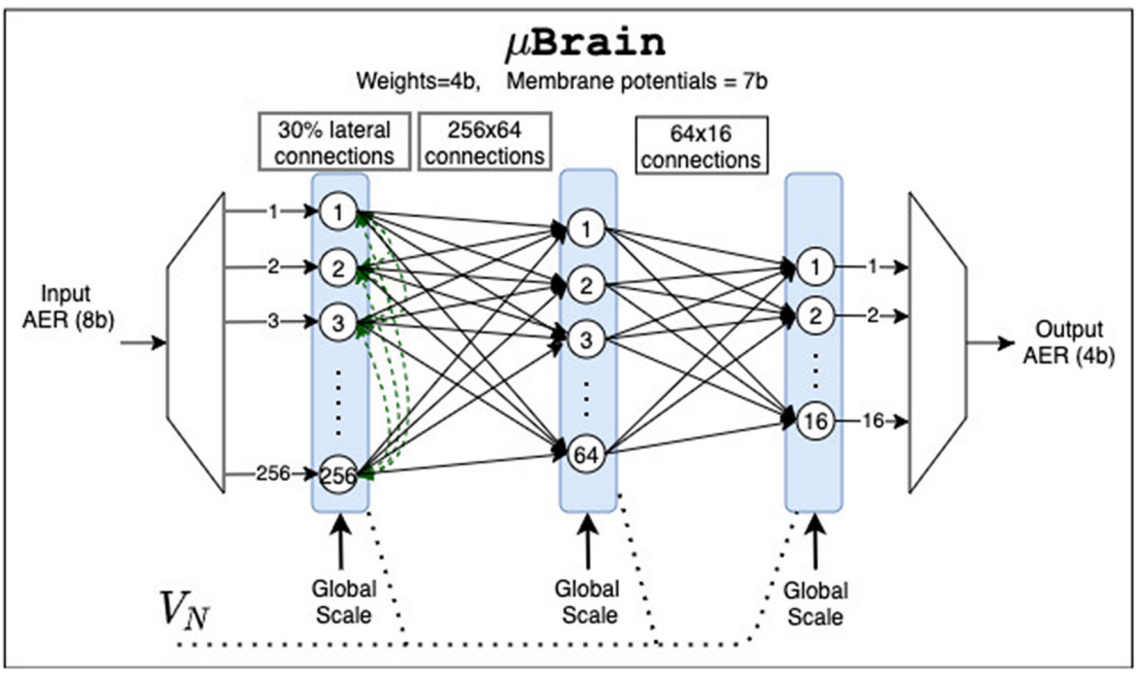


FIGURE 4 | μBrain's ASIC instantiation for the experiments in this paper consists of three layers: a recurrent layer of 256 neurons with circa 30% lateral connectivity and two fully connected layers counting 64 and 16 neurons, respectively. V_N is the global near-threshold bias voltage used to tune the delay cells. The global scale inputs are digital inputs used to set to scale within a layer the synaptic weights.

3.2. Handwritten Digits Classification With μBrain

μBrain is designed for inference only, and training spiking neural networks can be done off-line with various techniques (Rueckauer et al., 2017; Neftci et al., 2019; Sengupta et al., 2019). μBrain is compatible with both spike-time and mean-rate coding schemes. As a proof of concept, we tested the μBrain prototype with a mean rate approach in which we converted a pre-trained Artificial Neural Network (ANN) into a spiking neural network (as first introduced by Pérez-Carrasco et al., 2013). This

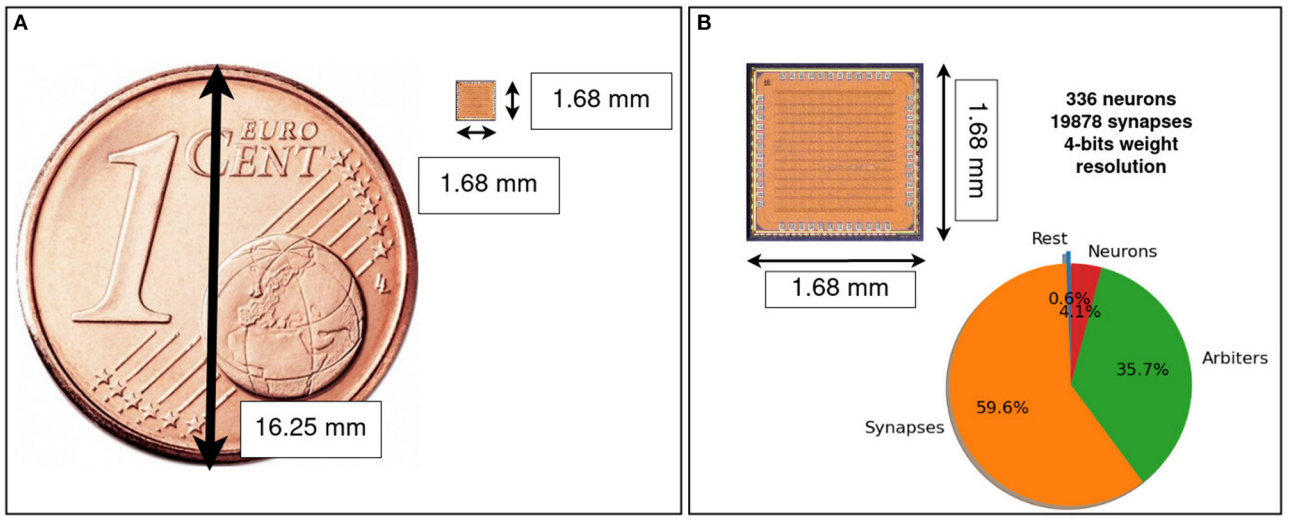


FIGURE 5 | μBrain's micro-graph, the IC is implemented in 40 nm TSMC technology with an I/O voltage of 2.5 V and a core voltage of 1.1 V. **(A)** Micro-graph picture compared with a Euro cent coin. IC area is 2.82 mm^2 (including pads). **(B)** Area breakdown: 59.6% flip-flops for synaptic weights and tri-state weight selectors (synapses), 35.7% spike arbiters, 4.1% neuron accumulators, and 0.6% remaining routing logic. Memory is completely distributed over the area (no Von-Neumann bottleneck).

choice has been dictated by the static nature of the MNIST images and the simplicity of training and testing offered by the standard deep-learning frameworks [e.g., Tensorflow (Shukla and Fricklas, 2018)]. For these reasons, we have also exploited a feed-forward ANN network without relying on recurrent lateral connections. We trained a fully connected network of Rectified Linear Units (ReLU) with 256 inputs, 64 hidden, and 10 output units, respectively, and no biases. Since our instantiation of μBrain has only 256 inputs, we reduced the MNIST input images to 16 × 16 pixels. Pixel grayscale values are mapped into firing rates for the first layer of 256 neurons. The grayscale values $[0, 255]$ are linearly mapped in the arbitrary selected frequency range [100, 655 kHz].

After training, the ANN activation values are encoded in the spiking neurons through their mean rate activations[2]. The weight values transferred from the trained ANN model to the SNN remain the same but are quantized and scaled to fit the limited 4-bit precision in the μBrain instance (i.e., the range $[-1, 1]$ maps to the integer range $[-7, +7]$). The network's output is read out using a single measure of Inter Spike Interval (ISI). The output neuron that has the shortest ISI is considered the correct output class, and the network can proceed to compute the following input.

Figures 6A,B show the impact of weight quantization. The software simulation of the spiking neural network closely matches the hardware measurements. With <4 bit weights, the accuracy decreases significantly. The accuracy in the classification of the 10,000 digits in the MNIST test set (16 × 16 pixels) is consistently 91.7% (92% in the software trained model), with an average energy per prediction of 308 nJ. This performance is consistent with the literature (for the quantization scheme and size of the network used, as reported in **Table 2**).

[2]Note that the actual mean rate frequencies are not significant: it is their frequency ratios that matter.

3.3. Radar-Based Hand Gesture Classification With μBrain

Unlike vision-based imaging sensors, radar imaging systems directly capture motion profiles and temporal variations in the environment through active probing and intercepting the back-scattered power. Here, we applied machine learning to classify these motion patterns as previously proposed in Lien et al. (2016). To use our μBrain prototype in a radar signal classification use case, we converted the traditional micro-Doppler maps into tiny binary images that have been interpreted as spiking inputs for the μBrain device. These binary images indicate which of the 256 input neurons receive spiking inputs, just as in the case of MNIST. Binary images achieve comparable accuracy as grayscale input images, with no statistical difference. This motivates the use of micro-Doppler features as good features for gesture recognition. In contrast to camera-based vision, radar micro-Doppler can provide compressed outputs (sparse FFT coefficients) for faster inference while being robust in low-visibility conditions (e.g., in dark environments).

3.3.1. Event-Based Frequency-Modulated Continuous-Wave (FMCW) Radar Sensor

For proof of concept experimentation, we used a low-power, low-resolution, 8 GHz Ultrawide-Band (UWB) Frequency Modulated Continuous Wave (FMCW) radar from Liu et al. (2019). The low range-resolution (<20 cm) and use of UWB technology in this radar make it a very low-power consumption sensor (20 mW), yet still very effective for various IoT applications, such as vital sign detection (Liu et al., 2019; Mercuri et al., 2019).

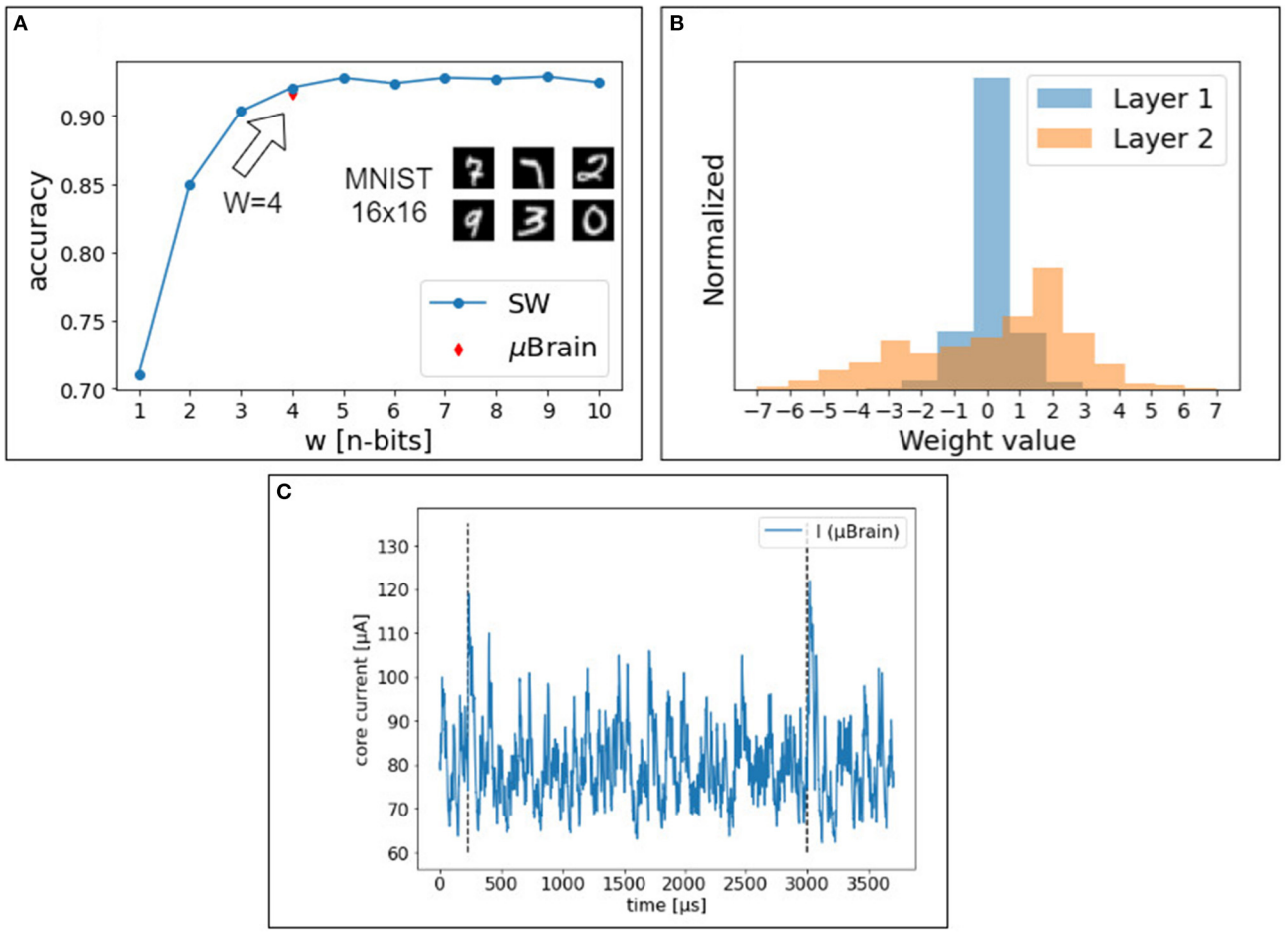


FIGURE 6 | (A) Accuracy achieved in simulation and on the μBrain device, with a two-layer fully connected neural network. The red mark shows the accuracy achieved with the μBrain device. **(B)** Quantized weight distribution for the two layers of the shallow network. **(C)** The blue line shows the current measurement on the chip during handwritten digits classification (MNIST). The network is reset after two consecutive spikes are emitted by any output neurons (peak current reflects the reset). Vertical dashed lines indicate a single-digit classification. The mean current consumption for this digit is $< I \geq 88\mu A$, and it varies among test samples. To classify this digit, it takes 2,769 μs.

FMCW radars transmit a continuous wave with linearly ramping up and/or down frequencies (chirp), starting from a frequency f_0 up to frequency f_n. **Figure 7** shows a measurement of the back-scattered power. Here, we only state that the 8 GHz radar has a range resolution of about 30 cm, making it challenging to detect single finger movements, but enough to detect whole hand gestures' temporal trajectory. The bandwidth of a radar is defined as the frequency interval $B_w = f_n - f_0$. This frequency interval defines the range resolution according to $res = c/2Bw$, in which c is the speed of light.

A photo of the lab prototype platform on which the radar sensor IC is mounted is provided in **Figure 8**. This serves as a test platform for the pre-fabrication of a miniaturized IoT sensor for vital-sign monitoring, activity classification, and other indoor applications. In this prototype, the bulkiest part is an SoC platform, where backend logic (time-and-frequency domain) and communication is implemented and tested on a Field Programmable-Gate Array (FPGA) and embedded Linux processor. A Unix socket interface is used to communicate the spike event data to μBrain. The overarching objective is that the whole FPGA SoC will be obsolete and μBrain will be ultimately packaged in the same IC with the radar sensor. We refer the reader to Liu et al. (2019) for detailed circuits and operational range descriptions.

3.3.2. Radar-Based Hand Gesture Classification in μBrain

With the aforementioned radar setup, we collected a hand-gesture dataset containing four dynamic gestures from five subjects. Data recordings include the subject standing at a distance of 2 m from the antennas (RX and TX). The gestures consist of swinging the right or left arm in the horizontal direction (horizontal), waving with the right or left hand by keeping the palm facing out (hello), moving the hand with the palm facing out radially toward and away from the radar (toward) and finally we recorded background activity in which none of

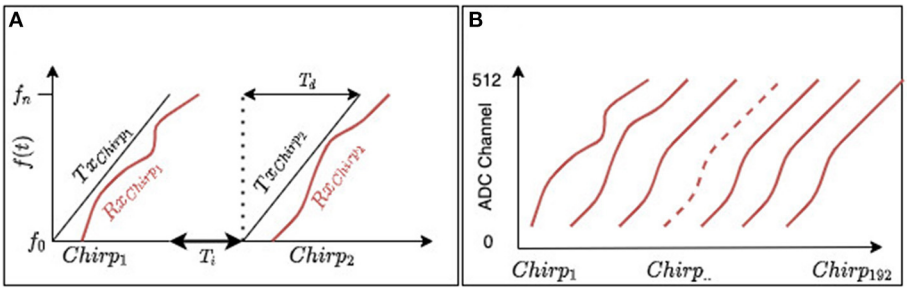


FIGURE 7 | FMCW SISO radar signal illustration. **(A)** A transmitter antenna transmits a signal of linearly increasing frequency starting at f_0 until f_n. A receiving antenna captures back-scattered signal from the environment. T_d represent chirp duration, while T_i is the PRI (time interval) between chirps. **(B)** A radar frame is a collection of 192 consecutive chirp receptions.

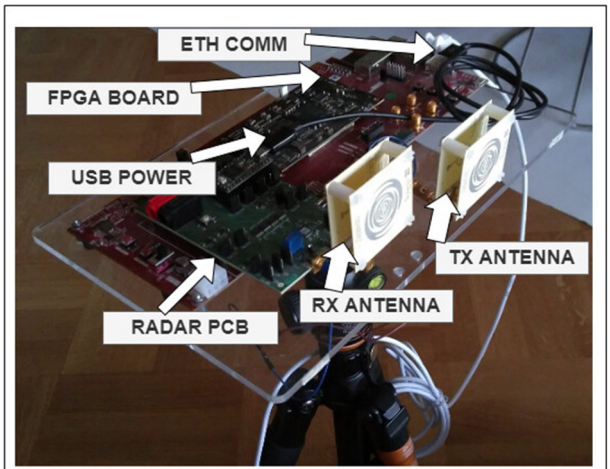


FIGURE 8 | The lab prototype test platform on which the 8 GHz UWB FMCW radar IoT sensor IC is mounted for collecting data and carrying out measurements for vital-sign monitoring, activity classification, and other indoor applications.

the above gestures appeared in a static background (background). The radar system streams out chirp frames (collections of a fixed number of received chirp signal returns; as a 2d-matrix of time-domain data). In our setup, we collect 192 chirps in a single frame, while the number of ADC samples per chirp is 512. The ADC resolution is 10 bits. The time interval between emitted chirps has been set to $T_i = 1.2$ ms while the chirp duration is $T_d = 41\mu s$; therefore, a frame consists of 238 ms of recordings. **Figure 9** (top left) shows three successive frames divided by a vertical dashed line. The second figure from the top left in **Figure 9** shows a micro-Doppler map obtained by processing three frames of radar signal (Chen et al., 2014) (computed as described in **Supplementary Material**). The micro-Doppler maps show the distribution of reflected energy over velocity, at a fixed distance, as a slow-time function. These maps thus provide rich information of the gesture dynamics over time. We converted the micro-Doppler maps into binary images, which serve as spike inputs, to directly interface the radar system with spiking neural networks in μBrain. In this conversion we apply a dynamic threshold on the micro-Doppler map, the threshold on the micro-Doppler map has been set to $Thr = \mu + s \cdot \sigma$, in which μ is the mean of the micro-Doppler map as $\mu = \frac{1}{n}\sum_{i=1}^{n} P_i$, σ is the standard deviation, and s a scaling factor ($s = 0.15$). The scaling factor is a hyper-parameter, serving as a crude noise filter by means of quantizing, and its optimal value is determined through grid search. After thresholding, the pixel values above the threshold value have been set to one while all the others to zero. The image has been scaled to 16×16 pixels as μBrain only supports up to 256 input channels. We show samples from the dataset in the right panel of **Figure 9**.

As per the MNIST use case, we have trained a traditional ANN, and then we have converted it into a spiking neural network. The binary images [0,1] have been mapped with input frequencies equal to 0 Hz and 655 kHz. As previously, we have evaluated the output of the network using a single measure of ISI. The output neuron index with the lowest ISI predicts the input class. Using this dataset, we have achieved an accuracy of 93.4% and energy consumption of 340 nJ per classification. **Table 1** show the confusion matrix for the radar-gesture classification on the test set.

For comparison, in Scherer et al. (2020), the authors developed a very low power embedded processing system for real-time gesture recognition based on radar sensing, which achieves 86.6–92.4% accuracy with energy consumption per classification of 4.52 mJ on inputs from a constellation of high-resolution 60 GHz FMCW radars. One of the two datasets they consider (11-gesture) includes fine gestures with fingers, while the other one (5-gesture) contained more coarse-grained gestures analogous to ours. The radar sensor we used is a much lower resolution (operating at only 8 GHz, with a range resolution in the order of ten of cm instead of sub-cm), and the antenna we used does not provide angular information therefore, the samples are much less informative. The networks they trained were one 2D-CNN (seven layers deep) in tandem with a 1D TCN (10 layers deep) with 16 bit fixed-precision weights, which is to be contrasted with our 2–3 layer SNN of only 4-bit weight precision. Nevertheless, the accuracy we achieve is competitive while our energy consumption per classification is 3-plus orders

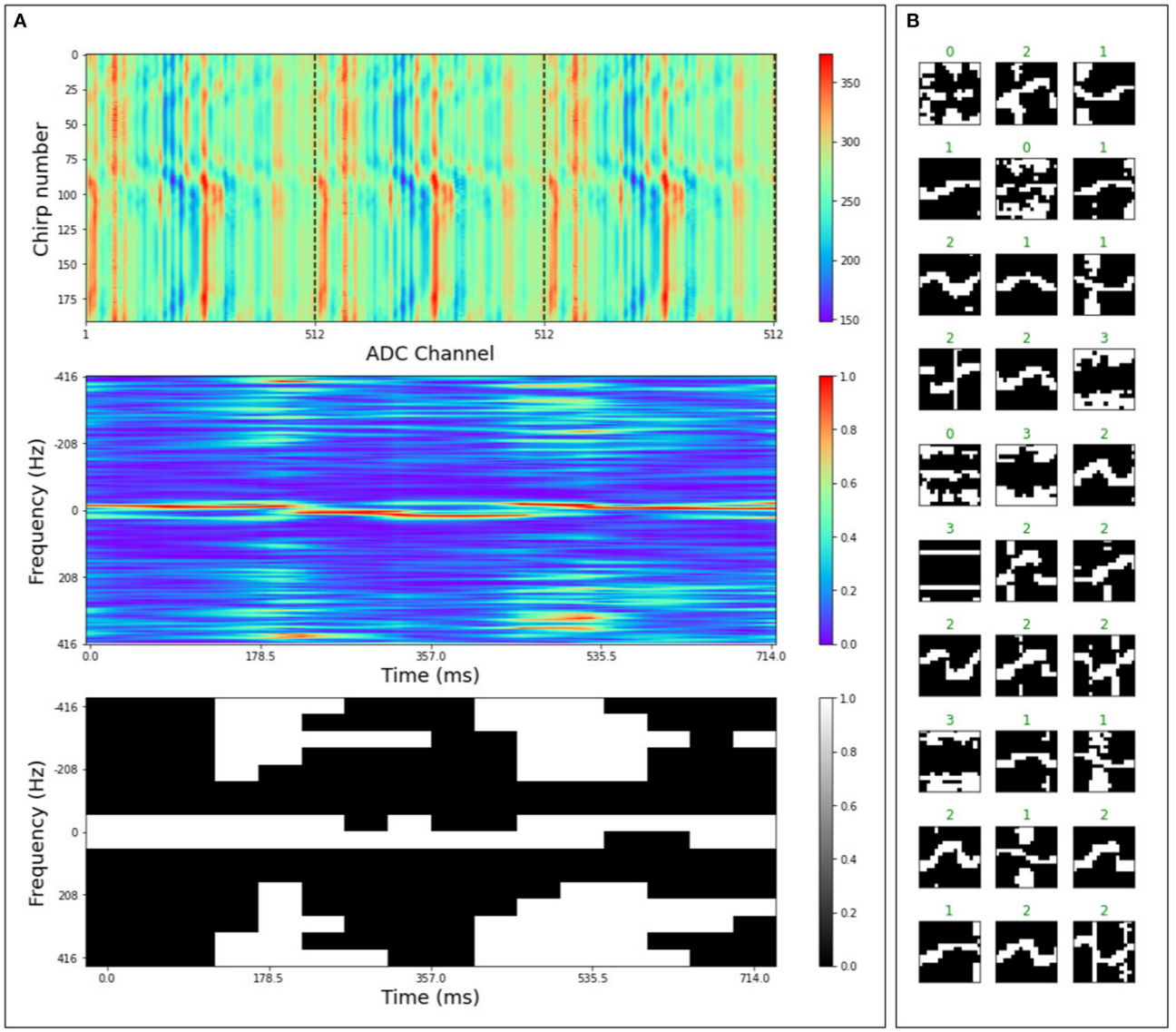


FIGURE 9 | (A) Shows the preprocessing of the radar signal for 3 frames of raw ADC data (**A** top), to a micro-Doppler map (**A** middle), to a thresholded, scaled (16 × 16), and binarized version of the micro-Doppler map (**A** bottom). The binary image gets converted into a spike stream for the μBrain chip. **(B)** Shows examples from the preprocessed radar gesture dataset in which the label at the top associates to its respective gesture as 0: hello, 1: toward, 2: horizontal, 3: background.

of magnitude lower, making our solution truly an ultra-low-power one.

While not directly comparable (but rather as an indicative reference), this performance is on par with results in the literature based on the DvsGesture dataset (Amir et al., 2017) for gesture recognition from a dynamic vision sensor (Delbrück et al., 2010). Using various spiking networks and other machine learning models, the reported accuracy (Amir et al., 2017; Shrestha and Orchard, 2018; Ghosh et al., 2019; Wang et al., 2019; Kaiser et al., 2020; Maro et al., 2020) lies in the range between ∼91 and 96% for 10-gesture classification. In a more closely related to our setup, the authors in Maro et al. (2020) report ∼82 and ∼93% accuracy with and without, respectively dynamic background suppression filtering, using a two layer network and based on a new dynamic vision sensor dataset (NavGesture) that contains five gestures very similar to ours. Last but not least, it is worth pointing that in Amir et al. (2017) from the above list, a 3,951-neuron spiking CNN was deployed in a single True North IC, measuring 44.5 mW power consumption (without the leak) for this task.

4. DISCUSSION

This paper introduced μBrain, a lightweight neuromorphic inference engine for ultra-low power applications in the IoT domain. It offers an alternative to neural network accelerators

TABLE 1 | The confusion matrix for on-chip classification of the radar gesture dataset (test-set).

	Hello	Toward	Horizontal	Background
Hello	70	0	0	5
Toward	0	66	5	4
Horizontal	0	6	120	0
Background	2	0	0	55

when there is a high degree of sparsity (temporal, low-rate) in the input signal that can be exploited to reduce power consumption. Off-the-shelf deep-learning accelerators for edge inference, such as Google EdgeTPU (Cass, 2019), Intel Movidius (Ionica and Gregg, 2015), and Nvidia Jetson (Mittal, 2019) perform a competitive number of operations per watt. However, they cannot efficiently exploit sparsity in the signals to scale their energy use. This means that when the input signal is highly sparse (e.g., natural signals like audio/video/EEG/etc.), they end up performing a large number of redundant operations, which can be skipped. For example, when the sparsity is higher than 95%, <5% of operations are required, and the remaining are just overhead. In deep learning algorithms achieving over 70% activation sparsity while maintaining accuracy within 2% is challenging (Wen et al., 2016; Kurtz et al., 2020). By contrast, in Yin et al. (2020) SNN architectures achieve a very high degree of spatio-temporal sparsity (more than 95%) with negligible accuracy loss.

Compared to many typical ANN accelerators for edge AI, μBrain inherently exploits all types of sparsity (spatial, structural, and temporal) in achieving its ultra-low-power signal processing tasks. Spatial and temporal sparsity relate to neuron activations, while structural sparsity relates to synaptic weights. μBrain takes advantage of spatial sparsity by operating in a truly event-driven fashion: computations take place only for the parts of the input that are non-zero and only when a non-zero activation is propagated through the network, all other lateral parts of the network remain silent conserving energy. It also takes advantage of temporal sparsity since it uses stateful neurons: the memory potential in each neuron is integrating the changes of its inputs, state is thus updated only when there are changes between subsequent inputs and a neuron fires and activates other downstream neurons only when there is sufficient amount of change in the inputs (level crossing). In the absence of any input spikes nothing is active downstream (conserving energy) until there is a change (spike) in space or time. Finally, structural sparsity is programmable in μBrain at synthesis time. Suppose a model has a pruned network topology. In that case, μBrain can be synthesized with reduced synaptic connectivity, which saves area and static power for maintaining weight memory which would otherwise be set to zero as at runtime (an overhead in fully connected crossbar architectures). To give an impression of the related energy costs and savings from reducing spike activity (dynamic power) and synaptic connectivity (static power), in the topology of the MNIST use-case (section 3.2), we measure on average 11,500 spikes per classification (for 6,400 input stimuli per image), where μBrain consumes around 26pJ per spike (including communication, neuron accumulation, and synaptic read) and out of which 30% is static power[3]. Reducing the network connectivity (structural sparsity) or increasing the speed of the network reduces linearly the static power expended due to leakage. Increasing the thresholds in the neuron parameters (spatio-temporal sparsity) also reduces the dynamic power.

One big challenge in digital neuromorphic chips and μBrain's design is static power consumption (leakage power). While the architecture is designed to have event-driven dynamic power consumption (consume dynamic power only when there is an event), there is no control on static power. Since the architecture area is dominated by memory, most of the static power is consumed to keep the flip-flop-based memories alive. However, this challenge can be tackled at various levels, such as using Fully-Depleted Silicon-On-Insulator (FDSOI) (Carter et al., 2016) manufacturing technology, advanced non-volatile memory technologies (Burr et al., 2017), digital design tricks (e.g., power gating when no inputs are present), and by pruning at synthesis time unneeded synaptic connectivity (as discussed above).

μBrain has been designed to offer flexibility and customizability for different applications in the IoT domain. This means that it is possible to change the number of neurons in each layer, the number of layers, connectivity structure, and the parameters' resolution. The design incentive is to empower in this way IoT applications where power consumption is the number one priority and make integration with various sensors effortless (more often than not by packaging μBrain and the sensor in the same IC); to perform tiny machine learning tasks that were not possible or affordable (energy-wise) before. It is less efficient for implementing very deep neural networks as silicon area efficiency plays an essential role. The lack of time-multiplexed neuron cores in μBrain limits the scalability. However, avoiding time-multiplexing of neuron processing has been a conscious trade-off given the target application domain (i.e., small networks, energy efficiency), since it has enabled the co-location of memory and processing.

Another aspect that, at first sight, might appear as a limitation of μBrain is the use of Integrate-and-Fire (IF) neurons. However, there is recurrent synaptic connectivity among neurons the absence of leakage in the neurons may see as unnecessarily restrictive to the effectiveness of recurrent network architectures. In practice, however, quite the opposite holds. It is easy to introduce leakage at a fine-grained neuron level (different leak functions and with varying parameters per neuron); by sacrificing for this purpose, one neuron's inputs. This choice has been motivated by the intended use of μBrain primarily for experimental purposes.

Finally, one current inconvenience in the μBrain architecture is that the delay cell, which is one of the critical components, requires re-customization when ported to different manufacturing technologies. Moreover, while there is an advantage in going to small node technologies in terms of power consumption and area, the delay cell's speed will remain the

[3]These numbers are for Vdd 1.1 V in 40 nm technology, with 53 μA leak current and 74 μA total current in 42 s of classifying 10,000 samples.

TABLE 2 | Reference comparison of μBrain with other neuromorphic processors for the MNIST handwrittend digit classification.

	μBrain	**Frenkel et al. (2018)**	**Park et al. (2019)**	**Cho et al. (2019)**	**Chen et al. (2018)**	**Moradi et al. (2017)**	**Davies et al. (2018)**
MNIST accuracy (%)	91.7 (16 × 16)	91.4 (16 × 16)	97.83	91.6 (16 × 16)	97.9	–	96.4
Neuron/Synapses used for MNIST	74/17k	10/2.5k	410/199k	2048/149k	1546/666k	–	10/7840
VDD (V)	1.1	0.55–1.1	0.8	0.7	0.525–0.9	1.3–1.8	0.5–1.25
Energy/Prediction (nJ)	308	15 @ 75 MHz, 54 @ 1.3 MHz	236.5	–	1700	–	85,52*
Technology (nm)	40	28 FDSOI	65	40	10 FinFET	180	14 FinFET
Physical neurons cores/total neurons	336/336	1/256	410/410	2048/2048	4096/4096	1024/1024	128/131072
Power	73 μW	35–447 μW	23.6 mW	46.6 mW (2.3 uW * 4096 neurons)	94 mW	400 μW @ 10 Hz average firing rate	110 mW
Area (mm^2)	2.68 (1.42 core only)	0.086**	10.08	2.56	1.7	43.79	60
Synaptic resolution # bits	4	4	>10	2/3	7	2 (analog)	1–9
Clock frequency	Event-driven	75 MHz	20 MHz	Global Async. Locally sync 110 MHz (neurons)	105 MHz	Event-driven	Event-driven
Fully synthesizable	Yes	Yes	Yes	Yes	Yes	No (Analog Mixed Signal design)	Yes
Supported algorithm	SNN feed-forward, recurrent	SNN online learning, feed-forward	SNN on-line learning	SNN feed-forward, recurrent	SNN/BNN online-learning, feed forward, recurrent	SNN feed-forward, recurrent	SNN, online-learning, feed-forward, recurrent

*The μBrain's power is measured with the input frequencies of [100, 655 kHz], this result in an average time per classification of 4.2ms. *Blouw et al. (2019). **Only IP core area without peripheral and pads.*

same in practice. While this is a minor nuisance, it is slightly at odds with the otherwise general design portability provided by the synthesizability in a complete digital design.

4.1. μBrain and Low-Power Neuromorphic Devices

Several other ultra-low-power neuromorphic processors have recently been developed. **Table 2** compares our proposed architecture with the other state-of-the-art neuromorphic architectures for which the power consumption reported is <120 mW. Among them, μBrain achieves competitive energy consumption per prediction (308 nJ/MNIST classification) without compromising accuracy. It is an entirely event-driven design (i.e., consumes only leakage power in the absence of input) and is fully synthesizable.

μBrain should be categorized as a small-scale neuromorphic processor. Unlike large-scale processors (like Davies et al., 2018), where the power consumption is several mW, small-scale processing units like μBrain only consume a few μW and therefore can be integrated with battery-powered always-on devices (for example, in wearable or implantable devices). Additionally, these processors can be integrated with the sensors to build a highly efficient sensor-processor system-on-chip (SoC).

Frenkel et al. (2018) designed and implemented a 256-neuron processor with online learning capability and time-multiplexing of an entire topology in a single physical neuron core. The neurons in this design are fully connected (256 × 256 synapse), which allows for arbitrary topologies. However, this high amount of synaptic connections is an overhead not required for many applications. In μBrain, our approach is to sacrify runtime flexibility for efficiency. Therefore, we decided to perform mapping-synthesis co-optimization. After synthesis and fabrication of the chip, in μBrain, it is only possible to modify the synaptic weights of the SNN but not the main configuration (synaptic connectivity). This saves substantial area and allows for highly efficient implementation of the processing unit for a target application (for example, when integrating with a radar sensor).

Also, by contrast to Frenkel et al. (2018) as well as Davies et al. (2018), μBrain does not time-multiplex neurons in neuron cores, which leverages the co-localization of memory and compute (to improve latency and energy consumption).

Park et al. (2019) also presented a clocked SNN architecture processor, but the proposed processor consumes over 20 mW and cannot be used for always-on, battery-powered applications. In contrast to this work and Frenkel et al. (2018), μBrain does not use a fixed clock frequency, making it more efficient for event-based applications. Compared to other event-driven ASICs like Davies et al. (2018), the shallow processing pipeline of μBrain allows for a lightweight oscillator to generate just a few pulses upon each event's arrival.

Moradi et al. (2017) presented an analog neuromorphic processor. Even though the analog design has clear advantages

over the digital one, it is not easily integratable and synthesizable with other digital units (e.g., sensors) and therefore different from our proposed solution. As we discussed before, analog design is also vulnerable to manufacturing variations, making its simulation and training in software difficult. It is challenging to use for critical applications like healthcare. Nevertheless, μBrain gets as close as possible to an analog design by featuring a clock-less architecture (truly event-driven) and co-localizing computation and memory in the same die.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found at: https://github.com/federicohyo/8GhzGestureDataset.

AUTHOR CONTRIBUTIONS

JS and FC designed the μBrain architecture and performed the experiment. JS implemented the μBrain's architecture in digital logic. FC and MS collected the dataset and performed the pre-processing. FC designed the experiment. All authors contributed with discussions and assisted in editing the manuscript.

FUNDING

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No. 826610. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Austria, Belgium, Czech Republic, France, Italy, Latvia, and Netherlands. This work has also been partially supported by the EUREKA cluster PENTA and funded by Dutch authorities under grant agreement PENTA2018e-17004-SunRISE.

ACKNOWLEDGMENTS

The results presented in this work were also obtained thanks to the collaboration of many colleagues, whom we like here to acknowledge Yao-Hong Liu, Ali Safa, André Bourdoux, Ilja Ocket, and Francky Catthoor.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.664208/full#supplementary-material

REFERENCES

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Honolulu, HI), 7388–7397. doi: 10.1109/CVPR.2017.781

Bartolozzi, C., and Indiveri, G. (2007). Synaptic dynamics in analog VLSI. *Neural Comput.* 19, 2581–2603. doi: 10.1162/neco.2007.19.10.2581

Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2019). "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop* (Albany, NY), 1–8. doi: 10.1145/3320288.3320304

Boahen, K. A. (2000). Point-to-point connectivity between neuromorphic chips using address events. *IEEE Trans. Circuits Syst. II Analog Digital Signal Process.* 47, 416–434. doi: 10.1109/82.842110

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys. X* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Cappy, A. (2020). *Neuro-inspired Information Processing*. John Wiley & Sons. doi: 10.1002/9781119721802

Carter, R., Mazurier, J., Pirro, L., Sachse, J., Baars, P., Faul, J., et al. (2016). "22 nm FDSOI technology for emerging mobile, internet-of-things, and RF applications," in *2016 IEEE International Electron Devices Meeting (IEDM)* (IEEE), 2. doi: 10.1109/IEDM.2016.7838029

Cass, S. (2019). Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectrum* 56, 16–17. doi: 10.1109/MSPEC.2019.8701189

Chen, G. K., Kumar, R., Sumbul, H. E., Knag, P. C., and Krishnamurthy, R. K. (2018). A 4096-neuron 1m-synapse 3.8-pJ/SOP spiking neural network with on-chip stdp learning and sparse weights in 10-nm FinFET CMOS. *IEEE J. Solid State Circuits* 54, 992–1002. doi: 10.1109/JSSC.2018.2884901

Chen, V. C., Tahmoush, D., and Miceli, W. J. (2014). *Radar Micro-Doppler Signatures*. Institution of Engineering and Technology. doi: 10.1049/PBRA034E

Cho, S. G., Beigné, E., and Zhang, Z. (2019). "A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40 nm CMOS," in *2019 IEEE Custom Integrated Circuits Conference (CICC)* (Austin, TX: IEEE), 1–4. doi: 10.1109/CICC.2019.8780116

Corradi, F., Pande, S., Stuijt, J., Qiao, N., Schaafsma, S., Indiveri, G., et al. (2019). "ECG-based heartbeat classification in neuromorphic hardware," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest: IEEE), 1–8. doi: 10.1109/IJCNN.2019.8852279

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Delbrück, T., Linares-Barranco, B., Culurciello, E., and Posch, C. (2010). "Activity-driven, event-based vision sensors," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris: IEEE), 2426–2429. doi: 10.1109/ISCAS.2010.5537149

Frenkel, C., Lefebvre, M., Legat, J. D., and Bol, D. (2018). A 0.086-mm 212.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS. *IEEE Trans. Biomed. Circuits Syst.* 13, 145–158. doi: 10.1109/TBCAS.2018.2880425

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001

Ghosh, R., Gupta, A., Nakagawa, A., Soares, A., and Thakor, N. (2019). *Spatiotemporal Filtering for Event-Based Action Recognition*. IEEE Transactions in Pattern Analysis and Machine Intelligence.

Grollier, J., Querlioz, D., Camsari, K., Everschor-Sitte, K., Fukami, S., and Stiles, M. D. (2020). Neuromorphic spintronics. *Nat. Electron.* 3, 360–370. doi: 10.1038/s41928-019-0360-9

Hao, Y., Wu, H., Yang, Y., Liu, Q., Gong, X., Han, G., et al. (2021). Preface to the special issue on beyond moore: Resistive switching devices for emerging memory and neuromorphic computing. *J. Semisond.* 42:010101. doi: 10.1088/1674-4926/42/1/010101

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., Linares-Barranco, B., Legenstein, R., Deligeorgis, G., and Prodromakis, T. (2013). Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology* 24:384010. doi: 10.1088/0957-4484/24/38/384010

Ionica, M. H., and Gregg, D. (2015). The movidius myriad architecture's potential for scientific computing. *IEEE Micro* 35, 6–14. doi: 10.1109/MM.2015.4

Kaiser, J., Mostafa, H., and Neftci, E. (2020). Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Front. Neurosci*. 14:424. doi: 10.3389/fnins.2020.00424

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., et al. (2020). "Inducing and exploiting activation sparsity for fast inference on deep neural networks," in *International Conference on Machine Learning* (Vienna: PMLR), 5533–5543.

Kuzum, D., Jeyasingh, R. G. D., Yu, S., and Wong, H. S. P. (2012). Low-energy robust neuromorphic computation using synaptic devices. *IEEE Trans. Electron Devices* 59, 3489–3494. doi: 10.1109/TED.2012.2217146

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128 × 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Lien, J., Gillian, N., Karagozler, M. E., Amihood, P., Schwesig, C., Olson, E., et al. (2016). Soli: Ubiquitous gesture sensing with millimeter wave radar. *ACM Trans. Graphics* 35, 1–19. doi: 10.1145/2897824.2925953

Liu, C., Yan, B., Yang, C., Song, L., Li, Z., Liu, B., et al. (2015). "A spiking neuromorphic design with resistive crossbar," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (San Francisco, CA: IEEE), 1–6. doi: 10.1145/2744769.2744783

Liu, S. C., Van Schaik, A., Minch, B. A., and Delbruck, T. (2010). "Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris: IEEE), 2027–2030. doi: 10.1109/ISCAS.2010.5537164

Liu, Y. H., Sheelavant, S., Mercuri, M., Mateman, P., Dijkhuis, J., Zomagboguelou, W., et al. (2019). "A 680 μw burst-chirp UWB radar transceiver for vital signs and occupancy sensing up to 15 m distance," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Francisco, CA: IEEE), 166–168. doi: 10.1109/ISSCC.2019.8662536

Maro, J. M., Ieng, S. H., and Benosman, R. (2020). Event-based gesture recognition with dynamic background suppression using smartphone computational capabilities. *Front. Neurosci*. 14:275. doi: 10.3389/fnins.2020.00275

Mercuri, M., Lorato, I. R., Liu, Y. H., Wieringa, F., Van Hoof, C., and Torfs, T. (2019). Vital-sign monitoring and spatial tracking of multiple people using a contactless radar-based sensor. *Nat. Electron*. 2, 252–262. doi: 10.1038/s41928-019-0258-6

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., and Modha, D. S. (2011). "A digital neurosynaptic core using embedded crossbar memory with 45 pJ per spike in 45 nm," in *2011 IEEE Custom Integrated Circuits Conference (CICC)* (San Jose, CA: IEEE), 1–4. doi: 10.1109/CICC.2011.6055294

Mittal, S. (2019). A survey on optimized implementation of deep learning models on the Nvidia Jetson platform. *J. Syst. Archit*. 97, 428–442. doi: 10.1016/j.sysarc.2019.01.011

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2017). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Trans. Biomed. Circuits Syst*. 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

Nandakumar, S., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., and Eleftheriou, E. (2018). A phase-change memory model for neuromorphic computing. *J. Appl. Phys*. 124:152135. doi: 10.1063/1.5042408

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., et al. (2018). Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proc. IEEE* 107, 144–164. doi: 10.1109/JPROC.2018.2881432

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag*. 36, 51–63. doi: 10.1109/MSP.2019.2931595

Ni, L., Liu, Z., Yu, H., and Joshi, R. V. (2017). An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism. *IEEE J. Explor. Solid State Comput. Devices Circuits* 3, 37–46. doi: 10.1109/JXCDC.2017.2697910

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., et al. (2013). Spinnaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation. *IEEE J. Solid State Circuits* 48, 1943–1953. doi: 10.1109/JSSC.2013.2259038

Park, J., Lee, J., and Jeon, D. (2019). "7.6 A 65 nm 236.5 nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Francisco, CA: IEEE), 140–142. doi: 10.1109/ISSCC.2019.8662398

Pérez-Carrasco, J. A., Zhao, B., Serrano, C., Acha, B., Serrano-Gotarredona, T., Chen, S., et al. (2013). Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing-application to feedforward convnets. *IEEE Trans. Pattern Anal. Mach. Intell*. 35, 2706–2719. doi: 10.1109/TPAMI.2013.71

Prucnal, P. R., and Shastri, B. J. (2017). *Neuromorphic Photonics*. CRC Press. doi: 10.1201/9781315370590

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Front. Neurosci*. 9:141. doi: 10.3389/fnins.2015.00141

Rueckauer, B., Lungu, I. A., Hu, Y., Pfeiffer, M., and Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci*. 11:682. doi: 10.3389/fnins.2017.00682

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Scherer, M., Magno, M., Erb, J., Mayer, P., Eggimann, M., and Benini, L. (2020). TinyRadarNN: combining spatial and temporal convolutional neural networks for embedded gesture recognition with short range radars. *arXiv* 2006.16281. doi: 10.1109/JIOT.2021.3067382

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci*. 13:95. doi: 10.3389/fnins.2019.00095

Shrestha, S. B., and Orchard, G. (2018). Slayer: spike layer error reassignment in time. *arXiv* 1810.08646.

Shukla, N., and Fricklas, K. (2018). *Machine Learning With TensorFlow*. Manning Greenwich.

Wang, Z., Hou, Y., Jiang, K., Dou, W., Zhang, C., Huang, Z., et al. (2019). Hand gesture recognition based on active ultrasonic sensing of smartphone: a survey. *IEEE Access* 7, 111897–111922. doi: 10.1109/ACCESS.2019.2933987

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). "Learning structured sparsity in deep neural networks," in *In Proceedings of 2016 conference in Advancesin Neural Information Processing Systems (NIPS)* (Barcelona), Vol. 29, 2074–2082.

Yin, B., Corradi, F., and Bohté, S. M. (2020). Effective and efficient computation with multiple-timescale spiking recurrent neural networks. *arXiv* 2005.11633. doi: 10.1145/3407197.3407225

Yu, L., and Yu, Y. (2017). Energy-efficient neural information processing in individual neurons and neuronal networks. *J. Neurosci. Res*. 95, 2253–2266. doi: 10.1002/jnr.24131

Zhang, J., Cooper, S. R., LaPietra, A. R., Mattern, M. W., Guidash, R. M., and Friedman, E. G. (2004). "A low power thyristor-based CMOS programmable delay element," in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04ch37512)*, Vol. 1 (Vancouver, BC: IEEE), I-769. doi: 10.1109/ISCAS.2004.1328308

Zhang, X., Huang, A., Hu, Q., Xiao, Z., and Chu, P. K. (2018). Neuromorphic computing with memristor crossbar. *Phys. Status Solidi A* 215:1700875. doi: 10.1002/pssa.201700875

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Artificial Intelligence

ORIGINAL RESEARCH
published: 09 June 2021
doi: 10.3389/frai.2021.659060

NeuroSim Simulator for Compute-in-Memory Hardware Accelerator: Validation and Benchmark

*Anni Lu, Xiaochen Peng, Wantong Li, Hongwu Jiang and Shimeng Yu**

School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, United States

Compute-in-memory (CIM) is an attractive solution to process the extensive workloads of multiply-and-accumulate (MAC) operations in deep neural network (DNN) hardware accelerators. A simulator with options of various mainstream and emerging memory technologies, architectures, and networks can be a great convenience for fast early-stage design space exploration of CIM hardware accelerators. DNN+NeuroSim is an integrated benchmark framework supporting flexible and hierarchical CIM array design options from a device level, to a circuit level and up to an algorithm level. In this study, we validate and calibrate the prediction of NeuroSim against a 40-nm RRAM-based CIM macro post-layout simulations. First, the parameters of a memory device and CMOS transistor are extracted from the foundry's process design kit (PDK) and employed in the NeuroSim settings; the peripheral modules and operating dataflow are also configured to be the same as the actual chip implementation. Next, the area, critical path, and energy consumption values from the SPICE simulations at the module level are compared with those from NeuroSim. Some adjustment factors are introduced to account for transistor sizing and wiring area in the layout, gate switching activity, post-layout performance drop, etc. We show that the prediction from NeuroSim is precise with chip-level error under 1% after the calibration. Finally, the system-level performance benchmark is conducted with various device technologies and compared with the results before the validation. The general conclusions stay the same after the validation, but the performance degrades slightly due to the post-layout calibration.

Keywords: compute-in-memory, hardware accelerator, deep neural network, design automation, benchmarking and validation

OPEN ACCESS

Edited by:
Irem Boybat,
IBM Research - Zurich, Switzerland

Reviewed by:
Manan Suri,
Indian Institute of Technology Delhi, India
Matthew Marinella,
Sandia National Laboratories (SNL), United States

****Correspondence:***
Shimeng Yu
shimeng.yu@ece.gatech.edu

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: *26 January 2021*
Accepted: *14 May 2021*
Published: *09 June 2021*

Citation:
Lu A, Peng X, Li W, Jiang H and Yu S (2021) NeuroSim Simulator for Compute-in-Memory Hardware Accelerator: Validation and Benchmark. Front. Artif. Intell. 4:659060. doi: 10.3389/frai.2021.659060

INTRODUCTION

State-of-the-art deep neural network (DNN)–based machine learning algorithms have demonstrated remarkable effectiveness for various artificial intelligence applications such as image processing, speech recognition, and language translation (Deng et al., 2020). However, due to the requirement of high parallelism and power consumption for data movement, computing platforms with traditional von Neumann architecture are inadequate for efficient processing of DNNs. Compute-in-memory (CIM) is a promising solution to alleviate the memory access bottleneck and has achieved attractive energy efficiency when implemented with mature SRAM technology at 7 nm (Dong et al., 2020). With recent progress in emerging nonvolatile memory (eNVM) devices such as resistive random

access memory (RRAM) (Xue et al., 2020), phase change memory (PCM) (Burr et al., 2015), and ferroelectric field-effect transistor (FeFET) (Dutta et al., 2020), the application of a CIM-based DNN accelerator is even more intriguing since eNVMs offer low leakage power and nonvolatility which are necessary for dynamic power gating and instant on and off operations in smart edge devices.

However, the performance of CIM can be highly dependent on design factors such as sub-array size, analog-to-digital converter (ADC) precision, and device conductance. Though accurate, the circuit-level SPICE simulation requires dramatically increasing time with the scale of the DNN model. Therefore, a design automation simulator that supports fast modeling of CIM accelerators with various memory technologies and flexible architecture topologies is required to realize an early-stage design space exploration. Among all the reported CIM simulators, NeuroSim (Chen et al., 2018) stands out as a comprehensive platform as it covers a wide variety of design options from a device level to a circuit level and up to an algorithm level. The inputs to the simulator include memory types, nonideal device parameters, transistor technology nodes, network topology and sub-array size, and training dataset and traces. The outputs of the simulator include the hardware performance metrics, such as area, latency, dynamic energy and leakage power consumption, and algorithm-level training/inference accuracy in the run-time. NeuroSim is interfaced with PyTorch, forming an end-to-end benchmark framework, namely, DNN+NeuroSim (Peng et al., 2019), which is publicly available at GitHub with hundreds of users including industry researchers from Intel, Samsung, TSMC, and SK Hynix.

To our best knowledge, *none of other CIM simulators have been validated with the actual silicon data*, although the peripheral circuit modules (e.g., decoder, switch matrix, mux, and adder) of NeuroSim have been validated with SPICE simulations using the PTM model (PTM, 2011) and FreePDK (FreePDK, 2014). It is known that the PTM model and FreePDK are for educational purposes, rather than for foundry fabrication purposes. Therefore, it is imperative to validate the simulator's prediction with the silicon implementation. In this study, we will validate NeuroSim against a 40-nm 16-kb CIM macro using the TSMC 40-nm RRAM process (Chou et al., 2018), which has been taped out recently (Li et al., 2021). First, the parameters of the memory device and CMOS transistor are exacted from the TSMC's PDK and employed in the NeuroSim settings. Next, the comparison is made on the analog and digital modules, respectively. New modules such as a level shifter, which uses I/O transistors (to support RRAM's high write voltage), is added to NeuroSim libraries. The area, critical path delay, and energy consumption are evaluated between the analytical modeling and the SPICE simulations from Cadence Spectre. Finally, adjustment factors are introduced to tune the transistor size, add the wiring area in layout, consider the gate switching rate and the post-layout performance drop, etc. Using the validated NeuroSim settings, we will further benchmark CIM accelerators with a variety of device technologies and compare the performance prediction before and after the validation. It is noted that we only focus on the hardware performance validation in this work and do not focus on the software accuracy estimation, though the inference accuracy is reportable from the framework.

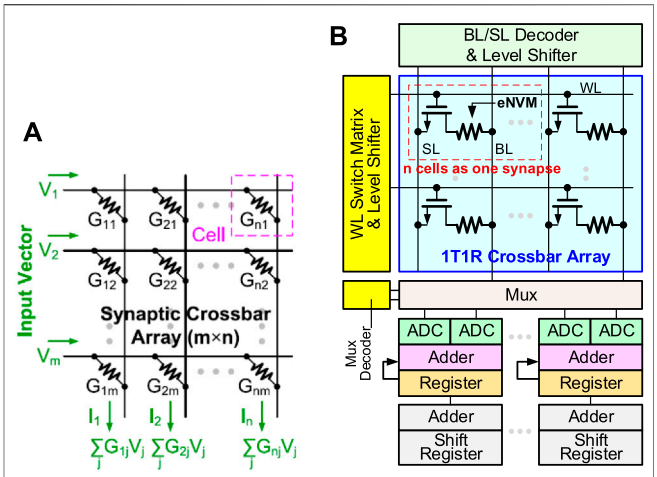


FIGURE 1 | (A) Weighted sum (MAC) operation in a conceptual crossbar array structure. **(B)** RRAM-based sub-array with peripheral circuits (e.g., CIM macro).

BACKGROUND

The convolution neural network (CNN) is one of the most popular DNN models, consisting of multiple convolutional layers to learn the salient features and a few fully connected layers for classification. In this study, we focus on the acceleration of the inference engine where the weights have been pretrained offline. In a convolutional layer, an output feature map (OFM) is the result of multiply-and-accumulate (MAC) operations on a collection of weights (or filters) operating in a sliding window fashion over the input feature map (IFM). Consider the case where the IFM of size W×W×D is processed by N filters, each of size K×K×D. Then the OFM of size W×W×N is computed as follows:

$$O[x][y][n] = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \sum_{k=0}^{D-1} I[x+i][y+j][k] \times W[i][j][k][n],$$

where I, W, and O are the IFM, weights, and OFM, respectively. CIM is an attractive solution for the extensive MAC operations in DNN inference as it combines memory access and computation. The conceptual crossbar structure for CIM is shown in **Figure 1A**, where the memory device is located at each cross point. If the weights are programmed as the conductance of the memory devices, when the input vectors encoded by read voltage signal, the weighted sum (MAC) operation can be performed in a parallel fashion and obtained as currents at the end of each column. Resistive random access memory (RRAM) is a two-terminal nonvolatile memory based on the metal/oxide/metal structure that stores the multi-bit weight by changing cell's multilevel conductance states. RRAM has been successfully demonstrated in industrial 40 nm (Chou et al., 2018)

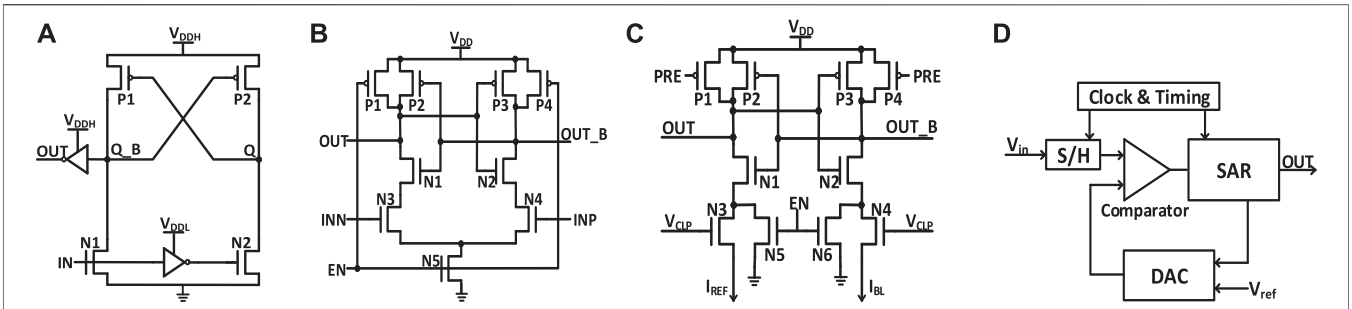


FIGURE 2 | Schematics of **(A)** level shifter; **(B)** voltage sense amplifier (VSA); **(C)** current sense amplifier (CSA); **(D)** successive approximation register (SAR) ADC.

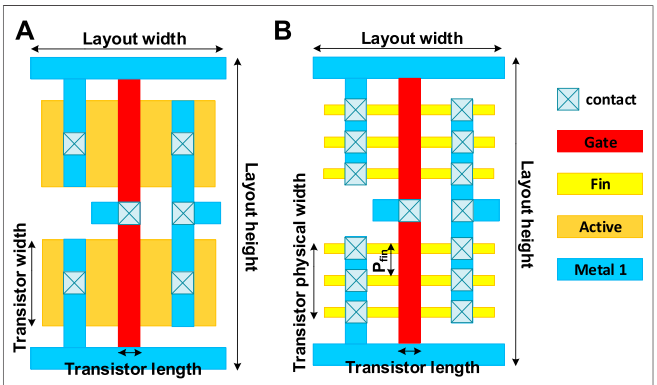


FIGURE 3 | Layout of inverter cells for **(A)** bulk and **(B)** FinFET.

and 22 nm platform (Xue et al., 2020). The one-transistor-one-resistor (1T1R) structure is widely used in RRAM-based CIM macro where the word-line (WL) to switch rows of cells and the MAC results are read out through bit-line (BL) voltage converted from weighted sum currents. As shown in **Figure 1B**, a complete RRAM-based CIM macro also contains peripheral circuits such as a WL switch matrix and BL/SL decoder (to select specific rows or columns), level shifter (to convert the logic V_{DD} to high write voltage for RRAM), MUX and its decoder, analog-to-digital converter (ADC), shift-add, and accumulator to support multi-bit input and multi-bit weight operations.

NEUROSIM SETTINGS

NeuroSim is designed for the CIM-based hardware accelerators. The hierarchy of the simulator consists of different levels of abstraction and analytical modeling from the memory cell and transistor technology to the gate-level standard cell and peripheral circuit modules and then to the one sub-array (or a macro as defined in this article). Then multiple sub-arrays will form one processing element (PE), and multiple PEs will form one tile with H-tree–based interconnect routing. An arbitrary neural network model could be mapped with a number of tiles.

New Features of NeuroSim

Compared with the last version of NeuroSim (Chen et al., 2018), many new modules and features are added in this version.

- Level-shifter is normally required for RRAM (or PCM/FeFET) array to support the need of higher write voltage (than logic V_{DD}). Now, a level-shifter is added as a peripheral module and will be validated later.
- Different types of ADCs are supported such as Flash ADCs using voltage-mode sense amplifiers or current-mode sense amplifiers and successive approximation register (SAR) ADC, as shown in **Figure 2**. They have trade-offs in the area/power and latency. For each technology node, latency and energy data from Cadence simulation are collected with sweeping of a reasonable dynamic voltage (or current) range and then are fitted with polynomial functions for fast

TABLE 1 | Updated transistor model of bulk (130–22 nm) and FinFET (14–7 nm) technologies.

Technology (nm)	Bulk						FinFET		
	130	90	65	45	32	22	14	10	7
Fin pitch (nm)							48	36	30
Fin height (nm)							37	42	52
Fin width (nm)							8	6	6
NMOS width of bulk (nm) / #Fin of FinFET	907	689	507	352	267	198	3	3	2
PMOS width of bulk (nm) / #Fin of FinFET	1,809	1,191	850	587	401	262	3	3	2
Gate length (nm)	75	55	35	30	28	26	22	20	18
Standard cell layout width (nm)	988	684	494	342	243	167	143	104	78
Standard cell layout height (nm)	3,640	2,520	1,820	1,260	896	616	462	336	250

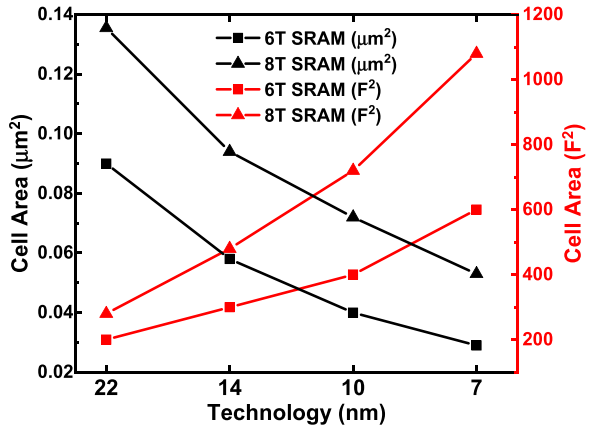


FIGURE 4 | Scaling trend of SRAM cell area with technology nodes (assuming F is the same as the technology node).

estimation of NeuroSim, given real traces from the workloads.

- Inverter, NAND, and NOR gates based on FinFET technologies (down to 7 nm) are optimized considering the layout rule. **Figure 3** shows the FinFET-based inverter gate layout. It should be pointed out that FinFET decouples the physical width (determined by the Fin pitch) and the electrical width (determined by the Fin height).
- The technology file is updated for FinFET. The default transistor models in NeuroSim were calibrated with the PTM model (PTM, 2011), which is available to the public and has a wide range of technology nodes from 130 to 7 nm. However, as the PTM model (of 14, 10 and 7 nm) was proposed far earlier than the industry adoption of FinFET, their prediction of Fin geometry actually deviated from the actual values today. We corrected the Fin height, width, and pitch following the recent trends in leading foundries and made some corresponding changes in the standard cell height/width and interconnect wire pitch, and switched to the assumption of using a maximum electrical width/or fin number in the standard cell for digital circuit design. The detailed values are shown in **Table 1**.
- A Scaling trend of the SRAM cell area with technology nodes is calibrated and shown in **Figure 4**. Since the technology node name F deviates from the transistor physical dimensions in the recent generations, the SRAM cell area that is normalized to F^2 significantly increases in 14 nm and beyond.
- The H-tree–based routing between memory arrays is optimized with a low-swing interconnect to improve energy efficiency.
- The extra-large SRAM buffers are split into smaller block buffers for a more realistic and efficient performance estimation.
- The peripheral mux used to be sized up significantly to avoid large voltage drop for a memory device with small on-state resistance (Ron). Considering the DNN model sparsity, the sizing of mux is decided by the average column resistance, instead of the worst-case all "on" resistance to alleviate the area overhead.
- Latency is measured by clock cycles, instead of directly accumulating the critical path of each module. The clock period is decided by the sensing cycle, which is the critical path from giving input to the memory array till the ADC generating the digital partial sum as this is an analog process and no digital buffer could be added in between. The latency of other digital modules is measured cycles needed for the processing because their timing could be adjusted by adding digital buffer.

Transistor and Peripheral Circuit Modules

The default transistor models in NeuroSim are calibrated with a predictive technology model (PTM) (PTM, 2011), which is available to the public and has a wide range of technology

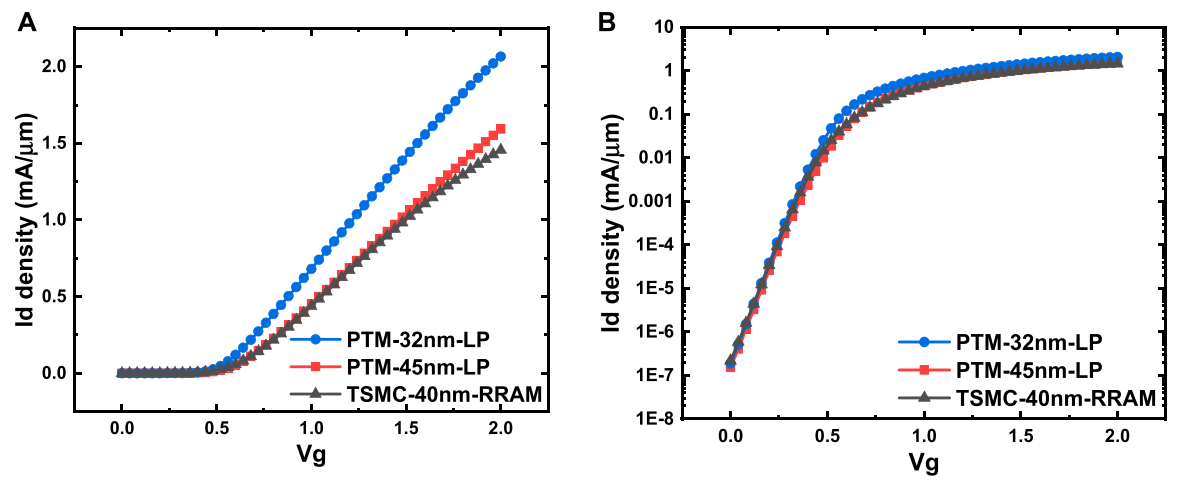


FIGURE 5 | I_d-V_g comparison of PTM model and TSMC PDK in **(A)** linear-scale and **(B)** semilogarithmic scale.

TABLE 2 | Table of simulator settings.

Technology	TSMC 40 nm w/RRAM
Array size	256 × 256 (only 128 × 128 in computation)
ADC precision	3-bit
Weight precision	1/2/4/8 bit
Operating voltage	0.9 V
Rows turned on simultaneously	7

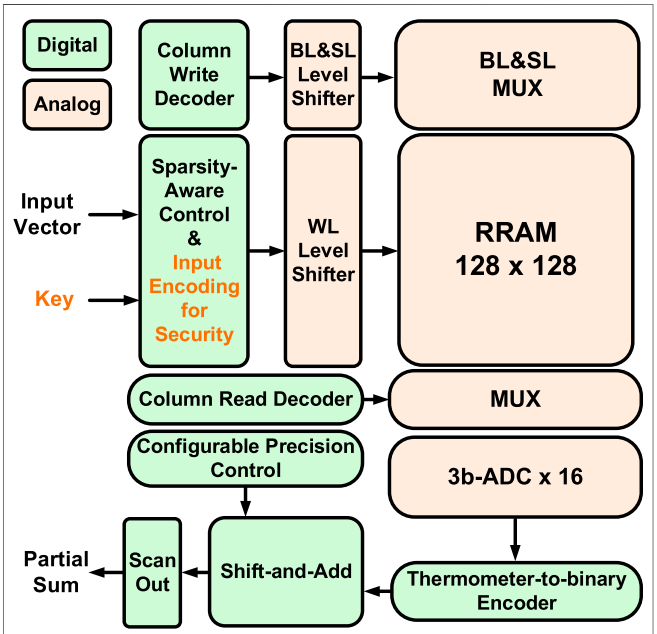


FIGURE 6 | RRAM CIM macro organization that supports input zero-skip and reconfigurable weight precision. ©2021 IEEE. Reprinted, with permission, from Li et al., 2021.

nodes from 130 to 7 nm. However, it is known that the commercial foundry process may differ noticeably from the PTM model. **Figure 5** shows the comparison of the I_d-V_g curve between the PTM model and TSMC PDK. In this validation, the transistor parameters are directly extracted from TSMC 40-nm RRAM PDK and specifically set in the NeuroSim transistor library, including device W/L, the supply voltage (V_{DD}), threshold voltage (V_{TH}), gate and parasitic capacitance, and NMOS/PMOS on/off current density. Based on these parameters, the area and intrinsic RC/power model of standard logic gates can be calculated analytically using the formula, as discussed in Ref. Chen et al. (2018); thus, the performance metrics of each sub-circuit can be estimated. The transistor W/L in ADC, mux, switch matrix, and drivers are predefined according to the required drivability, while transistor W/L in the other logic gates used fixed size (to be corrected later with validation). The capacitances at the logic gate level are also fixed with their transistors' sizing known, $\tau = RC$ and CV_{DD}^2 are calculated to estimate the module delay and dynamic energy consumption. Leakage power is also considered for sub-circuit modules and SRAM cells.

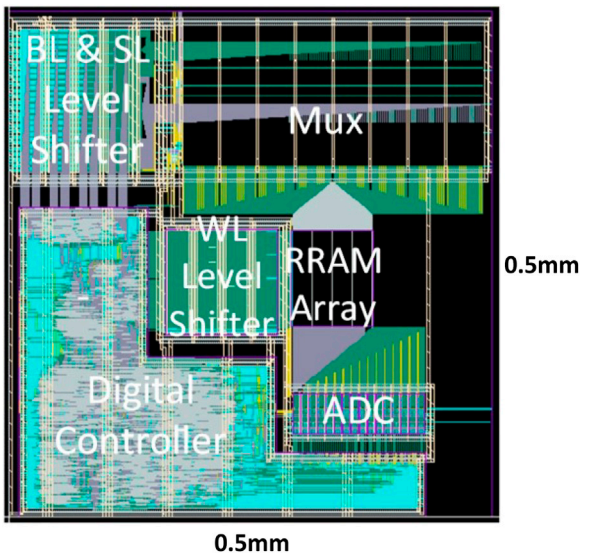


FIGURE 7 | CIM macro layout implemented with TSMC 40 nm RRAM process.

CIM Macro Configurations

In this particular design (Li et al., 2021) with TSMC 40-nm RRAM, the CIM macro could support MAC operation with zero-skip and reconfigurable precision for DNN inference. The input sparsity-aware controller counts the number of 1's in the input vector, and the scanned rows are asserted in parallel once the counter reaches the threshold (7 in this design, considering the ADC sensing range and the practical RRAM on/off ratio). By skipping the 0's in the input, only meaningful ADC conversions take place to improve throughput and energy efficiency. Flexible weight precision (1/2/4/8 bits) is supported to suit the optimized quantization levels for a variety of DNN models. On-chip shift-add and accumulator adaptively justify the different significances of weight bits and accumulate the partials sums in the digital domain. Each 3-bit ADC consists of seven voltage-mode sense amplifiers (VSAs) and is shared among eight columns as the RRAM cell pitch is much smaller than the size of the ADC. One reference voltage (Vref) is required for each VSA. For the ease of routing, the data column and the reference column are interleaved in a 256 × 256 physical array, but the actual computation array size is 128 × 128. Overall, the simulator settings are kept consistent with the actual macro and are summarized in **Table 2**. **Figures 6**, **7** separately show the macro organization and physical layout.

NEUROSIM VALIDATION

Analog Modules: RRAM Array, Level Shifter, Mux, and ADC

In the validation of analog sub-circuits, we mainly care about the RRAM array, level shifter, mux, and ADC. We will compare the area, latency, and energy consumption between NeuroSim simulation and the actual macro, as shown in **Table 3**.

TABLE 3 | Analog module validation.

Module	Area (um^2)		Latency (ns)		Energy (pJ)	
	NeuroSim	Real chip	NeuroSim	Real chip	NeuroSim	Real chip
Level shifter	256 × 28xαx3 = 30,966	12,084 (WL)+19,505 (BL+SL) = 31,589	1.12		1.02	0.99
Mux	459	5,400	0.06		0.07	
ADC	4,271		5		74.50	
Array	256 × 256 × 0.12 = 7,864	7,864	0.19		4.99	
(Mux decoder)	Counted in control part of digital modules		0.37		Counted in control part of digital modules	
(ADC encoder)			0.02			
Total	43,560	44,853	6.76xβ	~10	80.58	86.79

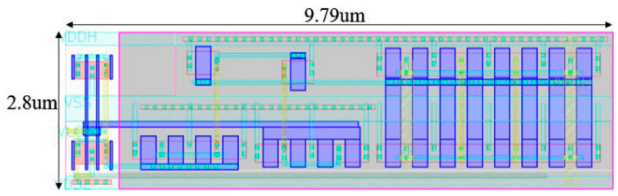


FIGURE 8 | Layout of level shifter.

Area: First, the RRAM cell size is a user-defined parameter in terms of F^2 (75 F^2 in this design) to estimate the array area according to the array size. In the simulator, the gate area is estimated according to transistor W/L and pitch requirements in the layout rules. In general, logic transistors with minimal length are utilized to constitute the sub-circuit modules. To simulate the I/O transistors in the level shifter, the gate layout width is multiplied by 2.5 times considering the poly width and the gap between gate polys in the PDK; the gate layout height is also multiplied 2.5 times to simulate the practical gate area measured in the macro design. After these corrections, the simulator shows 2.8 um × 10 um per level shifter unit, which is quite close to the measurement on the actual layout (**Figure 8**). There are totally 256 × 3 level shifters for WL, BL, and SL for the entire array size of 256 × 256. By comparison with the actual area measured in the layout, a wiring area factor α = 1.44 will be imposed on the level shifter for calibration. ADCs and their mux are located together on the layout occupying about 5,400 um^2 (the ADC block labeled in **Figure 7**), and the simulator estimates a result of 4,730 um^2 with acceptable error by its default settings. The other visible mux block labeled in **Figure 7** is for selecting the signal to BL and SL for programming the memory cells.

Latency: The chip could operate around 200 MHz with the digital blocks only, but the clock frequency drops to 100 MHz (post–layout simulation, 110 MHz for pre-layout) when the analog modules are included. It means the critical path is within the analog modules and it is the sensing delay from activating the level shifters to the currents summing along the columns till the ADCs converting the digital outputs. The sensing delay in the actual macro is ~10 ns. The latency of each module estimated by NeuroSim is listed in **Table 3**. A latency factor β = 1.4 will be utilized in the simulator based on the comparison.

Energy: The energy consumption of analog modules is measured by SPICE simulation. In NeuroSim, the energy estimation of ADCs is also based on a lookup table–like fitting function with various weight patterns and Vref swept that are predefined by SPICE simulations. Other digital-like modules utilize CV^2 as the dynamic energy estimation. Leakage power is also considered in NeuroSim, but the values are typically small. With precise settings demonstrated in *NeuroSim Settings* section, the estimation of NeuroSim is sufficiently accurate, as shown in **Table 3**.

Digital Modules: Shift-Add, Accumulator, and Controller

The breakdown performance of digital sub-circuits in the macro design is not easy to extract because they are together automatically synthesized through register transfer-level (RTL) codes. For simplicity, we consider digital modules as only three classes to be validated: shift-add, accumulator, and control circuits. The sparsity-aware controller, encoder, and decoders are all categorized as control circuits. It is noted that although zero-skipped input is supported in this macro to improve throughput and energy efficiency, our pre-layout SPICE simulation and NeuroSim estimation are both based on 0% input sparsity (no zero-skip).

Area: From the actual macro's digital design, the number of different types of gates and their corresponding areas can be extracted to validate the prediction. In order to support reconfigurable weight precision, the D-type flip-flops (DFFs) in the shift-add and accumulator are required to accommodate the largest precision (8-bit), and the adders have to be prepared for each precision (1/2/4/8-bit). We confirmed that the settings in NeuroSim could support the function and are similar as those in the actual chip, as shown in **Table 4**. In NeuroSim, the DFF contains four transmission gates, four inverters, and another four inverters for clock; the adder consists of nine NAND gates per bit. Although the exact number and types of gates cannot be guaranteed to be the same as the actual chip, the area comparison shown in **Table 5** is already close to the default models in NeuroSim. Unlike shift-add and accumulator, control circuits might consist of all types of gates and the composition can be quite diverse in different designs. Therefore, all the gates in the controller are normalized to the inverter gate count according to their area for simulation simplicity. The inverter layout height in NeuroSim is multiplied by 1.84 to mimic the

TABLE 4 | Shift-add and accumulator settings for reconfigurable precision.

Module		1-bit weight	2-bit weight	4-bit weight	8-bit weight	NeuroSim	Real chip
Shift-add	#DFF		64 registers × 5 bit/register	32 registers × 7 bit/register	16 registers × 11 bit/register	16 registers × 11 bit/register = 176	176 DFFs
	#Adder bit		16 adders × 3 bit/adder	16 adders × 3 bit/adder	16 adders × 3 bit/adder	16 adders × 3 bit/adder × 3 = 144	144 full adders +16 half adders
Accum	#DFF	128 registers × 8 bit/register	64 registers × 10 bit/register	32 registers × 12 bit/register	16 registers × 16 bit/register	128 registers × 8 bit/register = 1,024	1,216 DFFs
	#Adder bit	16 adders × 7 bit/adder	16 adders × 9 bit/adder	16 adders × 11 bit/adder	16 adders × 15 bit/adder	16 adders × (7 + 9+11 + 15) bit/adder = 672	704 full adders +112 half adders

TABLE 5 | Digital modules validation.

Module		Area (um^2)		Latency	Energy (pJ)	
		NeuroSim	Real chip		NeuroSim	Real chip
Shift-add	DFF	719	681	1 cycle = 10 ns	2.504 x γ = 1.25	1.25
	Adder	662	663		0.81 x δ = 0.12	0.15
	Inverter	2,133 INV = 1,336	1,334		3.31 x ϵ = 0.17	0.33
Accumulator	DFF	4,968	4,706	1 cycle = 10 ns	17.3 x γ = 8.65	8.37
	Adder	3,089	3,291		3.99 x δ = 0.60	0.60
	Inverter	10,869 INV = 6,808	6,797		16.88 x ϵ = 0.84	0.80
Control	DFF		7,334		26.96 x ζ = 2.97	3.01
	Inverter	10,485 INV = 6,569	6,558		16.28 x ϵ = 0.81	0.75
Total		31,893	31,386		15.41	15.26

TABLE 6 | Chip-level pre- and post-layout energy comparison.

Energy for whole array	NeuroSim	Real chip
Pre-layout	3,177.75 pJ	10.4TOPS/W → 3,150.8 pJ
Post-layout	3,177.75 pJ×η	8.48TOPS/W → 3,864.2 pJ

actual inverter area on this PDK. After the calibration, **Table 5** shows that the overall area estimation of digital modules is quite accurate.

Latency: As we pointed out earlier, the sensing cycle of the RRAM array is typically the critical path of the entire chip as the digital blocks can always be partitioned into multiple stages to be hidden within this analog critical path delay. Therefore, we propose counting the number of operations for digital modules. Each operation of shift-add or accumulator is one cycle according to the timing. For the entire DNN processing, we estimated the chip-level latency as the total number of clock cycles to complete the computation in a layer-by-layer manner.

Energy: **Table 5** shows the comparison of dynamic energy consumption of digital modules. The energy of actual chip is extracted from SPICE simulations. As most gates actually do not switch during run-time, switching activity factors should be considered in real workloads. As the DFFs are able to accommodate the largest precision, most DFFs are on operation when the real chip is tested under 8 bits. While the adders are prepared for each precision, most of gates are inactive in practice. Therefore, we set activity factors γ = 50% and δ = 15% separately for DFF and adder of shift-add and accumulator. The normalized inverters and DFFs to simulate the control circuits are employed with factor ϵ = 5% and ζ = 11%.

Post-Layout Calibration

The above performance comparison (except sensing delay) is based on pre-layout SPICE simulation. For chip-level energy efficiency, the actual macro could run at 10 TOPS/W with 0% input and 50% weight sparsity, where we can derive that it costs 3,151 pJ to compute the entire array (128 × 128 × 2 operations). As a comparison, NeuroSim predicts 3,178 pJ after the calibration. In order to reflect the silicon data, the post-layout performance drop is also considered in our validation, as shown in **Table 6**. In post-layout SPICE simulation, the macro has an energy efficiency of 8.48 TOPS/W with the same input and weight patterns, which derives that 3,864 pJ is required to compute the entire array. Therefore, a factor η = 1.22 is imposed to estimate the chip-level post-layout dynamic energy consumption.

BENCHMARK

In this section, we evaluate the impact of the aforementioned calibration factors on the DNN+NeuroSim framework by implementing the VGG-8 model on CIFAR-10 dataset, testing on various technologies and memory devices with a general architecture and operation mode, following the methodologies reported in Ref. Peng et al. (2019). The simulation is set up across versatile device technologies (HfOx RRAM (He et al., 2020), TaOx/HfOx RRAM (Wu et al., 2018), PCM (Kim et al., 2019), and FeFET (Ni et al., 2018), as shown in **Table 7**. SRAM-based

TABLE 7 | Benchmark results of CIM accelerators on VGG-8 for CIFAR-10 and ResNet-18 for ImageNet, based on SRAM (at 7 and 22 nm), and reported eNVM devices (assumed at 22 nm).

Technology node (LP)	**7 nm**	**22 nm**				
Device	**8T-SRAM**	**8T-SRAM**	**RRAM [12]**	**RRAM [13]**	**PCM [14]**	**FeFET [15]**
MLSA-ADC precision	4-bit	4-bit	5-bit	5-bit	5-bit	5-bit
Memory cell precision	1-bit	1-bit	2-bit	4-bit	4-bit	4-bit
Ron (Ω)	/		6 k	100 k	40 k	240 k
On/off ratio	/		150	10	12.5	100
VGG-8 (8-bit activation; 8-bit weight) on CIFAR10, with novel weight mapping and dataflow						
Area (mm^2)	13.34	61.92	45.55	25.57	25.57	25.52
Memory utilization (%)	98.73%	98.73%	96.86%	93.47%	93.47%	93.47%
Clock period (ns)	2.98	4.87	2.05	2.02	2.22	2.30
L-by-L latency (ms)	2.09	3.60	1.46	1.30	1.43	1.48
L-by-L dynamic energy (uJ)	31.27	58.69	37.75	16.73	17.32	16.07
L-by-L leakage power (mW)	2.71	1.73	0.63	0.33	0.33	0.33
Compute efficiency (TOPS/mm^2)	0.044	0.006	0.019	0.037	0.034	0.033
Pre-layout energy efficiency (TOPS/W)	31.87	18.48	31.63	71.25	68.69	73.72
Post-layout energy efficiency (TOPS/W)	26.12	15.14	25.93	58.40	56.11	60.43
Before calibration						
Area (mm^2)	13.34	60.25	31.18	17.88	17.88	17.64
Compute efficiency (TOPS/mm^2)	0.147	0.027	0.057	0.118	0.118	0.120
Energy efficiency (TOPS/W)	47.66	21.78	40.89	85.44	82.12	89.14
ResNet-18 (8-bit activation; 8-bit weight) on ImageNet, with novel weight mapping and dataflow						
Area (mm^2)	16.77	80.37	62.04	39.68	39.68	39.61
Memory utilization (%)	94.59%	94.59%	91.42%	86.64%	86.64%	86.64%
Clock period (ns)	2.98	4.87	2.05	2.02	2.22	2.30
L-by-L latency (ms)	22.75	39.14	13.39	11.53	12.67	13.12
L-by-L dynamic energy (uJ)	148.50	275.81	197.03	92.62	96.00	89.18
L-by-L leakage power (mW)	3.29	2.11	0.80	0.50	0.50	0.50
Compute efficiency (TOPS/mm^2)	0.014	0.002	0.007	0.012	0.011	0.011
Pre-layout energy efficiency (TOPS/W)	25.87	15.93	26.68	56.41	54.27	58.07
Post-layout energy efficiency (TOPS/W)	21.20	13.06	21.87	46.24	44.48	47.60

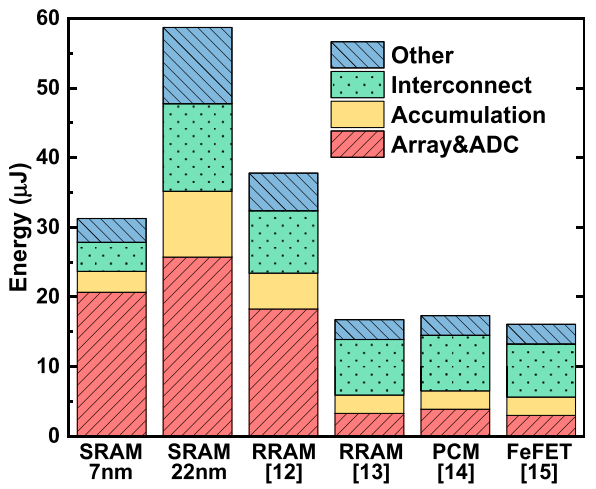


FIGURE 9 | Energy breakdown of CIM accelerators on VGG-8 for CIFAR-10, based on SRAM (at 7 and 22 nm), and reported eNVM devices (assumed at 22 nm).

CIM accelerators are evaluated at both 22 and 7 nm, and eNVM-based ones are evaluated at 22 nm as 22 nm is the state-of-the-art node where the eNVMs are integrated. Considering the read-noise and on/off ratio, the 4-bit/cell is assumed for eNVMs, except the 2-bit RRAM from Winbond (He et al., 2020). The subarray size is 128 × 128. A 4-bit precision ADC is utilized for 1-bit SRAM cells, with an inference accuracy of 92%; while a 5-bit precision ADC is utilized for multi-bit eNVMs to maintain an inference accuracy of 91% (Peng et al., 2019). Relatively high precision with 8-bit weight and 8- bit activation is also used to ensure no accuracy loss. A full 128-row parallel operation is assumed for the most efficient calculation. The number of operations is normalized to 8-bit, regardless of the memory cell precision.

The general conclusions stay the same as Ref. Peng et al. (2019). First, at the same technology node, eNVM-based designs outperformed the SRAM-based designs in both energy efficiency (in the unit of TOPS/W) and compute efficiency (in the unit of TOPS/mm^2). Second, devices with higher on-state resistance (Ron) such as FeFET show substantial improvements in energy efficiency. Third, SRAM at the leading-edge node (e.g., 7 nm or beyond) still show competitive energy efficiency and outstanding compute efficiency. Compared to the previous results before the validation, the new benchmark results show that the areas of eNVM-based designs are increased substantially owing to the calibration for the level-shifter area. The compute efficiency in all the design significantly decreases mainly because of the adopted clock cycle–based method to measure the latency. The

pre-layout energy efficiency is reduced mainly as a result of larger transistor size utilized after update, while the calibration on energy consumption of DFFs and adders somehow offset the more leakage caused by longer latency and the longer interconnect distance caused by the larger area. The post-layout energy efficiency is further dropped as a direct result of the calibration. The energy breakdown of simulated accelerators on VGG-8 for CIFAR-10 is shown in **Figure 9**. The devices with high Ron cost much less energy on the memory array charging and ADCs; devices with high cell precision could effectively reduce the operation of bit shift-and-add, thus reducing the energy consumption on accumulation; a smaller chip area contributes to less interconnection energy.

In this work, we also explore the scalability of the framework toward larger networks for more complex problem. The benchmark results of the ResNet-18 model on ImageNet dataset are also shown in **Table 7**, where the trend is similar as VGG-8 on CIFAR-10. The inference under 8-bit weight and 8-bit activation could reach 69% top-1 accuracy of ImageNet. The overall chip area increases by 25–50%, compute efficiency decreases by ~70%, and energy efficiency decreases by ~20% for ImageNet compared to CIFAR-10 workloads. In this version of the released framework, we assume a custom chip design for specific DNN models where all the weights are stored on chip. For the designs with chip area constraints where the weight reloading from off-chip DRAM is unavoidable, the readers could refer to the relevant discussions in Lu et al. (2020). For the reconfigurable chip design where one chip instance is able to support various DNN models, the readers could refer to the relevant discussions in Lu et al. (2021).

DISCUSSION

The related works in this field include the following reported simulators. NVSim (Dong et al., 2012) is a memory-oriented simulator, and its peripheral circuit modules do not support CIM functions. Other reported CIM-oriented simulator platforms such as MNSIM (Xia et al., 2018) and TxSim (Roy et al., 2021) have demonstrated powerfulness in the design space exploration or the device nonideality analysis, but they may have limited considerations either on the algorithm accuracy or on the hardware performance metrics. RxNN (Jain et al., 2020) is capable of various device and circuit nonideality analyses and rough energy estimation. Compared with RxNN, our work makes more comprehensive considerations on the hardware performance estimation. An IBM Analog AI HW Kit (IBM, in press) and CrossSim (CrossSim, 2018) only focus on the neural network accuracy estimation without the hardware performance estimation. PIMSim (Xu et al., 2018) is an architectural simulator for process in memory (most for near DRAM processing) with compatibility for traditional computer architecture simulator GEM5.

The prediction of NeuroSim is validated against the post-layout simulation of an actual 40 nm RRAM-based CIM macro design. Some adjustment factors are introduced: $\alpha = 1.44$ for the wire areas in the level shifter; $\beta = 1.4$ for the sensing cycle as the critical path; $\gamma = 50\%$ and $\delta = 15\%$ separately for dynamic energy of DFFs and adders in shift-add or accumulators; $\epsilon = 5\%$ and $\zeta = 11\%$ for dynamic energy of inverters and DFFs in control circuits; and $\eta = 1.22$ for a post-layout energy increase. After these calibrations, the chip-level simulation from NeuroSim is quite accurate with error under 1%.

However, we admit some inevitable limitations of this validation. First, the factors might be overfitted for this specific design. Limited by the available resources, it is unrealistic for us to have more chips fabricated with different technologies or design options. Although there are some other reported CIM macros developed by other groups, the lack of detailed design information and performance breakdown prevent using them for such validation. Second, even with our own CIM macro, the performance breakdown is not precise enough. For example, in NeuroSim, the latency is considered as the accumulation of the critical path delay of each module, while for the real chip, we could only get an overall estimation according to the clock cycle. Third, the calibration mainly focuses on the sub-array level as there is no large-scale multi-macro system with eNVM-based CIM accelerators as of today. Some additional factors may be required to capture the system-level activity rate of accumulators and buffer access frequency. Nevertheless, we believe this calibration with actual silicon implementation could offer an important reference and make the estimation of NeuroSim more convincing and reliable for the growing community of this simulator.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/neurosim. Open-source code availability: NeuroSim source code used in this work is publicly available at https://github.com/neurosim/DNN_NeuroSim_V1.3.

AUTHOR CONTRIBUTIONS

AL performed the validation, XP developed the simulation framework, WL and HJ designed the prototype chip, SY supervised the project, and AL and SY wrote the manuscript. All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

FUNDING

This work is supported by NSF-CCF-1903951 and ASCENT, one of the SRC/DARPA JUMP centers.

ACKNOWLEDGMENTS

The authors thank TSMC for providing the 40-nm RRAM tape-out shuttle.

REFERENCES

Burr, G. W., Shelby, R. M., Sidler, S., di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165 000 Synapses) Using Phase-Change Memory as the Synaptic Weight Element. *IEEE Trans. Electron. Devices* 62 (11), 3498–3507. doi:10.1109/ted.2015.2439635

Chen, P.-Y., Peng, X., and Yu, S. (2018). NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 37 (12), 3067–3080. doi:10.1109/tcad.2018.2789723

Chou, C.-C., Lin, Z.-J., Tseng, P.-L., Li, C.-F., Chang, C.-Y., Chen, W.-C., et al. (2018). "An N40 256K×44 Embedded RRAM Macro with SL-Precharge SA and Low-Voltage Current Limiter to Improve Read and Write Performance," in IEEE International Solid-State Circuits Conference (ISSCC). doi:10.1109/isscc.2018.8310392

CrossSim (2018). CrossSim. Available at https://cross-sim.sandia.gov/.

Deng, B. L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108 (4), 485–532. doi:10.1109/jproc.2020.2976475

Dong, Q., Sinangil, M. E., Erbagci, B., Sun, D., Khwa, W.-S., Liao, H.-J., et al. (2020). "A 351TOPS/W and 372.4 GOPS Compute-In-Memory SRAM Macro in 7nm FinFET CMOS for Machine-Learning Applications," in IEEE International Solid-State Circuits Conference (ISSCC). doi:10.1109/isscc19947.2020.9062985

Dong, X., Xu, C., Xie, Y., and Jouppi, N. P. (2012). NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 31 (7), 994–1007. doi:10.1109/TCAD.2012.2185930

Dutta, S., Ye, H., Chakraborty, W., Luo, Y.-C., San Jose, M., Grisafe, B., et al. (2020). "Monolithic 3D Integration of High Endurance Multi-Bit Ferroelectric FET for Accelerating Compute-In-Memory," in IEEE International Electron Devices Meeting (IEDM). doi:10.1109/iedm13553.2020.9371974

FreePDK (2014). FreePDK. Available at https://www.eda.ncsu.edu/wiki/FreePDK.

He, W., Yin, S., Kim, Y., Sun, X., Kim, J. J., Yu, S., et al. (2020). 2-Bit-per-Cell RRAM Based In-Memory Computing for Area-/Energy-Efficient Deep Learning. *IEEE Solid-State Circuits Lett.* 3, 194–197. doi:10.1109/LSSC.2020.3010795

IBM (in press). IBM Analog Hardware Acceleration Kit. Available at https://github.com/IBM/aihwkit.

Jain, S., Sengupta, A., Roy, K., and Raghunathan, A. (2020). RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 40 (2), 326–338. doi:10.1109/TCAD.2020.3000185

Kim, W., Bruce, R. L., Masuda, T., Fraczak, G. W., Gong, N., Adusumilli, P., et al. (2019). *Confined PCM-Based Analog Synaptic Devices Offering Low Resistance-Drift and 1000 Programmable States for Deep Learning*. IEEE Symposium on VLSI Technology. doi:10.23919/vlsit.2019.8776551

Li, W., Huang, S., Sun, X., Jiang, H., and Yu, S. (2021). "Secure-RRAM: A 40nm 16kb Compute-In-Memory Macro with Reconfigurability, Sparsity Control, and Embedded Security," in IEEE Custom Integrated Circuits Conference (CICC).

Lu, A., Peng, X., Luo, Y., Huang, S., and Yu, S. (2021). *A Runtime Reconfigurable Design of Comput-In-Memory Based Hardware Accelerator*. IEEE/ACM Design, Automation & Test in Europe (DATE).

Lu, A., Peng, X., Luo, Y., and Yu, S. (2020). Benchmark of the Compute-In-Memory-Based DNN Accelerator with Area Constraint. *IEEE Trans. VLSI Syst.* 28 (9), 1945–1952. doi:10.1109/tvlsi.2020.3001526

Ni, K., Grisafe, B., Chakraborty, W., Saha, A. K., Dutta, S., Jerry, M., et al. (2018). "In-memory Computing Primitive for Sensor Data Fusion in 28 Nm HKMG FeFET Technology," in IEEE International Electron Devices Meeting (IEDM). doi:10.1109/iedm.2018.8614527

Peng, X., Huang, S., Luo, Y., Sun, X., and Yu, S. (2019). "DNN+NeuroSim: An End-To-End Benchmarking Framework for Compute-In-Memory Accelerators with Versatile Device Technologies," in IEEE International Electron Devices Meeting (IEDM). Open-source code. doi:10.1109/iedm19573.2019.8993491Available at https://github.com/neurosim.

PTM (2011). Predictive Technology Model (PTM). Available at http://ptm.asu.edu.

Roy, S., Sridharan, S., Jain, S., and Raghunathan, A. (2021). TxSim: Modeling Training of Deep Neural Networks on Resistive Crossbar Systems. *IEEE Trans. VLSI Syst.* 29, 730–738. doi:10.1109/tvlsi.2021.3063543

Wu, W., Wu, H., Gao, B., Yao, P., Zhang, X., Peng, X., et al. (2018). *A Methodology to Improve Linearity of Analog RRAM for Neuromorphic Computing*. IEEE Symposium on VLSI Technology (VLSI). doi:10.1109/vlsit.2018.8510690

Xia, L., Li, B., Tang, T., Gu, P., Chen, P.-Y., Yu, S., et al. (2018). MNSIM: Simulation Platform for Memristor-Based Neuromorphic Computing System. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 37 (5), 1009–1022. doi:10.1109/TCAD.2017.2729466

Xu, S., Chen, X., Wang, Y., Han, Y., Qian, X., and Li, X. (2018). PIMSim: a Flexible and Detailed Processing-In-Memory Simulator. *IEEE Comp. Architecture Lett.* 18 (1), 6–9. doi:10.1109/LCA.2018.2885752

Xue, C.-X., Huang, T.-Y., Liu, J.-S., Chang, T.-W., Kao, H.-Y., Wang, J.-H., et al. (2020). "A 22nm 2Mb ReRAM Compute-In-Memory Macro with 121-28TOPS/W for Multibit MAC Computing for Tiny AI Edge Devices," in IEEE International Solid-State Circuits Conference (ISSCC). doi:10.1109/isscc19947.2020.9063078

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 29 June 2021
doi: 10.3389/fnins.2021.667011

Benchmarking Highly Parallel Hardware for Spiking Neural Networks in Robotics

Lea Steffen, Robin Koch, Stefan Ulbrich, Sven Nitzsche, Arne Roennau and Rüdiger Dillmann*

Interactive Diagnosis and Service Systems (IDS), Intelligent Systems and Production Engineering (ISPE), FZI Research Center for Information Technology, Karlsruhe, Germany

Animal brains still outperform even the most performant machines with significantly lower speed. Nonetheless, impressive progress has been made in robotics in the areas of vision, motion- and path planning in the last decades. Brain-inspired Spiking Neural Networks (SNN) and the parallel hardware necessary to exploit their full potential have promising features for robotic application. Besides the most obvious platform for deploying SNN, brain-inspired neuromorphic hardware, Graphical Processing Units (GPU) are well capable of parallel computing as well. Libraries for generating CUDA-optimized code, like GeNN and affordable embedded systems make them an attractive alternative due to their low price and availability. While a few performance tests exist, there has been a lack of benchmarks targeting robotic applications. We compare the performance of a neural Wavefront algorithm as a representative of use cases in robotics on different hardware suitable for running SNN simulations. The SNN used for this benchmark is modeled in the simulator-independent declarative language PyNN, which allows using the same model for different simulator backends. Our emphasis is the comparison between Nest, running on serial CPU, SpiNNaker, as a representative of neuromorphic hardware, and an implementation in GeNN. Beyond that, we also investigate the differences of GeNN deployed to different hardware. A comparison between the different simulators and hardware is performed with regard to total simulation time, average energy consumption per run, and the length of the resulting path. We hope that the insights gained about performance details of parallel hardware solutions contribute to developing more efficient SNN implementations for robotics.

Keywords: spiking neural networks, parallel hardware architectures, benchmark, robotic motion control, neurorobotic

OPEN ACCESS

Edited by:
Alexantrou Serb,
University of Southampton,
United Kingdom

Reviewed by:
Shuangming Yang,
Tianjin University, China
Chenchen Liu,
University of Maryland, Baltimore County, United States

****Correspondence:***
Lea Steffen
steffen@fzi.de

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *11 February 2021*
Accepted: *04 June 2021*
Published: *29 June 2021*

Citation:
Steffen L, Koch R, Ulbrich S, Nitzsche S, Roennau A and Dillmann R (2021) Benchmarking Highly Parallel Hardware for Spiking Neural Networks in Robotics. Front. Neurosci. 15:667011. doi: 10.3389/fnins.2021.667011

1. INTRODUCTION

Although impressive progress has been made in robotics, in the areas of perception and motor control, animal brains with significantly less performant components still outperform even the most sophisticated machines. While brain inspired research with Spiking Neural Networks (SNNs) and neuromorphic sensors shows great potential, their slow execution on common simulation frameworks running on serial CPUs prevent their broad application in robotic areas as vision, motion-, and path planning up to date.

As performing the updates for every neuron in parallel significantly reduces simulation time, hardware enabling massive parallelism enables the full exploitation of SNNs' capabilities. In order to allow researchers and developers to make a good choice regarding the software and hardware solutions when working with spiking neurons, comprehensive benchmarks are required.

Inspired by the nervous system, highly parallel platforms have been developed targeting low-power, large-scale SNN simulations in real time. The basis for biomimetic or neuromorphic hardware is the observation that the operation principles of information processing in nature differ greatly from artificial methods. In most cases, artificial methods are significantly less effective than their biological counterpart, for which reason scientists for instance, started to investigate retinal computation (Mead, 1990), inspired by processing in the brain. Well known representatives of this technology are IBMs TrueNorth chip (Merolla et al., 2014), Intel's Loihi (Davies et al., 2018), the SpiNNaker system of the University of Manchester (Furber et al., 2014), and BrainScaleS developed in Heidelberg (Friedmann et al., 2016).

TrueNorth is a digital neuromorphic chip that supports the simulation of up to 1 million neurons and 256 million synapses per chip. While the chip has a high density of neurons and synapses, one is limited to use the LIF neuron model and the synapses are static (Merolla et al., 2014). Loihi chips consist of 128 neuromorphic cores capable of simulating up to 1,024 LIF neurons each. The Loihi chip is fully digital and supports dynamic synapses (Davies et al., 2018). SpiNNaker is a fully digital neuromorphic system developed for large-scale simulations of SNNs. As it is composed of general-purpose ARM microprocessors, neuron and synapse models can be specified and adapted by software, making it very flexible (Furber et al., 2014). In Mayr et al. (2019), the second generation SpiNNaker-2, featuring 10 Million instead of 1 Million cores, is introduced. BrainScaleS, which is derived from the single-chip implementation Spikey (Schemmel et al., 2006), is a neuromorphic mixed-signal chip. In contrast to many others, BrainScaleS and its successor, BrainScaleS 2 are analog (Müller et al., 2020a,b).

Besides these popular candidates of non-von Neumann computing a multiplicity of neuromorphic hardware has been developed in the last decade. In Yan et al. (2021), a differentiation in 3 classes is made; (1) systems with static synapses like TrueNorth, NeuroGrid, Braindrop, HiAER-IFAT, DYNAPs, Tianjic, NeuroSoC, and DeepSouth, (2) systems supporting a configurable plasticity like ROLLS, ODIN, and TITAN and lastly, (3) systems supporting a programmable plasticity like both BrainScaleS, both SpiNNaker and Loihi. Another digital representative is focused on memory centric computing is Neurocube (Kim et al., 2016). However, since the development is progressing rapidly, a comprehensive list is difficult and quickly outdated. Comparatively new developments are the large-scale neuromorphic architectures CerebelluMorphic (Yang et al., 2021b) and BiCoSS (Yang et al., 2021a). CerebelluMorphic is a cerebellum-inspired neuromorphic architecture. Since the cerebellum is crucial for motor control, this technology is very interesting for robotics.

Another type of hardware well-suited for parallel computing is the Graphical Processing Unit (GPU), which was originally developed for computer game graphics. The introduction of high level programming languages such as CUDA or OpenCL, allowed GPUs to be used for general purpose parallel programming. This trend is supported by the development of accessible, inexpensive embedded systems like Nvidia's Jetson boards. Already in 2010 a parallel implementation of a SNN on NVidia CUDA showed a significant speed up (Nowotny, 2010). This idea was further investigated and in Yavuz et al. (2016) GPU enhanced neuronal networks (GeNN) a tool for code generation for specifying ANNs, especially focusing on SNNs, is introduced. It provides a simple C++ API generating optimized C++ and CUDA code. GeNN includes as well a C++ backend and CUDA backend and additional python module (PyGeNN) to support TensorFlow and PyNN. Further methods to execute ANNs on GPU are presented in Minkovich et al. (2014) and Mutch (2010). As stated in Vineyard et al. (2019), techniques for comparing neuromorphic architectures and similar systems are vital, as many event-driven methods may be well-suited for some, but inapplicable for others.

In Blundell et al. (2018), methods for code generation in computational neuroscience are reviewed and respective simulators, modeling languages, and frameworks are introduced and assessed. The authors cover, amongst others, code generations for a variety of hardware and software solutions like the neural simulators Brian (Stimberg et al., 2020), NEST (Gewaltig and Diesmann, 2007), NEURON (Carnevale and Hines, 2006), and GENESIS (Bower et al., 1998) running on serial CPU as well as techniques focusing on the execution on NVIDIA GPUs as GeNN (Yavuz et al., 2016) and Myriad (Rittner and Cleland, 2014). Furthermore, code generation for the neuromorphic hardware SpiNNaker (Furber et al., 2014) and the high-performance computing platform The Virtual Brain (TVB-HPC) (Sanzleon et al., 2013) are included.

In 2018, two benchmarks focusing on a neuroscientific use case, a cortical microcircuit model, have been presented. In van Albada et al. (2018), the models are implemented in PyNN and the performance of the SpiNNaker system running 6 SpiNN-5 boards and NEST running on a HPC cluster is compared. While the focus of the benchmark is on the accuracy of the simulation results, the total simulation time and energy per synaptic event are evaluated as well. Configurations of the HPC tuned for low energy consumption and simulation speed perform better than the SpiNNaker system. The simulations in both NEST- and SpiNNaker implementations are similar with regard to accuracy, hence showing the capabilities of the SpiNNaker system to perform large scale simulations. In contrast to van Albada et al. (2018), Knight and Nowotny (2018) uses C++ for simulating the cortical microcircuit model with GeNN on different pieces of hardware and comparing the performance to the SpiNNaker and NEST implementations. The authors state that—at least for their use case—certain GPUs outperform HPC systems as well as neuromorphic hardware regarding energy consumption and speed. As the accuracy of the simulation in GeNN is also comparable to the NEST implementation, GPUs are shown to be suitable architectures for SNN simulations that are able to compete with neuromorphic hardware. The

authors also outline the possibility of using GPU-based SNN controllers in robots. Benchmark scenarios more focused on a task applicable outside of the neuroscientific community are carried out in Diamond et al. (2016) and Ostrau et al. (2020). Both apply image classification and evaluate their models by using the MNIST data set. The former is executed on the neuromorphic systems Spikey and SpiNNaker, as well as GeNN. The latter uses Cypress and therefore the PyNN interface for NEST, Spikey and SpiNNaker and for GeNN and BrainScaleS the respective C++ interfaces are applied. In Diamond et al. (2016), the SNN used for image detection is a model of the olfactory system described in Schmuker and Schneider (2007).

As the work of Ostrau et al. (2020) shows some similarities to our work, it is noteworthy that they pre-learn several conventional ANNs which are transformed into SNNs. It is shown that the SpiNNaker system can be efficient if it is used to its full extend, whereas the GeNN implementations are most suitable if one focuses primarily on short simulation times.

Very recently, a comparison between Loihi and a prototype of SpiNNaker 2 has been done in Yan et al. (2021). Two kind of benchmark tasks are used, keyword spotting and adaptive robotic control to compare the hardware regarding power consumption and computation time. The authors conclude that a general statement is not possible as the energy efficiency is highly influenced by the number of input dimensions. As SpiNNaker 2 handles high-dimensional vector-matrices better, it is faster and more energy-efficient for keyword spotting. Loihi is superior in regard to less complicated vector-matrix multiplication. Furthermore, in DeWolf et al. (2020) a development workflow, targeting neurorobotics applications, running on standard as well as neuromorphic hardware, is presented. The authors illustrate how Nengo helps users to develop robotic sensor and actor applications, using two examples. The work creates a basis for benchmarking neuromorphic architecture, specifically Loihi, against standard hardware regarding robotic applications implemented in Nengo.

Even though applying SNNs to robotic use cases is very promising and despite the necessity of dedicated hardware for fast and resource-friendly execution, the performance of parallel hardware for SNNs has not been analyzed in the context of robotics sufficiently. One particular area of robotics, path finding, has not been investigated yet. In Davies (2019), an article giving guidance for benchmarking neuromorphic hardware, the temporal wavefront propagation was rated as an interesting candidate by name, as it is seen as a viable contribution to the greater field of neuromorphic benchmarks. While the authors of Yan et al. (2021) include a robotics scenario, their benchmark is limited to neuromorphic hardware. However, to develop efficient robotics solutions with SNNs, it is crucial to know specification- and performance-related details of all accessible systems to make an informed decision regarding hardware. Hence, we focus on an application-oriented robotic scenario. Using PyNN enables us to include representatives of GPU-based and neuromorphic computing as well as conventional simulators. Furthermore, this performance comparison covers several different GPU-based hardware realizations as the GeNN implementation is run on three candidates of the Jetson series by Nvidia.

In this work, we carry out a benchmark of hardware well suited for SNN simulation with an application-oriented test scenario intended to be used in robotics.

2. METHODS

To correctly derive how the different systems perform in comparison, it is crucial to run the experiments with a realistic workload. As this work compares parallel benchmarks for robotic applications, we chose the 3D neural path planning (Steffen et al., 2020), described in section 2.1, as the test scenario. It was chosen because path finding and motion control are corner stones for robotics. Due to their strong synergies, hardware and software in brain-inspired systems need to be considered together when selecting suitable candidates for benchmarking. In section 2.2, the decision-making process and its outcome are set out. As the architecture of parallel hardware, especially neuromorphic, is very different from the von Neumann architecture (VA), traditional benchmarks are hardly transferable to event-driven spiking use cases. Hence, meaningful metrics as introduced in section 2.3 are necessary.

2.1. A Robotic Scenario—The Wavefront Algorithm

A popular method for pathfinding is the so-called Wavefront algorithm. It represents the environment as a matrix Map(i, j). Each free cell has a value assigned to it which represents its distance from the target cell and is also referred to as the weight. Its value corresponds to the minimal value of its neighboring cells +1. If a cell is occupied it is not assigned value. This can be formalized by:

$$Map(i,j) = \begin{cases} min(neighborhood(i,j)) + 1 & if\ empty \\ nothing & if\ full \end{cases} \quad (1)$$

In order to find the shortest path the mobile agent then simply follows the cells with the smallest weights until it reaches its target (Nooraliei and Nooraliei, 2009; Pal et al., 2011).

In Steffen et al. (2020), the neural path planning algorithm for robotic motion control is introduced. This implementation is the test scenario of our benchmark. The method generates a synaptic vector field (SVF), revealing a path, by propagating a wavefront on a 3D environment. The environment is represented as a cognitive map, a grid of excitatory place cells realized as an SNN. The method, based on the 2D variation proposed in Ponulak and Hopfield (2013), applies bio-inspired techniques and is especially interesting for reactive flexible motion control as needed for Human-robot interaction. The implementation of Steffen et al. (2020) is carried out in NEST and tested on maps with varying degrees of complexity. The NEST implementation already allows fast simulation and query times but shows strong weaknesses regarding the creation time. The purposeful use of dedicated hardware, allowing massive parallelism, shall overcome these issues. A detailed visualization of the implemented algorithm is given as a sequence diagram in **Figure 1A**. As the evaluation in section 3 embodies a specific analysis of the

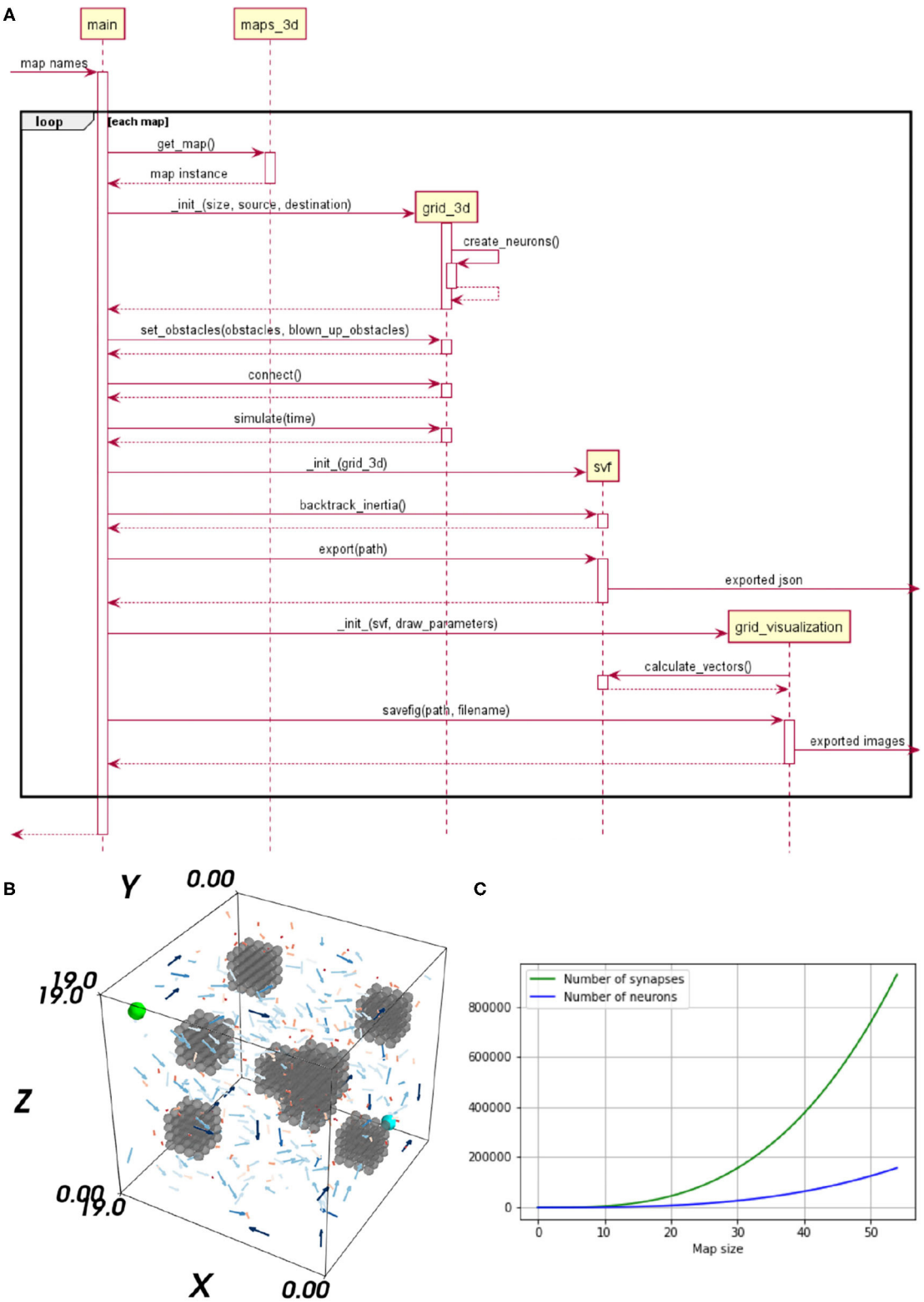


FIGURE 1 | (A) Sequence diagram of the neural Wavefront algorithm for robot motion control, as presented in Steffen et al. (2020). **(B)** A vector field's flow on a map including three obstacles. The start neuron is marked in cyan and the target neuron in light green. **(C)** Impact of an up scaled map on the number of neurons and synapses is shown. The x-axis represents the map size and the y-axis shows the number of neurons and synapses, respectively.

algorithm's sub-tasks, a brief explanation of the significant sub-processes is provided.

2.1.1. A Cognitive Map

The neural environment representation is realized as an SNN using place cells. The network's topology corresponds to a discretization of the environment. Each voxel is translated as a place cell implemented as a single neuron. The network's neurons are connected via the Manhattan method, solely supporting lateral connections. As we use bi-directional connections the synapses are not symmetrical. The neurons representing free and occupied space are identical. However, neurons embodying free space are connected by excitatory synapses and neurons embodying obstacles by inhibitory ones.

2.1.2. Synaptic Vector Field

The SVF is an interpretation of the synapses' weights, of the trained network. For learning spike-timing-dependent plasticity (STDP), a biologically plausible learning rule updating the weights depending on the precise pre-synaptic and post-synaptic spike times (Gütig et al., 2003) is used. To generate a SVF with neural waves, three steps are required. *(1) Initialization*, the place cell representing the target position triggers the neural wave. Applying an electrical current to the respective neuron, increases its membrane potential. Thereby a spike is emitted, exciting the neuron's neighbors and so starting the wave of activation which evolves through the neural grid. *(2) Learning*, through the learning rule STDP the synaptic weights are altered in the direction of the wave. *(3) Interpretation*, by retrieving the synapses' weights as vectors, a vector field is generated. This vector field is used for visualization purposes and for finding a path. In **Figure 1B**, a 3D visualization of the SVF is provided. The Vector's length and color provide information about the strength of their weights. Shorter, darker vectors stand for weaker synaptic weights and longer, brighter vectors, indicate strong synaptic weights. For clarification, only 5% of the vectors are visualized in **Figure 1B** and the length of all vectors has been doubled to increase their visibility. However, this does not affect their expressiveness as their relative length is still meaningful with respect to their strength. It can be seen that the vectors are pointing in the direction of the start.

2.1.3. Path Search

The synaptic weights, building up the vector field, are interpreted as forces used to move the agent and thus generate a path. The resulting path naturally leads away from obstacles as synapses connecting two neurons of the free space are stronger than synapses between free and occupied neurons. By averaging over the local vectors at each step local minimas can be avoided. The resulting force vector is subsequently added to the previous movement direction.

2.2. Tools and Techniques

Three different simulators, or rather hardware solutions, have been selected for the benchmark, representing three strongly deviating approaches for simulating spiking neurons. NEST is chosen as an actual simulator and the SpiNNaker system as a representative for neuromorphic architectures. GeNN constitutes a recently developed alternative running on conventional parallel hardware. GeNN can be used on a GPU but also in a CPU-only mode, which allows it to run on a broad range of hardware from desktop PCs to embedded systems. We evaluate GeNN with both, its CPU-only and GPU version. The CPU-only mode allows to compare the performance to the results obtained with the NEST simulator which were presented in Steffen et al. (2020). Both, GeNN on CPU and NEST, are run on a single processor core. A visualization of all applied hardware and software solutions is provided in **Figure 2**. As this paper aims to evaluate the performance of systems in context of robotics, the Jetson series by Nvidia is chosen as a hardware backend for GeNN on GPU. The Jetson series consists of several different embedded GPU systems, which were designed with the goal of supporting AI solutions in hardware with a small form factor. This enables the Jetson chipset to be integrated into mobile units (Franklin, 2018). The boards are general purpose hardware and are both more widely available and cheaper than the specialized neuromorphic hardware. Three different Jetson boards are evaluated in this benchmark, the Jetson Tx2, AGX Xavier, and Xavier Nx. In order to make the results of the embedded systems more comparable, a regular desktop PC is included in the analysis. The desktop PC is also used to run the NEST simulations. The PC has 32 GB of RAM and contains an Nvidia RTX2070 GPU and an AMD Ryzen 7 3700x CPU. A SpiNN-5 Board is used to run the SpiNNaker implementation.

2.2.1. Hardware Specific Adaptations

In order to allow a fair comparison between the simulators, the benchmarking scenario needs to be implemented in a similar way on all platforms. PyNN offers the possibility to use the same model for all simulators. However, as noted in Diamond et al. (2016) this implies the risk that individual strengths of the simulators are not accounted for. Despite this issue, in this paper, all SNNs are modeled with PyNN, whereas Ostrau et al. (2020) opted to use the Cypress library and Diamond et al. (2016) decided to model their networks in the native modeling languages of the simulators instead of using their PyNN interface. However, using PyNN to model all networks allows to simulate exactly the same model on all backends. **Figure 2** provides an overview of the hardware and software applied for this benchmark.

The benchmark is based on the NEST implementation of Steffen et al. (2020). Since all three models are implemented in PyNN, in theory it would be sufficient to specify a different PyNN backend. However, the different backends implement divergent subsets of PyNN's functions and models. Therefore, changes to the code are required to run the simulation on the different backends. However, it is also note-worthy that there are some possibilities to tailor the simulation more to the individual system. This allows to better account for individual strengths and weaknesses of each architecture. A big difference in the implementation of the simulators is the step size. While SpiNNaker uses a time step of 1 ms, GeNN and NEST simulations are usually run at time steps of 0.1 ms. This is due to the fact that 0.1 ms is the time step most often used for neuroscientific

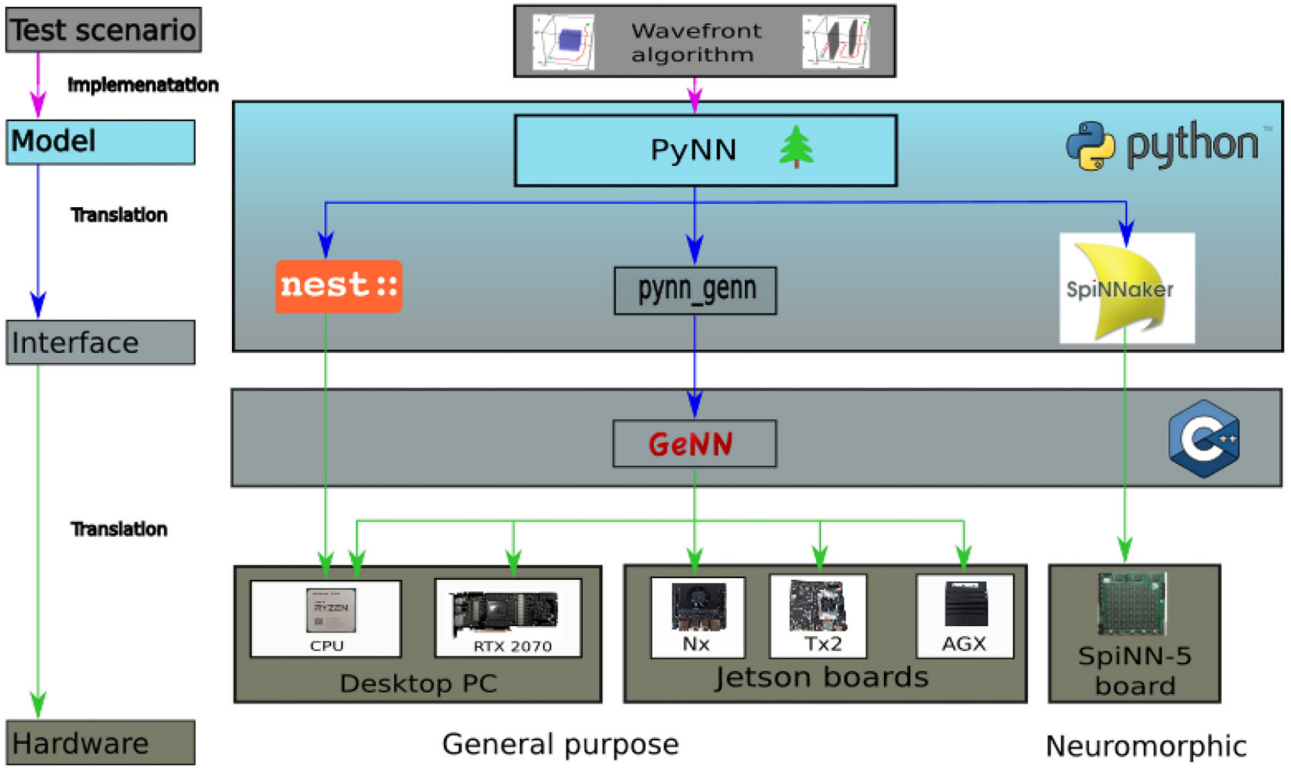


FIGURE 2 | Overview of the hardware and software applied for this benchmark. On the top level the Wavefront algorithm representing the test scenario is featured. In the model layer below it is visualized how the algorithm is implemented via PyNN running in python. On the interface level the model is translated to the respective back ends and subsequently transferred onto the final hardware layer for execution.

simulations (van Albada et al., 2018). The default time step is chosen for each simulator. The Wavefront algorithm is intended to solve a path planning problem in a time efficient manner, therefore, artificially slowing down the simulations in GeNN and NEST would not be realistic and distort the results. The time step of the simulation running on SpiNNaker is not lowered to match that of the other two simulators, as this would result in a slowdown of the simulation by a factor of 20. To account for the different time steps chosen, different total simulation times are required to allow the simulations to finish on the different platforms.

The original implementation of the neural wavefront algorithm presented in Steffen et al. (2020) uses the `IF_cond_alpha` model, which is a standard model in PyNN. It implements a LIF neuron with an alpha function to describe its post synaptic potential. This neuron model is currently not supported in sPyNNaker. The neuron model was thus changed to `IF_cond_exp`, an LIF neuron with an exponentially decaying post-synaptic potential. The SpiNNaker system has an additional constraint, the representation of synaptic weights as 16 bit integers. To convert the weights, a bit shift operation needs to be performed (van Albada et al., 2018). The algorithm used to determine the bit shift, does not allow the original maximum synaptic weight of $w_{max} = 4000.0$ µS. The highest maximum weight that can be implemented is $w_{max} = 63.0$ µS. The Wavefront Algorithm uses STDP with additive weight dependence. The value of $w_{max} = 63.0$ µS is too low, for the additive weight dependence to induce enough weight change to create the correct SVF. Therefore, for the implementation in sPyNNaker, the maximum weight is scaled down to $w_{max} = 63.0$ µS. After the simulation, the weight is scaled back up by a factor of $f_{scale} = 4000.0/63.0$. The re-scaling is not entirely correct, as the STDP rule introduces an additional term that cannot be properly re-scaled by this method. It would be possible to re-scale the weights correctly by sampling the weights before and after simulation, calculating Δw and scaling it independently. However, this would introduce additional overhead in the pathfinding phase of the algorithm. In practice, the re-scaling, even though not entirely correct, still produces acceptable results. In the original implementation, the target neuron which starts the neural wavefront receives a DC current as an input. The current increases the neuron's potential inducing a spike. DC current sources are currently not implemented in sPyNNaker. Hence, the spike in the target neuron is induced using a neuron population of type `SpikeSourceArray`. The `SpikeSourceArray` population can then be connected to specific neurons via a projection. In this implementation the `SpikeSourceArray` population consists of one neuron that is connected to the target neuron with a static synapse. Instead of building up the membrane potential in the target neuron, a spike

TABLE 1 | Overview of all system specific adaptations for each NEST, SpiNNaker, and GeNN, in respect to the original implementation of Steffen et al. (2020).

Features	Original implementation	NEST	SpiNNaker	GeNN
Neuron model	`IF_cond_alpha`	`IF_cond_exp`	`IF_cond_exp`	`IF_cond_exp`
Step size	0.1 ms	0.1 ms	1 ms	0.1 ms
Weights	Unscaled	Unscaled	Scaled	Unscaled
Spike source	DC source	DC source	SpikeSourceArray	DC source

is induced right away. This is advantageous as the wavefront can be initiated right after the first time step. The spike source array introduces a small overhead for the simulation time for small maps on `pynn_genn`. To induce almost instantaneous spiking, the current in the DC source is set to 1,000 mV. A complete overview about all system specific adaptations is given in **Table 1**.

2.2.2. Measurements

In order to gather data about performance, several measurements take place inside the simulation code. The individual functions `Create neurons`, `Create synapses`, `Simulation`, `Build SVF`, `Compilation/Load Simulation`, and `Finding path` are timed with the help of a python wrapper function. It saves the time stamp before the function is started and again when the function terminates. The delta is the execution time of the function and labeled with the function's name. In NEST the initialization of neurons and synapses in the code causes them to be initialized as soon as their respective definitions are executed. With sPyNNaker and `pynn_genn` the values are instantiated as well, however, when the `run()` function is executed, the synapses are instantiated again. This is due to the fact that they need to be placed onto the vertices of the MachineGraph in case of sPyNNaker, and executed in C++ or CUDA C in case of `pynn_genn`. The `run()` function combines the loading and running of the simulation in sPyNNaker and the compilation and running of the simulation for `pynn_genn` into one step. This makes it impossible to determine the exact time needed for the individual steps with the help of the wrapper. To determine the time required for loading and compilation, the sPyNNaker and GeNN simulation are initially run for 0 s. This triggers the loading and compilation respectively, but does not start the wavefront. When running the simulation on SpiNNaker again, the entire loading process is restarted. To avoid that the loading time is measured twice, the created logs of the SpiNNaker machine are read out. The logs contain timestamps that allow to determine correct start time. This timestamp is then used with the end timestamp of the python wrapper to determine the simulation time. The path length is determined during simulation time by obtaining the length of the list containing the path.

2.2.3. Map Scaling

To compare how the different hardware solutions handle an increasing number of neurons and synapses, the networks are scaled up. As the network is a direct neural representation of the environment, enlarging the maps implies a likewise grow of the neural embodiment. The original maps, used in Steffen et al. (2020) of size $20 \times 20 \times 20$, served as the reference value for the smallest maps. How an enlarged map influences the number of neurons and synapses within the network is visualized in **Figure 1C**. The number of synapses increases exponentially while the number of neurons is increased in a cubic manner, when the size of the map is scaled up. A prior examination regarding the maximum map size supported on each hardware specific implementation is required. The map size is limited by the available memory of the respective hardware architecture, for GeNN und NEST. For SpiNNaker, the long simulation times associated with larger maps limited the size of the network.

2.3. Metrics

To measure the performance of the applied hardware solutions, several metrics are introduced. As stated in Vineyard et al. (2019) it is practically meaningless to compare parallel architectures using the same metrics as applied to conventional VAs. Due to the architectural approaches being designed and optimized for different use cases, it is challenging—but absolutely necessary—to choose appropriate metrics enabling a solid understanding of their advantages and trade-offs.

2.3.1. Simulation Time

It is not insightful to compare only internal metrics of the systems such as the speed of the processor clock. Instead, a comparison indicating how the systems perform when given an actual task is required. The Wavefront algorithm aims to solve the pathfinding problem as fast as possible and ultimately in real-time. Hence, the most interesting metric to consider is the execution time. When considering the time, not only the actual execution time is of interest, but also the amount of time needed to load or compile the simulation.

2.3.2. Energy Consumption

The second meaningful metric is the energy consumption, more precisely, the average energy needed per run. For robots in general—but particularly for mobile robots—it is important that the individual components do not consume much energy. The measurement of the energy consumption is carried out externally. In literature, two methods for obtaining data regarding energy consumption are common. Firstly, a consumer grade power meter that allows to store time stamped data which can be extracted via an SD card. In Ostrau et al. (2020), the Ruideng UM25C power meter and PeakTech9035 power meter are used. Secondly, image recordings of the power meter with subsequential digital post-processing. In Diamond et al.

(2016), van Albada et al. (2018), and Knight and Nowotny (2018) cameras are used to record the display of the power meter. A digital post-processing is required to obtain meaningful energy data.

For this benchmark, the energy consumption is measured using a household energy meter of type Voltcraft 4000 Energy logger that allows to store and extract time stamped data. The power meter has a minimal resolution of 0.1 W and an accuracy of ±1% and one count for the expected power draw. The power meter samples the power draw only once per minute and saves the data internally with a time stamp. The internal storage is extracted with the help of an SD card and then subsequently converted into csv format by the Voltsoft software that comes with the Voltcraft 4000 Energy Logger. In order to obtain the total energy consumption, the values are integrated using the numpy trapz function. The time step is set to 60s.

2.3.3. Path Length

The path length is measured during simulation, after a path has been successfully determined. As the path is saved as a list, its longitude is simply the length of the list. To compare the length of the paths found on the different implementations, the median of the path lengths is taken over all runs on a particular map for each hardware solution. This allows to quickly determine if the path length differs throughout runs on the same simulator.

Comparing the path length determined on the different hardware solutions poses an issue. On SpiNNaker the weights differ because they are converted via bit shifting, causing some small errors, which are then amplified by the scaleup later in the process. Ultimately, we would use the path length to compare the accuracy of the different STDP implementations. In other words, we investigate if in case all simulators get the same initial weights, do they have the same final weights after learning via STDP.

2.3.4. Hardware Resources

In addition to the main metrics, the consumption of different hardware resources, namely the memory usage and CPU/GPU of the Desktop PC and the Jetson boards are measured. This is done with the help of logging software. For the Desktop PC, glances[1] is used for CPU and Memory as well as NVIDIA System Management Interface (nvidia-smi[2]) for the GPU data. On the jetson boards a logging script based on jetson-stats[3] is run to log the memory usage.

3. EXPERIMENTS AND RESULTS

3.1. Comparing Different Hardware Solutions

The total time of the simulation includes the time measurements of all functions that are needed to execute the Wavefront algorithm. **Table 2** gives an overview of the median total time, path length, and average energy consumption per run. However, as the comprehensive version is very long and detailed, **Table 2** is only an excerpt. It comprises SpiNNaker, NEST, and GeNN. The GeNN implementation was carried out on an AGX Xavier. The complete table including data for GeNN on RTX2070, in CPU-only mode, on a Tx2 and a Xavier Nx is provided as **Supplementary Material**. The detailed analysis in this paper is focused on map IV from Steffen et al. (2020), since this is the most complex. Tests on other maps presented in Steffen et al. (2020) show similar results, indicating that the statements can be generalized. As not to go beyond the scope of the work these additional results are not presented in this paper.

TABLE 2 | An overview of the results of the simulations on the map IV.

	Map size	Total time [s]	Path length	Average energy per run [J]
GeNN	20	54.13	36.0	2254.65
	25	85.38	40.0	4358.44
	28	116.33	46.0	5521.26
	30	143.63	49.0	7169.49
	33	201.29	50.0	2347.63
SpiNNaker	20	64.62	32.0	13132.09
	25	128.50	49.0	23881.05
	28	171.26	44.0	14484.20
	30	162.46	50.0	13593.85
	33	259.03	53.0	21046.42
	35	324.96	55.0	24599.44
	40	368.47	72.0	30573.19
NEST	20	8.39	38.0	1861.54
	25	18.34	42.0	1908.04
	28	26.76	41.0	2453.65
	30	35.20	48.0	3089.95
	33	50.28	48.0	4294.51
	35	72.84	58.0	6126.01
	40	114.15	65.0	7641.54
	45	180.23	66.0	12266.63
	55	421.65	81.0	50946.67

The median total time, path length, and average energy consumption per simulation run is listed for each implementations. The GeNN version was carried out on an AGX Xavier. Data regarding GeNN executed on other hardware solutions is given as ***Supplementary Material****.*

The times for the individual functions `Create neurons`, `Create synapses`, `Simulation`, `Build SVF`, `Compilation/Load Simulation`, and `Finding path` are at first evaluated separately for every map. The median, which is more robust against outliers than the mean, and the standard deviation of the time are determined for every function separately. In order to obtain a total time, as provided in the second column of **Table 2**, the duration for the individual functions are summed for every run and subsequently.

The implementations running on the desktop PC are generally faster than implementations on other hardware. With total simulation times between 7.20 and 409.27 s, the implementation of GeNN on the CPU has the shortest total time of all implementations, followed by the implementation in NEST (8.39–421.65 s). Both implementations run on a single thread on the CPU of the desktop PC. The GeNN implementation

[1]https://nicolargo.github.io/glances/.
[2]https://developer.nvidia.com/nvidia-system-management-interface.
[3]https://github.com/topics/jetson-stats.

running on the RTX2070 GPU has slightly longer total simulation times than the two implementations on the CPU. The implementations running GeNN on the Jetson Boards take significantly longer than the implementations on the desktop PC. The total simulation time of map size 30 on the RTX2070 is still considerably shorter than the total simulation time for map size 20 on any of the Jetson boards. The AGX Xavier has the best total simulation times out of the three boards, followed by the Xavier NX. The implementation running on the SpiNNaker board has worse total simulation times than the AGX Xavier, beating the Xavier NX and the Tx2, with the Tx2 being the slowest out of all hardware systems tested. The GeNN implementation running on the CPU scales better than the NEST implementation. Both are showing an exponential increase in total simulation time with increasing map sizes. For the AGX Xavier and Tx2 a similar trend can be observed, however, the trend is less noticeable due to the limited map sizes. For simulations with GeNN on the Xavier NX and the RTX2070, the total time appears to increase linearly. Some linear segments can be observed when looking at the scaling of total time on the SpiNNaker implementation. There is, however, a very striking deviation from the scaling, the total time decreases when the map size is increased from 28 to 30.

3.2. Performance Analysis of Isolated Functionalities

Additionally to the total time measurements in **Table 2**, **Figure 3B** shows how the individual functions scale with increasing map size for the implementation of GeNN running on the CPU. One can see that the exponential component in the scaling of total time is caused by the creation of synapses. In **Figure 4A**, the proportion for synapse creation is shown to increase considerably with increasing map sizes, eventually outweighing compilation time. For a map size of 55, synapse creation makes up 75.01% of the total time. On smaller maps, the compilation time dominates the total time. The proportion of the function `Simulation` increases only slightly with an increase in map size. Except for the creation of synapses, all other sub functions increase linearly with increasing map sizes. Compared to the implementation of GeNN on the RTX2070, the compilation times for GeNN on CPU are shorter. As shown in **Figure 4B**, on the RTX2070, compilation and loading times make up by far the largest fraction of the total time with 77.63% for a map size of 20. The proportion of compilation and loading time decreases with increasing map sizes, however, it still makes up almost 50% of the total time. Creating synapses takes more than twice as long as the simulation of the SNN itself for maps of all sizes. The development of the individual functions is illustrated in **Supplementary Material**. All sub functions scale linearly with increasing map sizes, except for the synapse creation, where the beginning of an exponential increase can be observed.

The proportions of individual functions on the Jetson boards show a pattern similar to the RTX2070. **Figures 3B,D**, **6B,D** show how the time of individual functions scale with respect to increasing map sizes[4]. The function `Compilation and loading` and `Simulation` show a linear increase for the Tx2 and the AGX Xavier, whereas `Create synapses` shows the beginning of an exponential increase when the map is scaled up. On the Jetson Xavier Nx, the function `Create synapses` scales linearly. The time required for compilation/loading reaches a plateau at maps of size 30. The time required to build the SVF remains almost constant on the AGX Xavier for all map sizes. For the Tx2 and Xavier NX, however, a strong increase is measured for the largest map size. The proportions of the different sub-functions are visualized in **Figure 5**. With 76.6, 73.8, and 69.3% compilation and loading time represents the largest part of the total time on all three Jetson boards for map size 20. The proportion of the compilation and loading time decreases when maps get larger, however, they still make up around 45% of the total time and are almost two times higher than the actual simulation of the SNN. The function to create synapses also takes longer than the simulation time itself. With exception to the Xavier Nx on map size 20, synapse creation takes more than twice as long as the actual simulation for all Jetson boards on all map sizes. The time required to build the SVF increases with larger maps in proportion to the total time. On the Xavier Nx, in particular, a strong increase can be observed for maps of size 30.

The latest update of `pynn_genn` introduced the `reuse_model` flag. I allows for the CUDA backend to reuse the model of a previous run of a simulation, if the same network is used. Most of the generated code can therefore be reused and does not need to be compiled again, thus significantly saving compilation time. For the simulation running on the RTX 2070 the compilation and loading time could be reduced by 8.01 s for the smallest map and 7.84 s for the largest map which accounts for 72.1 and 44.4% of the simulation and loading time. For the Jetson AGX Xavier a reduction by 24.51 s for the smallest map and 27.76 s for the largest map were observed which amounts to a reduction by 61.9 and 22.1% of the simulation and loading time.

Figure 4C shows that for the NEST implementation. The total time is dominated by the creation of synapses which makes up between 84.3 and 89% of the total simulation time for all map sizes. This reflects the results in Steffen et al. (2020), where the creation of synapses also contributes the most to simulation time. For the SpiNNaker implementation (see **Figure 4D**), the simulation and loading times far outweigh the time it takes to create the synapses, which is very similar to the observed behavior of GeNN on the CPU (**Figure 4A**). Similar times for `Create synapses` are expected, as the function is executed on the CPU of the desktop PC in both cases. However, on the SpiNNaker board, `Create synapses` makes up a much lower proportion of the total time, ranging from 3.3% to 15.6% for the smallest and largest map, respectively. The time it takes to load the simulation and to build the SVF makes up the highest proportions of the total time. As illustrated in **Figure 6B**, the sudden decrease of total simulation time, that can be observed in **Figure 6A**, appears to be mainly due to a decrease in the time it takes to build the SVF. It contributes up to 59% to the total time, which is a much larger proportion than the NEST

[4]Plots regarding the scaling properties of GeNN on Tx2, Xavier NX, and RTX2070 (desktop GPU) are provided in **Supplementary Material**.

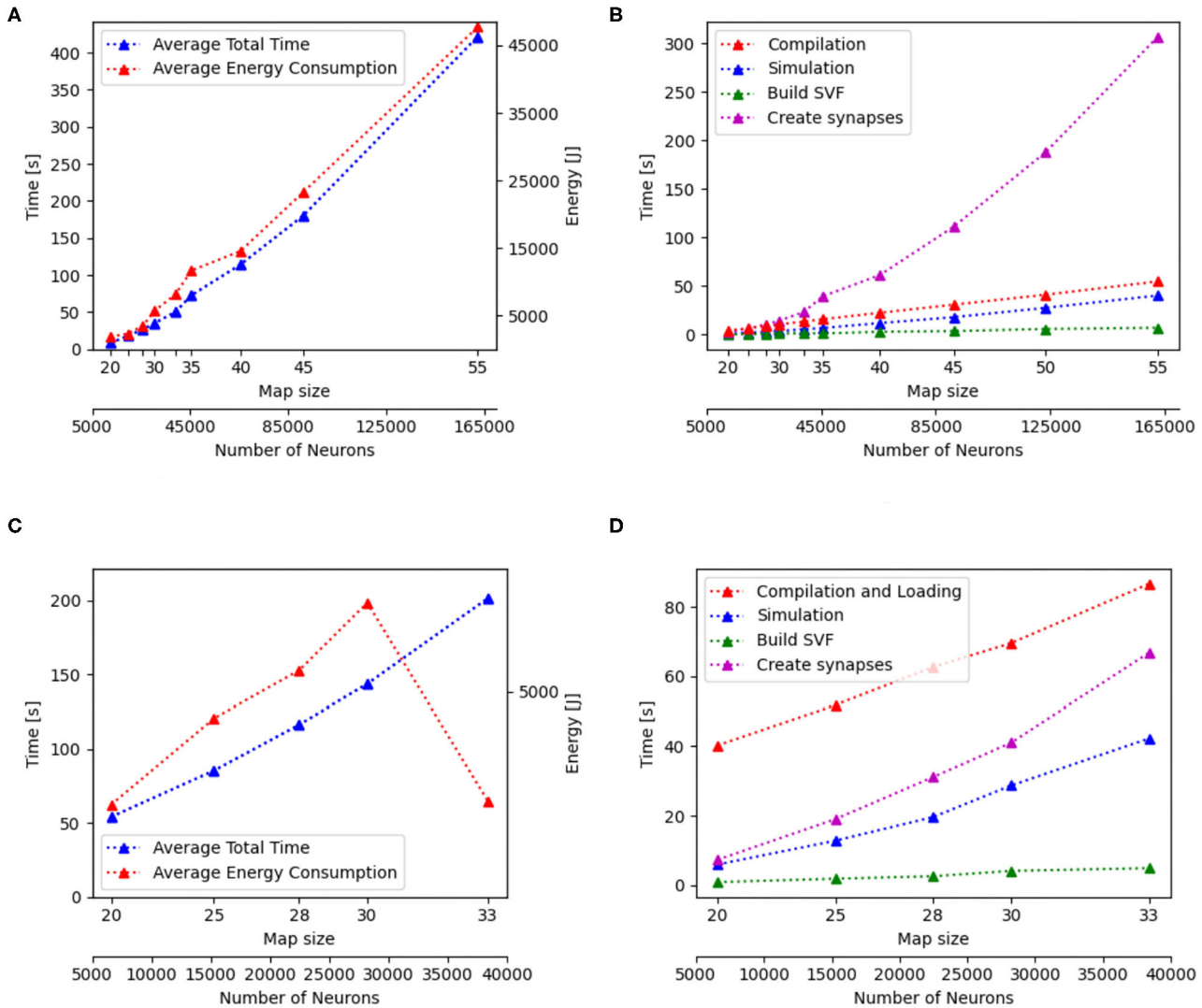


FIGURE 3 | Scaling properties of functions for GeNN on the CPU in **(A,B)** and for Jetson AGX Xavier in **(C,D)**. **(A,B)** Show how the total time and energy consumption scale with increasing map sizes. In **(C,D)**, the development of the different functions of the Wavefront algorithm is displayed with regard to an increasing map size. The x-axes show the map size and thereby, the number of neurons.

implementation or the GeNN implementations running on the desktop PC.

On all systems, the time it takes to create neurons is negligible compared to all other functions which is not surprising as the number of neurons is only a fraction of the number of synapses and weights. The function `Path finding` also makes up only a small proportion of the total time.

3.3. Path Length

The path length, stated in the third column of **Table 2**, is measured during the simulation, after the pathfinding process is finished. The path is saved as a list, which means the path's length equals the list's length. To compare the length of the paths found on the different implementations, the median of the path lengths is taken over all runs on a particular map. To check if the path length differs between individual runs, the standard deviation of the path lengths is considered as well. Since the Wavefront algorithm does not guarantee to find an optimal path (Steffen et al., 2020), the path for map size 20 has more detours than the path in the larger map. This is visualized for GeNN on PC in **Figure 7**.

The wavefront Algorithm relies on the weights of the synapses to create the SVF and find a path. In order to make the path length comparable, it needs to be ensured that the simulators start out with the same initial weight. Unfortunately this can only be ensured for the simulations in NEST and GeNN, as the bit shifting in the SpiNNaker system causes rounding errors, which are amplified by the scaling of the weights. As expected the

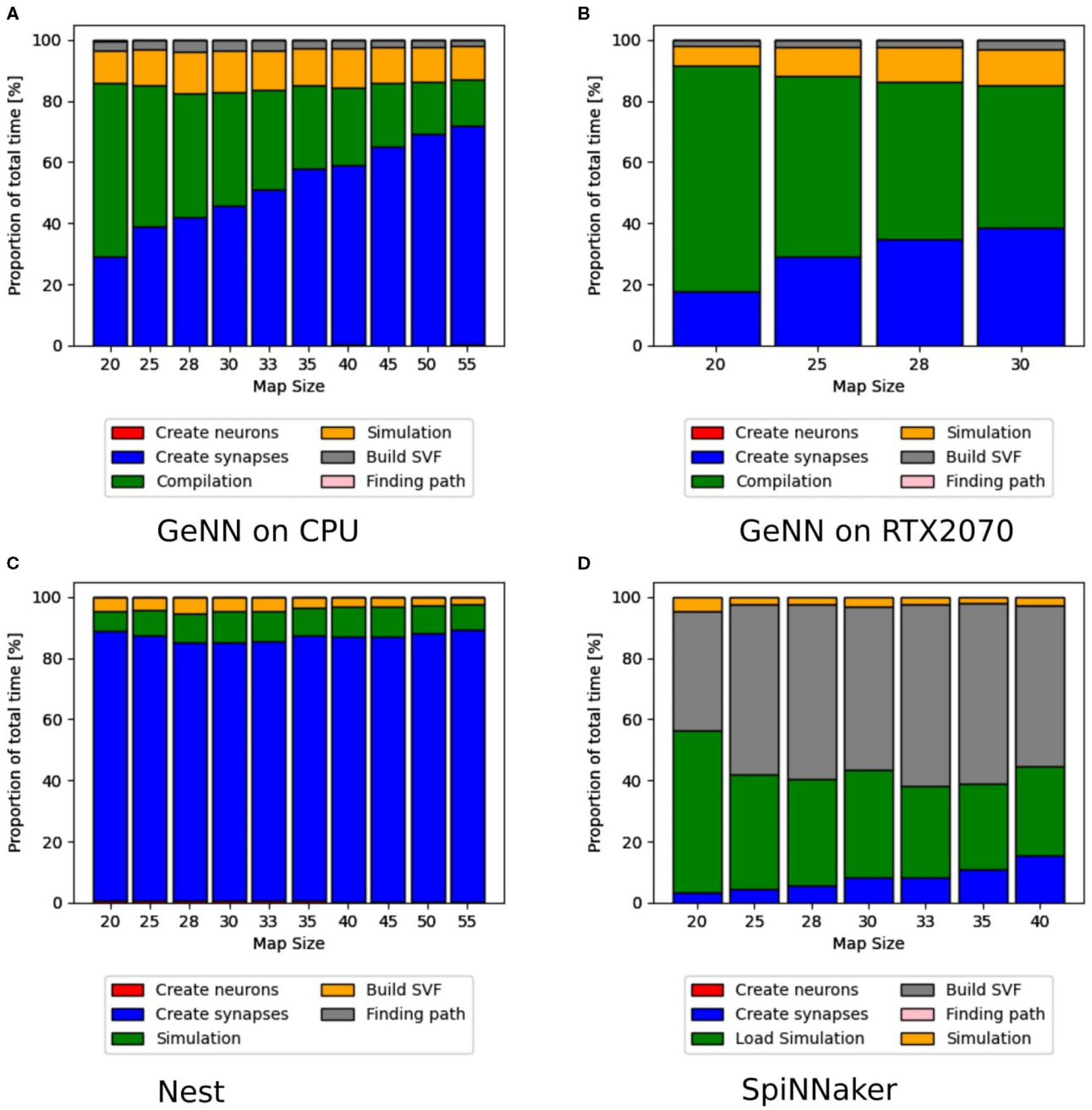


FIGURE 4 | Distribution of total time simulated with GeNN on a desktop PC in **(A,B)** and NEST in **(C)** and SpiNNaker in **(D)**. The different proportions of the sub-functions in percent are shown as bars for every map size. For **(A,B)**, the compilation time, shown as a green bar, make up the highest proportion of the total time. With increasing map sizes, the proportion of synapse creation increases. In **(B)** (RTX2070), the proportion of the actual simulation also increases steadily, whereas it remains more constant in the CPU implementation in **(A)**. In **(C)**, the NEST implementation, the synapse creation has by far the highest proportion of the total time. For the SpiNNaker implementation in **(D)**, the functions to load the simulation and build the SVF form the highest proportion of the total time.

path lengths differ between the SpiNNaker implementation and the other implementations. However, there is also a difference between the NEST and GeNN implementations. The most striking observation is that the path lengths differ on different GeNN implementations. Path lengths on the Jetson boards even differ between individual runs on the same map and same Jetson board. To rule out that the different GeNN implementations start out with different parameters, the initial synaptic weights are compared. The initial weights are measured after the code is compiled and loaded onto the GPUs. This ensures that

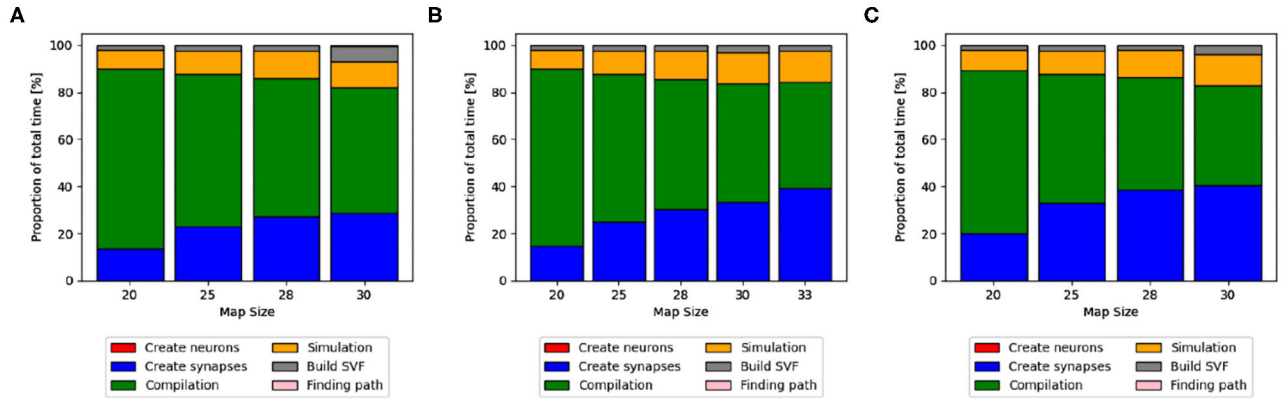


FIGURE 5 | Distribution of total time on Jetson boards. The different proportions of the sub-functions in percent are shown as bars for every map size. Compilation and loading time, shown as a green bar, make up the highest proportion of the total time. With increasing map sizes, the proportion of synapse creation (blue bar) and simulation (orange) increases. On the Xavier Nx a sharp increase in the proportion of the time to build the SVF (gray) can be observed for map size 30. **(A)** Xavier Nx, **(B)** AGX Xavier, **(C)** Tx2.

values are not altered during compilation or initialization on the GPU. All GeNN implementations share the same initial weights. To further narrow down the cause of the different path lengths, the final weights are analyzed as well. As expected, the final weights differ between the Jetson boards and the GeNN implementations on the PC. The final weights also differ between runs on the maps on all three Jetson boards. The differences in the path length, hence arise during the simulation of the Wavefront algorithm.

3.4. Energy Measurement

The data about power consumption, as provided in the fourth column of **Table 2**, is stored with a timestamp associated with every data point. To get the relevant data for each map, the power measurement data is matched with the timing data, by comparing their timestamps. All power draw data with timestamps between `timestamp_start` of the first run and `timestamp_end` of the last run are considered. As the power meter only logs power draw once a minute, very few data points are available for every map. The data is linearly interpolated and integrated to obtain the total energy. The total energy obtained is then divided by the number of simulation runs to get an average value for the energy consumption of a single run.

The energy consumption depends on the total time and the energy efficiency of the applied hardware. **Figures 3A,C**, **6A,C** show how the total time and energy consumption per run increase with an up scaled map size for GeNN on the CPU, GeNN on an AGX Xavier, the SpiNN-5 board and in NEST. It can be observed that the average energy consumption is increasing very similarly to the total time. The AGX Xavier and Xavier NX both require less energy per run than the GeNN implementations on the desktop PC. This is despite the fact that both Jetson boards have a much longer total time. The GeNN implementation on CPU consumes less energy than the implementation on the RTX 2070 which correlates with the shorter total simulation times.

3.5. Hardware Resources

The usage of hardware resources shows similar development for the different simulators on the different map sizes, therefore resource use is discussed by using exemplary data. Additional data is provided as **Supplementary Material**. The development of memory usage is similar for all GeNN simulations. **Figure 8** shows memory utilization in percent for the Jetson AGX Xavier which is similar to the other GeNN simulations and SpiNNaker for map size 33. For the Jetson AGX Xavier there is a very slow steady increase in memory usage, until the simulation is compiled and loaded which causes a sharp rise in memory utilization and is followed by a slower steady increase during the simulation itself. For SpiNNaker the memory usage rises much earlier, during the creation of the synapses and then rises again sharply when the simulation is loaded. The development of the CPU usage on the Desktop PC differs between the different simulators. **Figure 9**, shows the CPU utilization in percent for GeNN on CPU, GeNN on the RTX2070 and for SpiNNaker. For GeNN on the RTX270 a large single spike can be observed that takes place during compilation and loading of the simulation. For the SingleThreaded CPU backend of GeNN a large spike and a second smaller spike can be observed which coincide with compilation ad loading of the simulation and the simulation itself, respectively. Curiously, a larger proportion of the CPU is used during the compilation and loading process of the CUDA backend. For SpiNNaker one can observe a large spike in CPU usage during the creation of synapses and during the function `Build SVF`.

4. DISCUSSION

The fact that there is no significant difference in path lengths between the different implementations shows that the mechanisms to simulate STDP produce similar and consistent results. The Jetson boards form an exception in this regard as they have different path lengths in different runs of the simulation

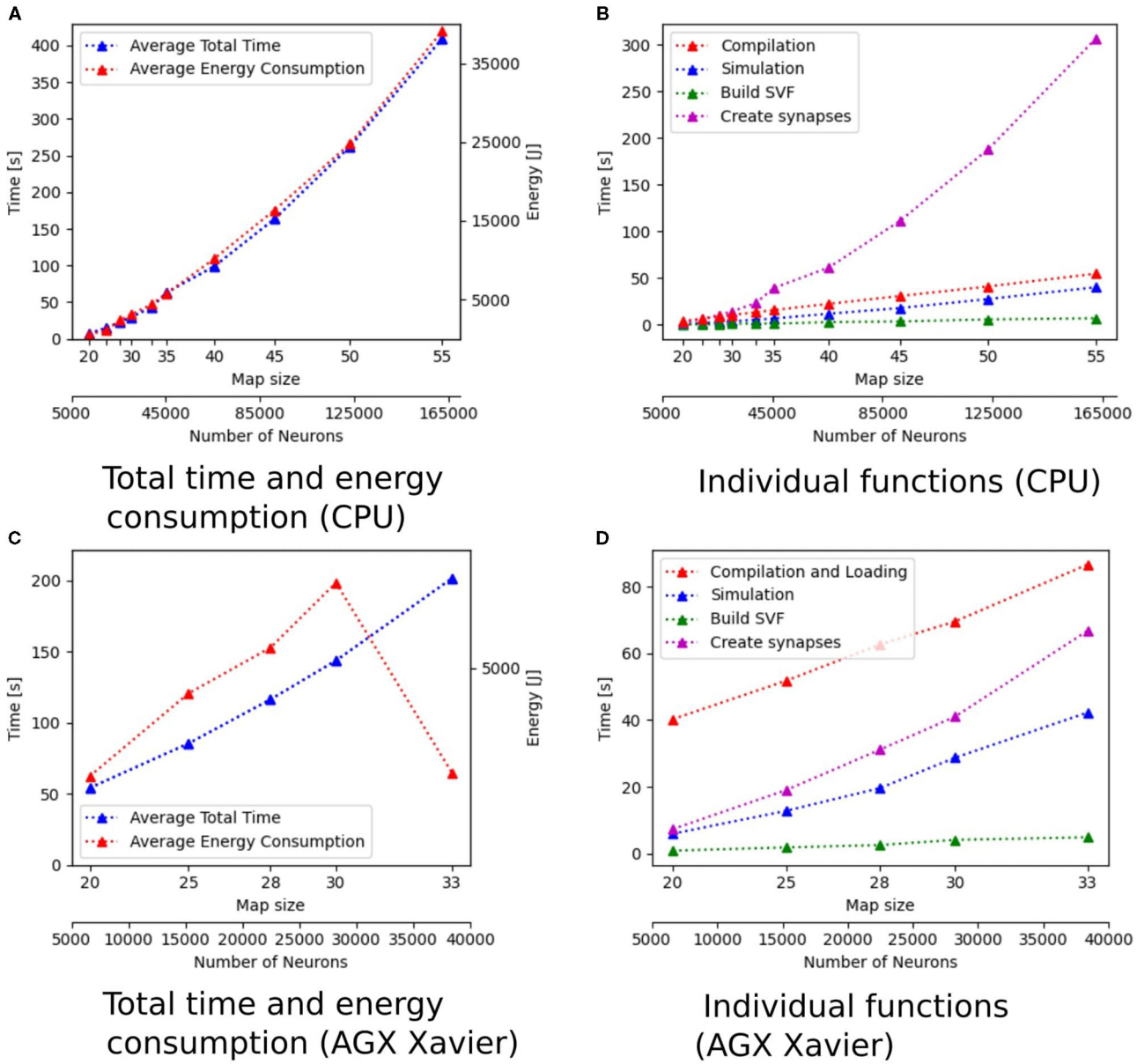


FIGURE 6 | Scaling properties of functions for SpiNNaker in **(A,B)** and NEST in **(C,D)**. **(A,C)** Show how the total time and energy consumption scale with increasing map sizes. In **(B,D)**, the development of the different functions of the Wavefront algorithm is shown with regard to an increasing map size. The x-axes show the map size and the number of neurons.

with the same map size. This difference is most likely caused by the non-associative nature of floating point numbers, which can lead to different results when additions are parallelized in GPUs. This makes it difficult to compare results obtained from calculations on a GPU to results obtained from calculations on CPUs (Whitehead, 2011). The possibility of simulation results differing between runs of a simulation, especially when STDP is used, is described in Yavuz et al. (2016). In Knight and Nowotny (2018), no noticeable divergence between simulations of the microcircuit mode were reported. In our benchmark only a single spike wave is simulated, making the model more susceptible to small differences in the simulation. However, it is surprising that the result deviations only appear on the embedded systems and not on the discrete GPU. Particularly as both are used with the single precision floating point operations, which makes the deviations in the simulation results more likely than double precision operations.

The Jetson boards, representing edge computing, stand out with regard to energy efficiency. As they are designed with a use for mobile applications in mind, they are optimized for low

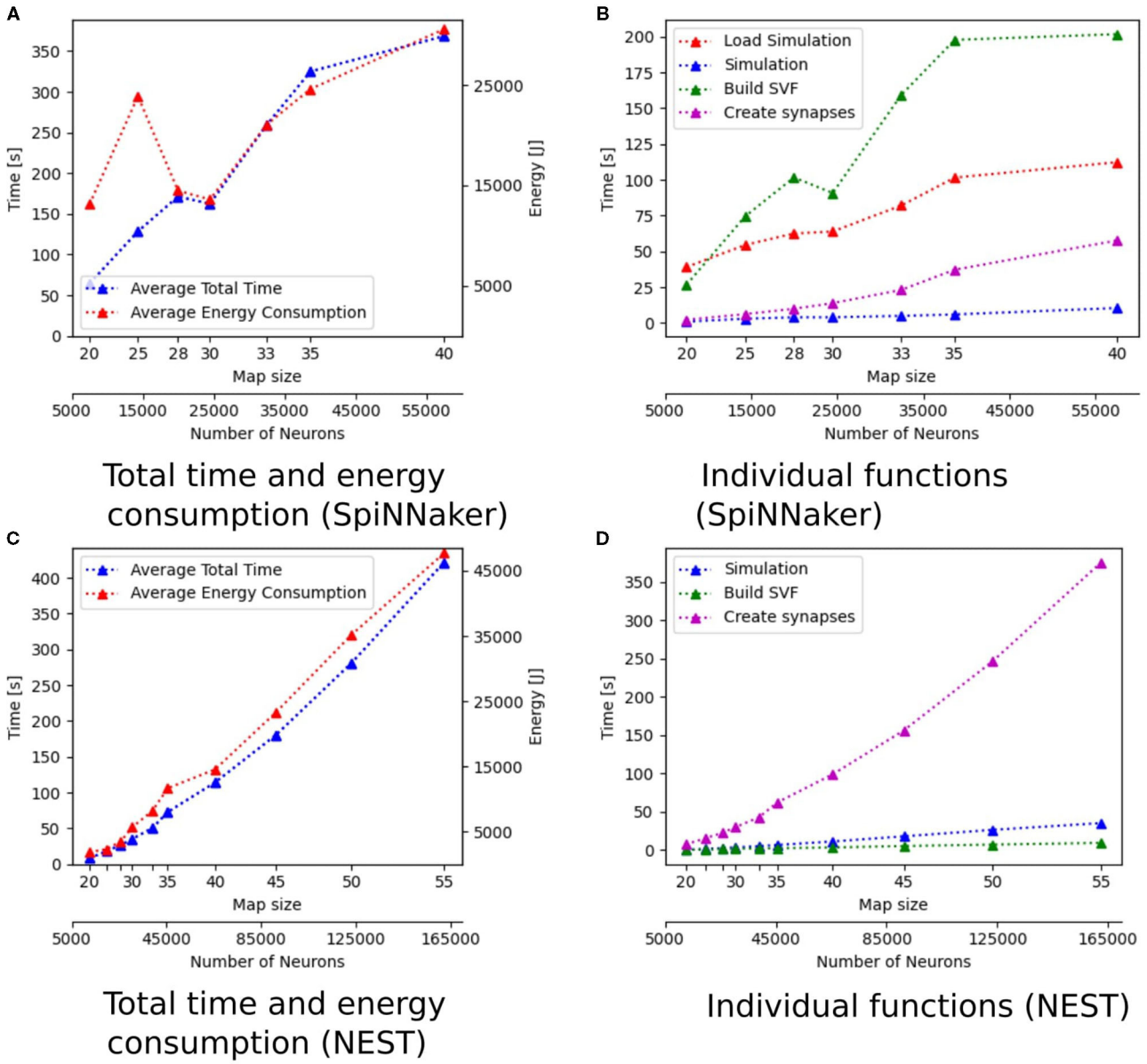


FIGURE 7 | Path found for size 20 and 25 with GeNN on CPU. The start neuron is highlighted in green, the target neuron cyan, and the path orange. For map size 25 a more direct path is found, which causes the path length to be slightly shorter, despite the larger map size. **(A)** Path for map size 20. **(B)** Path for map size 25.

power draw. A desktop PC on the other hand is not optimized in this regard. This explains the high energy consumption per run on the implementations on the desktop PC, despite the much shorter total simulation times. The NEST and GeNN implementation running only on the CPU of the desktop PC need less power than the GeNN implementation on the RTX2070 as their simulation times are shorter and the GPU is an additional device requiring energy. The high energy consumption of the SpiNNaker implementation shows that hardware with a higher power draw, that at the same time is able to run the simulation faster can still be more efficient than systems that have a low power draw but longer simulation times.

4.1. Contextual Analysis

For all implementations of GeNN, compiling/loading of the simulation and the creation of synapses are both tasks which are performed by the CPU. Therefore, for the majority of the time, only the CPU performs operations. Only the actual simulation of the SNN is carried out by the GPU. As the Jetson boards are designed for GPU heavy tasks, their CPUs are rather lightweight. The CPU complexes of all Jetson boards is made up of ARM based CPUs. For the Xavier Nx the CPUs can reach a maximum frequency of 1.9 GHz. A maximum frequency of 2.26 GHz can be reached by the CPUs of the AGX Xavier, while the CPUs of the Tx2 reach a maximum frequency of 2.0 GHz.

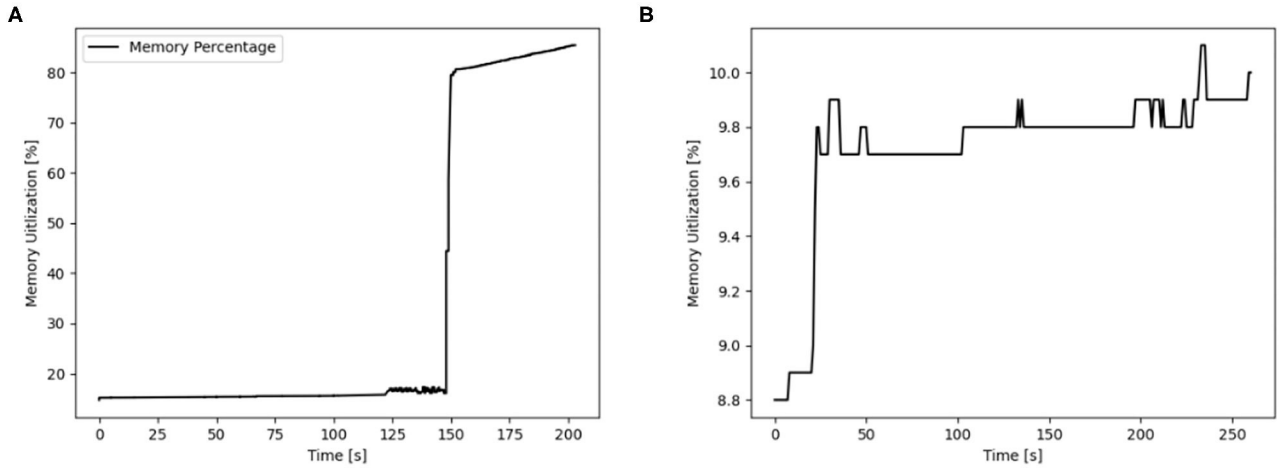


FIGURE 8 | Memory utilization in percent for GENN (exemplary AGX Xavier) and neuromorphic hardware for the map size 33. **(A)** AGX Xavier. **(B)** SpiNNaker.

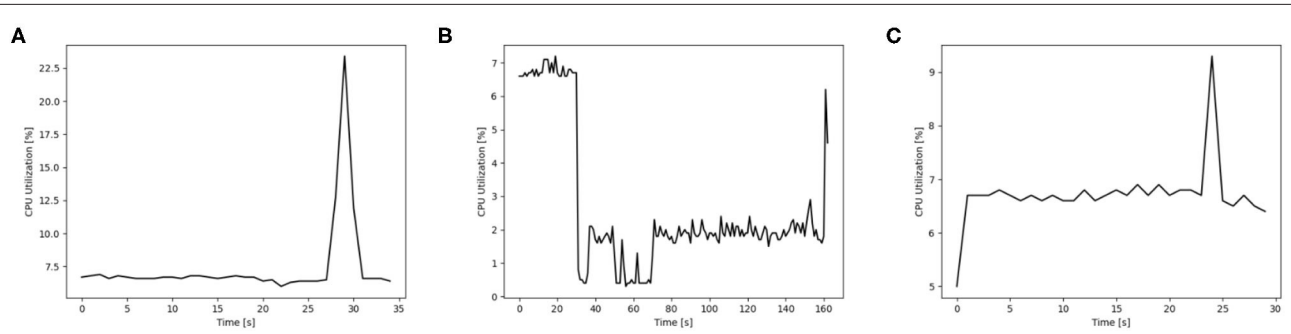


FIGURE 9 | CPU utilization in percent for different simulation options of GeNN on a Desktop PC and additionally on neuromorphic hardware for the map size 30. **(A)** GENN (GPU), **(B)** SpiNNaker, **(C)** GENN (CPU).

These maximum frequencies are low, however, consumer CPUs are generally more powerful than embedded processors. This explains the large difference in compilation times between the implementations on the Jetson boards and the simulations of GeNN on PC. The difference in compilation time between pure CPU implementation and the one using the GPU can be explained by the fact that no transfer between host RAM and GPU memory needs to take place when only a CPU is used. Also, no additional CUDA code needs to be generated and compiled in the CPU only version of GeNN.

The implementation in NEST does not need to be compiled or loaded to an external device. However, it takes longer to create neurons and synapses than on the GeNN CPU implementation. As shown in **Figure 1C**, the amount of synapses drastically increases with increasing map sizes which also leads to an increase in the time needed to create them. The creation of synapses is performed on the CPU using python for all implementations. For GeNN implementations, this entails the instantiation of PyNN projections. However, the synapses need to be instantiated again later when the C++/CUDA code is run. The compilation and simulation portions of GeNN do not display any exponential increase when maps are scaled. This suggests that GeNN, when used with its native frontend, can cope significantly better with large numbers of synapses than when it is used with its `pynn_genn` frontend. This comes as no surprise as C++ is a compiler language while python is an interpreter language which tends to be slower than compiler languages. When comparing the time required to create neurons and synapses in the GeNN and NEST implementations one also needs to account the compilation time in GeNN, but even when doing so, neurons and synapses are instantiated faster on the GeNN implementation. This advantage becomes especially apparent for large map sizes where it takes more than 20 s longer for the NEST implementation to create the synapses then for the GeNN CPU implementation. This difference in the time it takes to construct the network causes the overall shorter total time of the GeNN implementation on CPU, as NEST has shorter times for all map sizes when comparing only the simulation of the network.

For the implementation on the SpiNN-5 board, a similar amount of time is spent on synapse creation as in the GeNN CPU implementation. However, as the functions `Simulation`, `Load Simulation`, and `Build SVF` take much longer, it is less noticeable. As described in Rowley et al. (2020), both

simulation and loading require communication with the host system. This additional overhead explains the large difference in time required for simulation between the SpiNNaker system and the GeNN implementations. Especially the time it takes to build the SVF for the SpiNNaker stands out, as the SVF is built by the CPU of the desktop PC just like the implementations of GeNN on the desktop PC and NEST, which are all considerably faster. This can be explained by the fact that to build the SVF, the weights of all inhibitory- and excitatory connections are required, which need to be extracted from the simulation. For the implementation of GeNN on CPU and NEST, the weights are already present on the system RAM, hence requiring short loading times. These loading times get a bit bigger for GeNN on the RTX2070, where they need to be loaded from the device memory of the GPU. In the SpiNNaker implementation, the weights need to be transferred between the SpiNNaker device memory and the system RAM via a 100 Mbit Ethernet cable which is about 10 times slower than on the internal data busses used in the GeNN and NEST implementations.

4.2. Limitations and Outlook

The hardware performance comparison between high-performance GPU (RTX2070) and embedded GPU (Nvidia Jetson) can be seen as inappropriate. As one can foresee that the implementation running for desktop GPU are much faster while consuming more energy. However, it is still very interesting to see the exact delta of embedded GPU and discrete GPU/CPU. It provides a good insight about how far the state of development with embedded systems really is.

Memory is a limiting factor in this benchmark, as it limits the size of the SNN that can be simulated. Especially for the GeNN implementations that use the CUDA backend this poses a problem. Memory on GPUs tends to be less than the RAM on a PC. Regarding the Jetson boards, memory is shared between CPU and GPU which further limits their capabilities to simulate large SNNs, as there is an increasing performance penalty once the memory reaches its limits. The scripts that log the hardware data only allow for a sampling rate of 1 Hz as logging data more often than once per second results in uneven sampling intervals. Due to the low sampling rates of the energy logger, an in-depth analysis of the energy consumption and power draw is not possible. The data, however still shows trends and gives an order of magnitude of the energy consumption of the different hardware platforms. The implementation of models in `pynn_genn` introduces a large overhead on the GeNN implementations, as the synapses and neurons first need to be instantiated as pynn projections and populations before they can be simulated in GeNN which is implemented in C++.

As neuromorphic platforms with static synapses, such as True North or NeuroGrid, do not support neural plasticity, this benchmark using STDP is not well applicable for them. Platforms with configurable plasticity are generally capable of on-line learning. However, ROLLS does not support the required learning rule and the ODIN and TITAN chip have a quite limited number of neurons, only allowing the investigation of tiny maps. Platforms with programmable plasticity like BrainScaleS 1 & 2, SpiNNaker-2, and Loihi not only support online learning but also the required learning rule STDP. Therefore, our benchmark implementation is easily transferable to them. Hence, we want to transfer the implementation to either Loihi, SpiNNaker-2, or BrainScaleS 2.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

LS, AR, and RD are responsible for the idea, the core concept, and the architecture of this paper. LS, RK, SU, and SN did the research and wrote the paper. All authors contributed to the article and approved the submitted version.

FUNDING

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 945539 (Human Brain Project SGA3) as well as the Baden-Württemberg Stiftung under the research program Neurorobotik.

ACKNOWLEDGMENTS

We would like to acknowledge the effort of our partners within the HBP who took the time to give helpful advice and feedback. Particularly Thomas Nowotny, James Knight, and Sacha van Albada.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.667011/full#supplementary-material

REFERENCES

Blundell, I., Brette, R., Cleland, T. A., Close, T. G., Coca, D., Davison, A. P., et al. (2018). Code generation in computational neuroscience: a review of tools and techniques. *Front. Neuroinformatics* 12:68. doi: 10.3389/fninf.2018.00068

Bower, J. M., Beeman, D., Bower, J. M., and Beeman, D. (1998). "Introduction," in *The Book of GENESIS* (New York, NY: Springer), 3–5. doi: 10.1007/978-1-4612-1634-6_1

Carnevale, N. T., and Hines, M. L. (2006). *The NEURON Book | NEURON*. Cambridge University Press. doi: 10.1017/CBO9780511541612

Davies, M. (2019). Benchmarks for progress in neuromorphic computing. *Nat. Mach. Intell.* 1, 386–388. doi: 10.1038/s42256-019-0097-1

Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

DeWolf, T., Jaworski, P., and Eliasmith, C. (2020). Nengo and low-power AI hardware for robust, embedded neurorobotics. *Front. Neurorobot*. 14:568359. doi: 10.3389/fnbot.2020.568359

Diamond, A., Nowotny, T., and Schmuker, M. (2016). Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms. *Front. Neurosci*. 9:491. doi: 10.3389/fnins.2015.00491

Franklin, D. (2018). *NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics*. NVIDIA Developer Blog, NVIDIA Corporation.

Friedmann, S., Schemmel, J., Gruebl, A., Hartel, A., Hock, M., and Meier, K. (2016). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Trans. Biomed. Circuits Syst*. 11, 128–142. doi: 10.1109/TBCAS.2016.2579164

Furber, S., Galluppi, F., Temple, S., and Plana, L. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gewaltig, M.-O., and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Gütig, R., Aharonov, R., Rotter, S., and Sompolinsky, H. (2003). Learning input correlations through nonlinear temporally asymmetric Hebbian plasticity. *J. Neurosci*. 23, 3697–3714. doi: 10.1523/JNEUROSCI.23-09-03697.2003

Kim, D., Kung, J., Chai, S., Yalamanchili, S., and Mukhopadhyay, S. (2016). Neurocube. *ACM SIGARCH Comput. Archit. News* 44, 380–392. doi: 10.1145/3007787.3001178

Knight, J. C., and Nowotny, T. (2018). GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Front. Neurosci*. 12:941. doi: 10.3389/fnins.2018.00941

Mayr, C., Hoeppner, S., and Furber, S. (2019). SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning. *Concurr. Syst. Eng. Ser*. 70, 277–280. Available online at: https://niceworkshop.org/wp-content/uploads/2018/05/2-27-SHoppner-SpiNNaker2.pdf

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Minkovich, K., Thibeault, C. M., O'Brien, M. J., Nogin, A., Cho, Y., and Srinivasa, N. (2014). HRLSim: A high performance spiking neural network simulator for GPGPU clusters. *IEEE Trans. Neural Netw. Learn. Syst*. 25, 316–331. doi: 10.1109/TNNLS.2013.2276056

Müller, E., Mauch, C., Spilger, P., Breitwieser, O. J., Klähn, J., Stöckel, D., et al. (2020a). Extending brainScaleS OS for brainScaleS-2. *arXiv [Preprint] arXiv:2003.13750*.

Müller, E., Schmitt, S., Mauch, C., Billaudelle, S., Grübl, A., Güttler, M., et al. (2020b). The operating system of the neuromorphic brainscales-1 system. *arXiv arXiv:2003.13749*.

Mutch (2010). *CNS: Cortical Network Simulator Programming Guide - Overview | The Center for Brains*. Minds and Machines.

Nooraliei, A., and Nooraliei, H. (2009). "Path planning using wave front's improvement methods," in *ICCTD 2009 - 2009 International Conference on Computer Technology and Development* (Kota Kinabalu), 259–264. doi: 10.1109/ICCTD.2009.202

Nowotny, T. (2010). "Parallel implementation of a spiking neuronal network model of unsupervised olfactory learning on NVidia®CUDA," in *Proceedings of the International Joint Conference on Neural Networks* (Barcelona). doi: 10.1109/IJCNN.2010.5596358

Ostrau, C., Homburg, J., Klarhorst, C., Thies, M., and Rückert, U. (2020). Benchmarking Deep Spiking Neural Networks on Neuromorphic Hardware. *arXiv:2004.01656*. doi: 10.1007/978-3-030-61616-8_49

Pal, A., Tiwari, R., and Shukla, A. (2011). A focused wave front algorithm for mobile robot path planning. *Lecture Notes Comput. Sci*. 6678(Pt 1), 190–197. doi: 10.1007/978-3-642-21219-2_25

Ponulak, F., and Hopfield, J. J. (2013). Rapid, parallel path planning by propagating wavefronts of spiking neural activity. *Front. Comput. Neurosci*. 7:98. doi: 10.3389/fncom.2013.00098

Rittner, P., and Cleland, T. A. (2014). Myriad: a transparently parallel GPU-based simulator for densely integrated biophysical models. *Society for Neuroscience (Abstract),* Washington, DC.

Rowley, A., Rhodes, O., Bogdan, P., Brenninkmeijer, C., Davidson, S., Fellows, D., et al. (2020). "Stacks of software stacks," in *SpiNNaker–A Spiking Neural Network Architecture*, eds S. Furber and P. Bogdan (Norwell, MA: Now Publishers), 79–128. doi: 10.1561/9781680836530.ch4

Sanzleon, P., Knock, S. A., Woodman, M. M., Domide, L., Mersmann, J., McIntosh, A. R., et al. (2013). The virtual brain: a simulator of primate brain network dynamics. *Front. Neuroinformatics* 7:10. doi: 10.3389/fninf.2013.00010

Schemmel, J., Grübl, A., Meier, K., and Mueller, E. (2006). "Implementing synaptic plasticity in a VLSI spiking neural network model," in *International Joint Conference on Neural Networks (IJCNN)* (IEEE), 1–6. doi: 10.1109/IJCNN.2006.246651

Schmuker, M., and Schneider, G. (2007). Processing and classification of chemical data inspired by insect olfaction. *Proc. Natl. Acad. Sci. U.S.A*. 104, 20285–20289. doi: 10.1073/pnas.0705683104

Steffen, L., Kübler da Silva, R., Ulbrich, S., Vasquez Tieck, J. C., Roennau, A., and Dillmann, R. (2020). Networks of place cells for representing 3D environments and path planning. *BioRob*. 8, pp. 1158–1165. doi: 10.1109/BioRob49111.2020.9224441

Stimberg, M., Goodman, D. F., and Brette, R. (2020). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Front. Neurosci*. 12:291. doi: 10.3389/fnins.2018.00291

Vineyard, C. M., Green, S., Severa, W. M., and Koç, Ç. K. (2019). "Benchmarking event-driven neuromorphic architectures," in *ACM International Conference Proceeding* (Knoxville, TN). doi: 10.1145/3354265.3354278

Whitehead, N. (2011). *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs*. Technical report, NVIDIA Corporation.

Yan, Y., Stewart, T. C., Choo, X., Vogginger, B., Partzsch, J., Höppner, S., et al. (2021). "Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control," in *Neuromorphic Computing and Engineering*. Available online at: http://iopscience.iop.org/article/10.1088/2634-4386/abf150

Yang, S., Wang, J., Hao, X., Li, H., Wei, X., Deng, B., et al. (2021a). BiCoSS: toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw. Learn. Syst*. doi: 10.1109/TNNLS.2020.3045492. [Epub ahead of print].

Yang, S., Wang, J., Zhang, N., Deng, B., Pang, Y., and Azghadi, M. R. (2021b). CerebelluMorphic: large-scale neuromorphic model and architecture for supervised motor learning. *IEEE Trans. Neural Netw. Learn. Syst*. doi: 10.1109/TNNLS.2021.3057070. [Epub ahead of print].

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: A code generation framework for accelerated brain simulations. *Sci. Rep*. 6, 1–14. doi: 10.1038/srep18854

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 05 July 2021
doi: 10.3389/fncom.2021.675741

Toward Software-Equivalent Accuracy on Transformer-Based Deep Neural Networks With Analog Memory Devices

Katie Spoon[1], Hsinyu Tsai[1]*, An Chen[1], Malte J. Rasch[2], Stefano Ambrogio[1], Charles Mackin[1], Andrea Fasoli[1], Alexander M. Friz[1], Pritish Narayanan[1], Milos Stanisavljevic[3] and Geoffrey W. Burr[1]*

[1] IBM Research–Almaden, San Jose, CA, United States, [2] IBM T. J. Watson Research Center, Yorktown Heights, NY, United States, [3] IBM Zurich Research Center, Zurich, Switzerland

Recent advances in deep learning have been driven by ever-increasing model sizes, with networks growing to millions or even billions of parameters. Such enormous models call for fast and energy-efficient hardware accelerators. We study the potential of Analog AI accelerators based on Non-Volatile Memory, in particular Phase Change Memory (PCM), for software-equivalent accurate inference of natural language processing applications. We demonstrate a path to software-equivalent accuracy for the GLUE benchmark on BERT (Bidirectional Encoder Representations from Transformers), by combining noise-aware training to combat inherent PCM drift and noise sources, together with reduced-precision digital attention-block computation down to INT6.

Keywords: analog accelerators, BERT, PCM, RRAM, in-memory computing, DNN, Transformer

OPEN ACCESS

Edited by:
Alexantrou Serb,
University of Southampton,
United Kingdom

Reviewed by:
Damien Querlioz,
Centre National de la Recherche Scientifique (CNRS), France
Matthew Marinella,
Sandia National Laboratories (SNL), United States
Daniele Ielmini,
Politecnico di Milano, Italy

****Correspondence:***
Katie Spoon
katherine.spoon@colorado.edu
Hsinyu Tsai
htsai@us.ibm.com

Received: *03 March 2021*
Accepted: *14 May 2021*
Published: *05 July 2021*

Citation:
Spoon K, Tsai H, Chen A, Rasch MJ, Ambrogio S, Mackin C, Fasoli A, Friz AM, Narayanan P, Stanisavljevic M and Burr GW (2021) Toward Software-Equivalent Accuracy on Transformer-Based Deep Neural Networks With Analog Memory Devices. Front. Comput. Neurosci. 15:675741. doi: 10.3389/fncom.2021.675741

1. INTRODUCTION

State-of-the-art Deep Neural Networks (DNNs) have now demonstrated unparalleled accuracy performance across a wide variety of fields, including image classification, speech recognition, machine translation, and text generation (LeCun et al., 2015). While current models are generally trained and run on general-purpose digital processors such as CPUs and GPUs, the rapid growth in both size and scope of these networks has fostered novel hardware architectures aiming to optimize speed and energy-efficiency, specifically targeting either neural network training or inference (Sze et al., 2017).

Among these, architectures based on Non-Volatile Memory (NVM) are increasingly gaining interest. Such technologies encode weight information in the conductance states of two-terminal devices — including Resistive RAM (RRAM) (Wong et al., 2012), using modulation of conductive filaments between electrodes, or Magnetic RAM (MRAM) (Matsukura et al., 2015), using ferromagnetic switching between parallel or antiparallel spin polarization. In particular, Phase-Change Memory (PCM) (Burr et al., 2016) is based on thermally-driven reversible transitions between amorphous and crystalline states of a chalcogenide layer, leading to low and high conductances, respectively (**Figure 1A**).

Analog accelerators leverage the massive parallelism of NVM-based crossbar arrays to perform computation at the location of data (Burr et al., 2017; Ambrogio et al., 2018; **Figure 1B**). This architecture can significantly mitigate the Von-Neumann bottleneck caused by communication between the processor and memory, and is particularly efficient for fully-connected neural network layers (Burr et al., 2015).

A recent development in DNN-based natural language processing (NLP) is the migration away from recurrence toward Transformer-based models such as BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018). BERT offers state-of-the-art performance over a wide range of Natural Language Processing (NLP) tasks. While the large fully-connected layers in these models are computationally expensive for both conventional hardware and custom digital accelerators, they are ideally suited for analog NVM-based hardware acceleration. However, NVM devices exhibit many conductance instabilities [conductance drift (Ambrogio et al., 2019), programming and read noise (Tsai et al., 2019), etc.], which can degrade accuracy, particularly as the time between programming and inference increases.

In this paper, after a brief overview of Transformer-based models including BERT, we use a device-aware simulation framework to develop and assess techniques that can increase the inference accuracy of BERT implemented using PCM devices. We show that these techniques allow these inherently fast and energy-efficient systems to also approach software-equivalent accuracy [as compared to the original BERT implementation (Devlin et al., 2018)], despite the significant noise and imperfections of current PCM devices. Since the high energy-efficiency of analog crossbar-arrays on the fully-connected layers will then expose the energy-inefficiency in digital computation of the attention blocks, we explore the impact of quantized attention-block computation. We show that the use of reduced precision down to INT6 can provide further energy optimization for Transformer-based models, applicable both to analog NVM-based as well as to other accelerator systems.

1.1. Transformer Architecture

The Transformer architecture (Vaswani et al., 2017) was a pivotal change-point in deep learning and is expected to remain a critical core as new models [BERT (Devlin et al., 2018), DistilBERT (Sanh et al., 2019), Albert (Lan et al., 2020), etc.] continue to build upon the underlying Transformer architecture. Here we describe how the Transformer architecture differs from recurrent DNNs, and how the basic building blocks of Transformers map to analog accelerators.

1.1.1. Why Transformer?

Recurrent neural networks (RNNs) have commonly been used for NLP tasks to account for the sequential nature of words and sentences (**Figure 2A**). The bottleneck of RNNs is their limited "memory" over very long sequences. Transformers (Vaswani et al., 2017) provide one solution by replacing recurrence with a self-attention mechanism. For any given word *w* in the sequence, an attention probability between 0 and 1 is computed between *w* and every other word in the sequence (**Figure 2B**), allowing the model to quantify the relative importance that each word has in predicting *w*.

1.1.2. BERT-Base Model Architecture

Building on the initial success of Transformers, BERT was developed to generate meaningful encodings of input sequences useful across a broad range of downstream tasks, such as classification, text generation, and machine translation, requiring only a few epochs of subsequent fine-tuning to prepare for the specific task. BERT consists of 12 layers of a large Transformer encoder (**Figure 3A**). In **Figure 3B**, detailing the main building blocks of each encoder layer, dark grey boxes represent trained weight-matrices (fully-connected layers) that can readily be mapped to analog crossbar arrays. The attention computations (**Figure 3C**) along with all activation functions (representing a small fraction of the total operations) are computed in digital processing units.

2. MATERIALS AND METHODS

2.1. Optimizing Analog Accuracy for BERT

In this section, we first describe the comprehensive analog tile model used in this paper to capture realistic PCM crossbar array behavior. We then describe our simulation procedure and datasets for evaluation before discussing inference accuracy results. The simulator is implemented using a modified pytorch framework (Paszke et al., 2019) (including Caffe2).

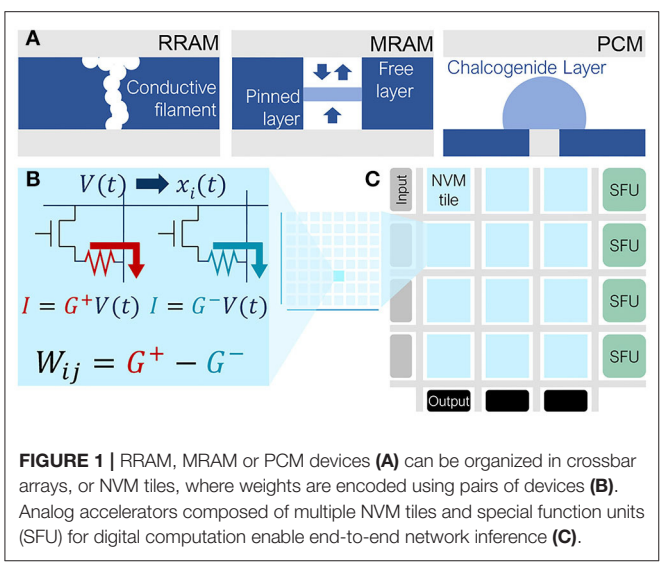


FIGURE 1 | RRAM, MRAM or PCM devices **(A)** can be organized in crossbar arrays, or NVM tiles, where weights are encoded using pairs of devices **(B)**. Analog accelerators composed of multiple NVM tiles and special function units (SFU) for digital computation enable end-to-end network inference **(C)**.

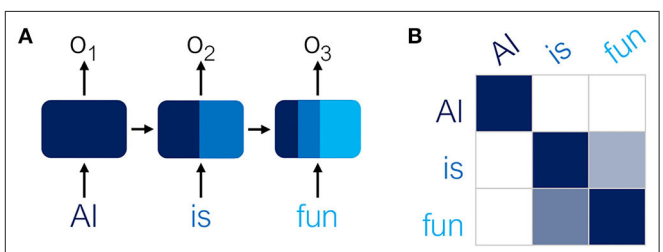


FIGURE 2 | (A) Recurrent Neural Networks (RNNs) use recurrence to maintain "memory" of the sequence. Hidden states of previous words contribute to the next state. **(B)** In contrast, Transformers compute an attention matrix, where higher (darker) probabilities indicate which words are interrelated.

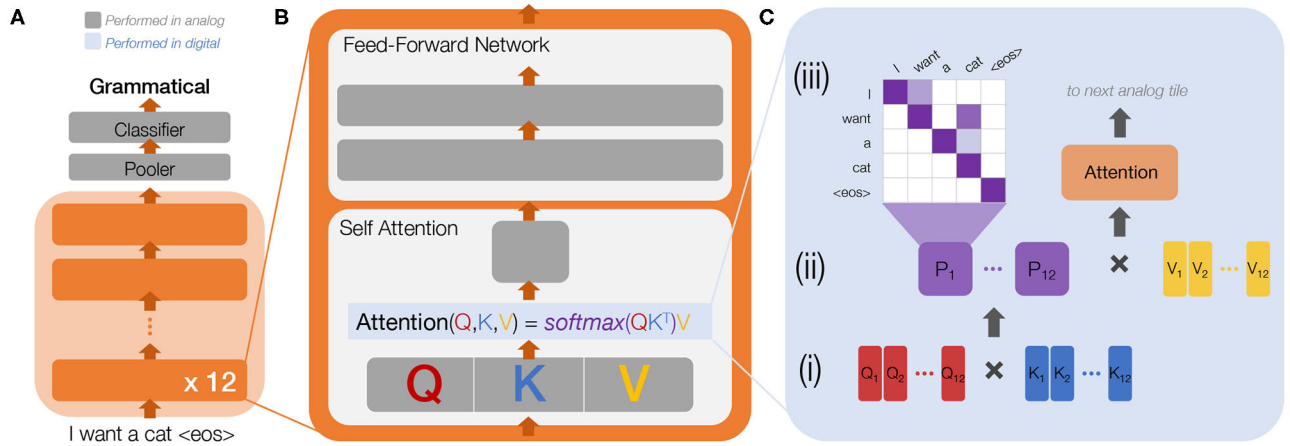


FIGURE 3 | (A) Bidirectional Encoder Representations from Transformers (BERT) with 12 encoder layers. The input to BERT is a sequence of tokens, where each token is either a word or a word-piece. This sequence is processed through each layer, followed by a pooler to reduce output size and a fully-connected classifier layer. For example, to classify "I want a cat <eos>" (where <eos> is the end-of-sentence token) as either grammatical (0) or not (1), the classifier needs only two outputs. Each encoder layer **(B)** is comprised of two main building blocks: (1) the **self-attention block**, where the model computes an attention matrix between the input and itself, and (2) a **feed-forward network** with two large fully-connected layers. Dark grey represents trained weight layers in analog, while **(C)** shows the attention processing in digital. The input sequence to the self-attention block passes through a trained weight layer split into three parts to compute ***Q* (query), *K* (key), and *V* (value) matrices**. To compute attention **(C)**, *Q*, *K*, and *V* are each split into multiple attention heads (for BERT, 12), both to reduce matrix sizes and to allow each to learn slightly different representations of the sequence. [c(i)] A similarity matrix is computed between *Q* and *K*, followed by a softmax operation along rows to produce values between 0 and 1. [c(ii)] These probabilities are then multiplied by *V* and move to the next analog tile followed by the feed-forward network. [c(iii)] A higher probability (darker shade) in one of the 12 probability (P) matrices might indicate, for example, that the word "cat" is important for prediction of the word "want".

2.1.1. Analog Tile Model

Weights, in this study, are encoded using a differential conductance pair G^+ and G^- without any redundancy scheme. Zero weights are encoded with $G^+ = G^- = 0$, therefore considering both devices at the RESET (lowest) conductance of the analog device. While, in practice, the minimum conductance cannot be zero, therefore the accuracy of the zero conductance could be limited, the large (100x–1,000x) PCM device on-off ratio ensures a fairly good approximation of a zero weight with very low RESET conductance and RESET noise.

Multiplication in the analog tile is performed by tuning the input voltage pulse-width, to prevent distortions due to conductance non-linearities as a function of read voltage (Chang et al., 2019). In order to accurately simulate the analog components in the analog accelerator system, we include various sources of non-ideality in the analog multiply-accumulate (MAC) operation, including quantization errors within the digital peripheral circuitry and conductance noise within the analog NVM devices. In this section, we describe the PCM-based device noise model and optimized design parameters we used to achieve near software-equivalent accuracy inference on BERT.

2.1.2. Programming Noise, Conductance Drift and 1/f Read Noise

The inference accuracy attainable in an analog accelerator system depends strongly on the analog device conductance properties, since these can be noisy and change over time. In order to estimate the accuracy characteristics of future analog accelerators, we model these effects by adding programming noise, read noise, and conductance drift to the DNN weights (**Figure 4A**). We aggregate model error over many simulation instances to arrive at the expected inference accuracy for a given time point. The noise model used here is based on the experimental characterization from Joshi et al. (2020), with PCM devices fabricated in a 90 nm technology. The associated open-source simulator (Rasch et al., 2021) includes the following PCM statistical model for inference:

- Programming noise represents the error incurred when encoding the weight in the PCM device. Instead of programming the correct target, the final achieved conductance generally shows some error, which is modeled based on the standard deviation of the iteratively programmed conductance values measured from hardware (Joshi et al., 2020):

$$g_{prog} = g_T + N(0, \sigma_{prog}) \quad (\mu S)$$

$$\sigma_{prog} = \gamma max(1:1731 g_T^2 + 1.965 g_T + 0.2635, 0) \quad (\mu S)$$

  where g_{prog} and g_T are the programmed and target conductances of a PCM device and $N(0, \sigma)$ is a normal distribution with standard deviation σ. The parameter γ is generally equal to 1, except when we explore the performances of devices with reduced noise, where $\gamma = 0.5$.
- PCM devices show a common trend for increasing time: after programming, due to the relaxation of the amorphous

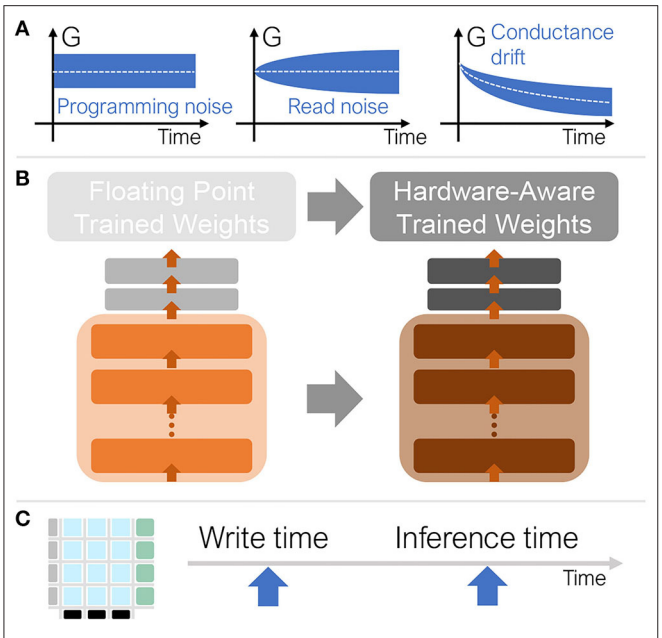


FIGURE 4 | (A) Conductance values (G) exhibit variability due to programming and read noise and decay toward zero over time due to drift noise. **(B)** To mediate these noise sources, we train the floating-point model with noise in order to prepare the model for noisy inference. **(C)** During inference, the weights are programmed, then some time passes before inference is performed.

state, conductance decays, following an empirical power-law function expressed as in Ielmini et al. (2007):

$$g_{drift}(t) = g_{prog}\left(\frac{t}{t_c}\right)^{-\nu} \quad (\mu S)$$

where g_{prog} is the programmed conductance measured at time t_c and $g_{drift}(t)$ is the conductance at time t, while ν represents the drift exponent, or slope on a log-G vs. log-t plot. In our simulations, ν is sampled from a normal distribution $N(\mu_\nu, \sigma_\nu)$. Both μ_ν and σ_ν, dimensionless, depend on the target conductance g_T and are modeled by fitting experimental data from Joshi et al. (2020), with the following expressions:

$$\mu_\nu = min(max(-0.0155log(g_T) + 0.0244, 0.049), 0.1)$$

$$\sigma_\nu = min(max(-0.0125log(g_T) - 0.0059, 0.008), 0.045)$$

- PCM non-idealities also include instabilities after the programming stage, such as read noise. Even in the absence of programming error or conductance drift, consecutive PCM reads lead to slightly different conductance evaluations (Ambrogio et al., 2019). Among the multiple causes generating read noise, $1/f$ noise and random telegraph noise show the strongest contributions, with increased noise on lower-frequency components. Such behavior leads to analog levels' intrinsic precision degradation for longer times. The overall contribution can be modeled using a normal distribution with time-dependent sigma (Joshi et al., 2020):

$$g(t) = g_{drift}(t) + N(0, \sigma_{nG}(t)) \quad (\mu S)$$

The standard deviation of the read noise σ_{nG} at time t is obtained by integrating the power spectral density over the measurement bandwidth:

$$\sigma_{nG}(t) = \gamma g_{drift}(t) Q_s \sqrt{log\left(\frac{t + t_{read}}{2t_{read}}\right)} \quad (\mu S)$$

where t_{read} = 250 ns is the duration of the read pulse. The parameter Q_s, dimensionless, measured from the PCM devices as a function of g_T is given by:

$$Q_s = min\left(\frac{0.0088}{g_T^{0.65}}, 0.2\right)$$

The noise model used in this work was calibrated using a large number of PCM devices to characterize the statistics of (1) the weight programming error (due to deviations between programmed and desired conductance values), (2) the accumulated 1/f read noise of their PCM devices, and the (3) conductance drift and (4) drift variability as a function of the programmed conductance value. Details of the device measurement and modeling methodologies are described in the supplementary information of reference (Joshi et al., 2020).

2.1.3. Analog MAC Design and Additional Non-Idealities

While weights are encoded using full precision, we include all noise sources, therefore reflecting the true analog nature of devices, we assume that each analog tile receives digital inputs at full precision, scales and quantizes to an integer representation, then converts to analog duration using digital to analog converters (DACs). The output of the analog tile is discretized using analog to digital converters (ADCs). Both DAC and ADC discretize the values in a fixed range symmetrically around zero. We assume 8 bit precision for DAC and 10 bit for ADC. The input scaling factor for the DAC is initialized using example data, learned during training to optimally match the input ranges, and kept static during inference. Target weight ranges are clipped to $-1.0, \ldots, 1.0$, where 1.0 corresponds to maximum target device conductance, $g_{\max}$, although programming noise can induce overshoot. The output ADC range is related to the ADC gain and a parameter that depends on the ADC design. Here we set it to $-10, \ldots, 10$, which means that 10 "fully on" input lines (each at 1.0) in conjunction with 10 weights at maximum (also 1.0) would saturate the ADC output. Even though the tiles have 512 rows, not all weights are at their maximum. In typical DNN models, most weights and activations have low values or are near zero. In addition, the random-walk nature of aggregation along the bitlines causes the signal to grow as the square-root of the number of rows, not linearly. The dynamic range of 10 for the ADC is a design parameter.

Each digital output from the ADC is individually scaled and offset, to map the conductances back to the high-precision digital

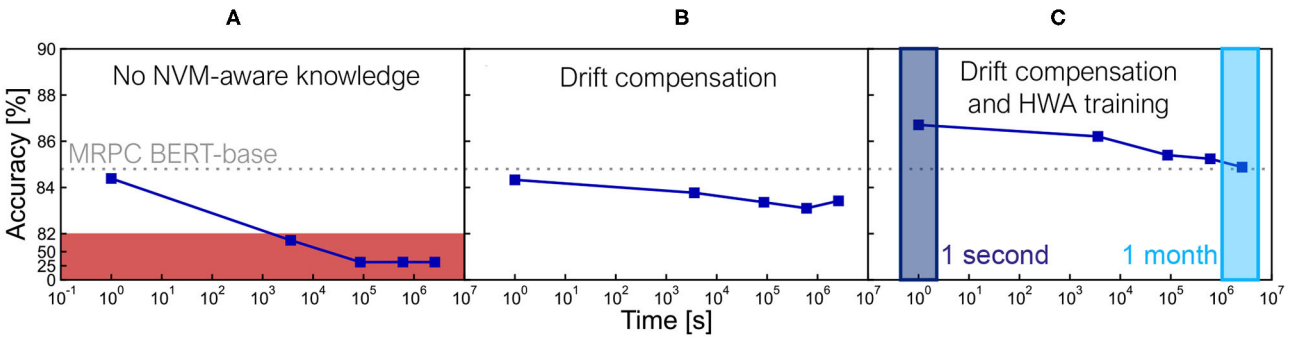


FIGURE 5 | (A) Without any noise-aware techniques, inference on the Microsoft Research Paraphrase Corpus (MRPC) task decays very quickly over time. **(B)** Drift compensation improves the decay over time significantly, but the inference results are still lower than the BERT-base ideal model with no NVM noise. **(C)** Hardware-aware (HWA) training with noise added during training helps close the gap, reaching software-equivalent accuracy for this task even at 1 month of drift.

domain (bfloat16 precision). These digital scaling factors are also learned during training and are critical to achieving software-equivalent accuracy during inference.

The analog MAC output is subject to short-term conductance-dependent noise that scales with the input current using the PCM read noise statistical model. We assume that the analog MAC output is subject to further additive Gaussian noise corresponding to 0.5 LSB (least significant bit) of the ADC, and use an approximated IR drop model. The analog tile size is set to 512×512 which, together with reduced read voltage (e.g., 0.2 V) ensures negligible IR drop impact; if layers are larger, they are distributed across multiple tiles and outputs are summed (in digital). Activation functions are computed in floating point 32-bit (FP32) format using standard functions.

2.2. Simulation Procedure–Training and Inference

Training for inference (i.e., hardware-aware training, or HWA) is done in software to make the subsequent hardware inference more robust, even in the presence of PCM non-idealities (**Figure 4B**). We apply noise during hardware-aware training, specifically during the forward propagation. While this helps the subsequent inference even in the presence of drift, this noise during training does not itself incorporate any explicit drift models. The subsequent backward propagation and weight update components or various scaling factors (described in previous sections) of software training are based on stochastic gradient descent (SGD) and are both carried out at full precision without additional noise.

Then, during inference, all hardware non-idealities—MAC cycle-to-cycle non-idealities, PCM programming noise, read noise, 1/f noise, drift, and drift variability—are considered, and drift compensation is applied as described below.

We train 5 models with different random seeds and select the best one for inference evaluation. Accuracy can sometimes exceed state of the art results for smaller datasets where run-to-run variation can be wider, while larger datasets show smaller accuracy variation. We re-evaluate each model 25 times for each inference time point[1] to reduce sampling error during inference. We also report the standard error in the tables of results (**Figures 6**, **8**). We evaluate accuracy at 5 time points after weight programming (**Figure 4C**): 1 second, 1 hour, 1 day, 1 week, and 1 month. Without any correction techniques, the inference accuracy drops markedly over time (**Figure 5A**).

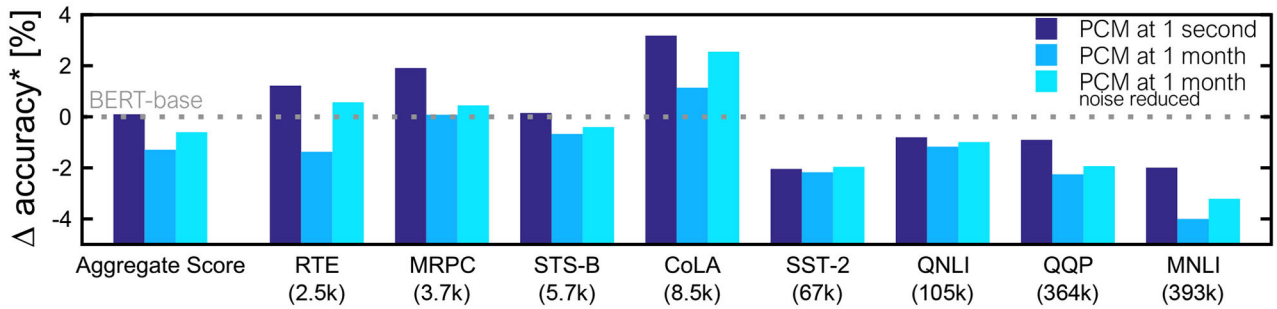
2.2.1. Drift Compensation

As described in Ambrogio et al. (2019) and Joshi et al. (2020) and illustrated in **Figure 5B**, signal loss by PCM conductance drift can be effectively compensated using a global correction-factor calculated from the mean drift over time. To calculate the drift compensation factor in the simulator, we first read out the weight matrix of each analog tile by performing the non-ideal MAC operations of the forward pass using one-hot input vectors, summing the values in an absolute manner to obtain an initial reference value. Then after applying conductance drift and accumulated 1/f noise to the weights up to a certain inference time-point, the weights are again read out through the same (non-ideal) MAC operations to produce a delayed reference value. Drift compensation is applied by adjusting the digital output scale-factor (applied after ADC) by the ratio of the delayed and initial reference values, and applied across the entire test set for all simulations of the model at that inference time-point. Once the average drift is compensated, the remaining noise effects act as a random walk process, as programmed conductances evolve away from their intended states. RRAM, FERAM, or any other device will also exhibit time-dependent conductance change, and these devices can also benefit from the methodology proposed in this work by substituting the corresponding device noise models.

2.2.2. Hardware-Aware (HWA) Training

Drift compensation helps with the accuracy decrease over time by boosting the signal, but cannot remove the underlying noise sources. In addition to training the static scale factors for DAC input and ADC output, we apply a variety of techniques to prepare our trained model for noise during inference (Gokmen

[1]For one particular task, Quora Question Pairs (QQP), we use only 5 repeats due to large test dataset size.



	Aggregate Score	RTE (2.5k)	MRPC (3.7k)	STS-B (5.7k)	CoLA (8.5k)	SST-2 (67k)	QNLI (105k)	QQP (364k)	MNLI (393k)
BERT-base [9]	80.7	66.4	84.8	85.8	52.1	93.5	90.5	89.2	83.4
1 second	80.80	67.62 ±0.32	86.71 ±0.17	85.95 ±0.05	55.28 ±0.21	91.46 ±0.07	89.70 ±0.03	88.30 ±0.03	81.41 ±0.07
1 hour	80.54	67.03 ±0.38	86.21 ±0.24	85.90 ±0.05	55.03 ±0.31	91.50 ±0.07	89.67 ±0.04	87.90 ±0.05	81.08 ±0.06
1 day	80.05	66.18 ±0.32	85.40 ±0.18	85.71 ±0.05	54.32 ±0.44	91.26 ±0.07	89.52 ±0.05	87.41 ±0.10	80.61 ±0.06
1 week	79.83	65.70 ±0.47	85.24 ±0.13	85.47 ±0.07	54.30 ±0.29	91.28 ±0.07	89.50 ±0.04	87.14 ±0.06	80.04 ±0.10
1 month	79.41	65.03 ±0.40	84.88 ±0.19	85.13 ±0.08	53.24 ±0.32	91.33 ±0.09	89.33 ±0.04	86.95 ±0.10	79.40 ±0.08
1 month noise reduced	80.10	66.97 ±0.34	85.25 ±0.20	85.40 ±0.05	54.65 ±0.33	91.54 ±0.07	89.51 ±0.04	87.27 ±0.08	80.19 ±0.07

* All datasets report accuracy, CoLA reports Matthew Correlation Coefficient, STS-B reports Pearson Correlation

FIGURE 6 | Inference results for all 8 GLUE tasks and the average score. Dataset training size shown in parentheses below each task name, and tasks appear in order of their size, with smallest on the left. Since each task has a different standard accuracy range, shown is the Δaccuracy between the results from the BERT-base model and our noise-aware trained model for two conditions: (i) full noise model applied, and (ii) 50% programming and read noise and full drift noise applied (noise reduced). For the full noise model, we consider several different time points, ranging from 1 month down to 1 day (with 1 hour and 1 second shown for context). The required time span would depend on the application. The table reports mean values across trials and standard errors of the mean.

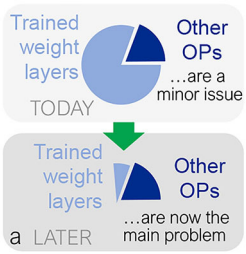
et al., 2019; Joshi et al., 2020). A noise model that includes digital periphery noise and additional noise on DNN weights that mimics a scaled version of our programming noise is applied during training, to prepare the network for inference with noisy weights. The standard deviation scale of this additional weight noise is a hyper-parameter of the HWA training. The effects can be seen in **Figure 5C**, reaching software-equivalent accuracy for a single language task only once these HWA training techniques are applied.

2.3. Datasets and Training

We evaluate our HWA-trained BERT on the General Language Understanding Evaluation (GLUE) Benchmark (Wang et al., 2019), consisting of 9 primary language tasks (see leaderboard at Wang et al., 2020). This benchmark is more robust than examining a single task, as it shows the network's ability to generalize. For example, one task tests the network's ability to identify a given sentence as grammatical or not. Another task assesses, given two sentences A and B, whether A is a paraphrase of B. We exclude one task, Winograd Natural Language Inference (WNLI), just as BERT (Devlin et al., 2018) did, due to the unusual construction of the data set and small test set of only 146 samples. This leaves 8 tasks:

- Microsoft Research Paraphrase Corpus (**MRPC**) (Dolan and Brockett, 2005)
- Recognizing Textual Entailment (**RTE**) (Bar-Haim et al., 2006; Dagan et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009)
- Semantic Textual Similarity Benchmark (**STS-B**) (Agirre et al., 2007)
- The Corpus of Linguistic Acceptability (**CoLA**) (Warstadt et al., 2018)
- The Stanford Sentiment Treebank (**SST-2**) (Socher et al., 2013)
- Question Natural Language Inference (**QNLI**) (Rajpurkar et al., 2016)
- Quora Question Pairs (**QQP**)
- Multi-Genre Natural Language Inference (**MNLI**) (Williams et al., 2018)

We evaluate each task separately by fine-tuning a pretrained BERT-base model (Wolf et al., 2020) using our HWA training techniques. We do not train BERT models from scratch using HWA training, but instead perform fine-tuning from the pretrained BERT model checkpoint with these techniques. Fine-tuning is a technique used in natural language processing, similar to transfer learning, where the main model is trained with a large amount of generic language data and later fine-tuned for a specific task (e.g., sentiment classification) using a much smaller set of data with limited epochs of training. This greatly reduces the runtime for the HWA training when compared to training from scratch. We use a maximum sequence length of 128 for efficiency, since the vast majority of data samples are much shorter that the maximum BERT sequence length of 512. We report the aggregated score of all 8 tasks, since this is a common metric reported for GLUE (Wang et al., 2019).



Circuit unit	Area [μm^2]	Energy [pJ]
IMA6 with ADD16	51.3	0.239
IMA8 with ADD20	103.3	0.521
IMA10 with ADD24	150.1	0.850
FP8FMA (1+4+3)	85.5	1.010
BFL16FMA (1+8+7)	240.6	2.697

b

FIGURE 7 | (A) While the energy-inefficiency in high-precision digital computation of the attention blocks may currently be a minor issue, the high energy-efficiency of analog crossbar-arrays on fully-connected layers will eventually expose this as a problem. **(B)** Particularly in Transformer-based models, quantizing the attention block to lower precision greatly reduces the area and energy usage of the multipliers, optimizing the new bottleneck: activation processing in attention. For example, decreasing from bfloat16 ("BFL16FMA") to INT6 ("IMA6") results in an estimated energy reduction of 91%.

Each task needs to be fine-tuned differently, so we scanned a variety of learning parameters for each task: batch size, learning rate, weight clipping, and dropout. Here we report the accuracy on the validation data set because the test set is only available online, which might result in a slight overestimation in the accuracy scores for the datasets with small validation set. We observe accuracy variation that correlates with the size of the datasets—models trained with smaller datasets exhibit larger variation in test accuracy. Therefore, we train 5 models per task per condition and choose the best model for inference simulation.

3. RESULTS

3.1. Results on BERT

Figure 5C shows an example of an HWA-trained BERT-base model reaching software-equivalent accuracy and the inference accuracy evolution over time for the MRPC task. Accuracy results on all 8 GLUE tasks, reported at times ranging from 1 second to 1 month after weight programming, are summarized in **Figure 6**. We show that several tasks reach software-equivalent accuracy at 1 month and the biggest accuracy drop is ∼4% for MNLI. The aggregate score over all 8 tasks is only 1.29% below the baseline at 1 month. Since there is hope for additional improvement with progress in PCM device technology (Giannopoulos et al., 2018), we show results for the full drift model but with only 50% of the programming and read noise applied during inference, achieved by setting the γ factor in the σ_{prog} and $\sigma_{nG}(t)$ equal to 0.5. In this way, we reduce the impacts of both programming and read noise contributions. Noise-reduced PCM devices can be expected to improve many of the tasks by >1% even for inference after 1 month, and increase the aggregate GLUE score to just 0.6% below baseline.

3.2. Attention Quantization

Attention-based models such as BERT pose unique challenges beyond previously studied models, because of the extensive activation computation in the self-attention block. Amdahl's law implies that when a system bottleneck is greatly improved, performance is invariably limited by something else, no matter how insignificant it was to begin with (**Figure 7A**). Self-attention computations in a Transformer model scale quadratically with sequence length S, and constitute <1% of the number of operations for small S, but ∼5% at $S = 128$. If this computation is done in digital processing units at full precision, the cost in both energy and area for such processing units can become the system bottleneck for Transformers, particularly as sequence length grows, despite constituting a relatively low fraction of the workload.

Reduction of the precision in the digital computation of this self-attention block can also help reduce overall computation costs, beyond consideration of the analog performance and precision of just the fully-connected layers. The attention matrix in this case is not mapped into analog crossbar arrays, but processed in digital multiply-and-add units.

3.2.1. Attention Computation

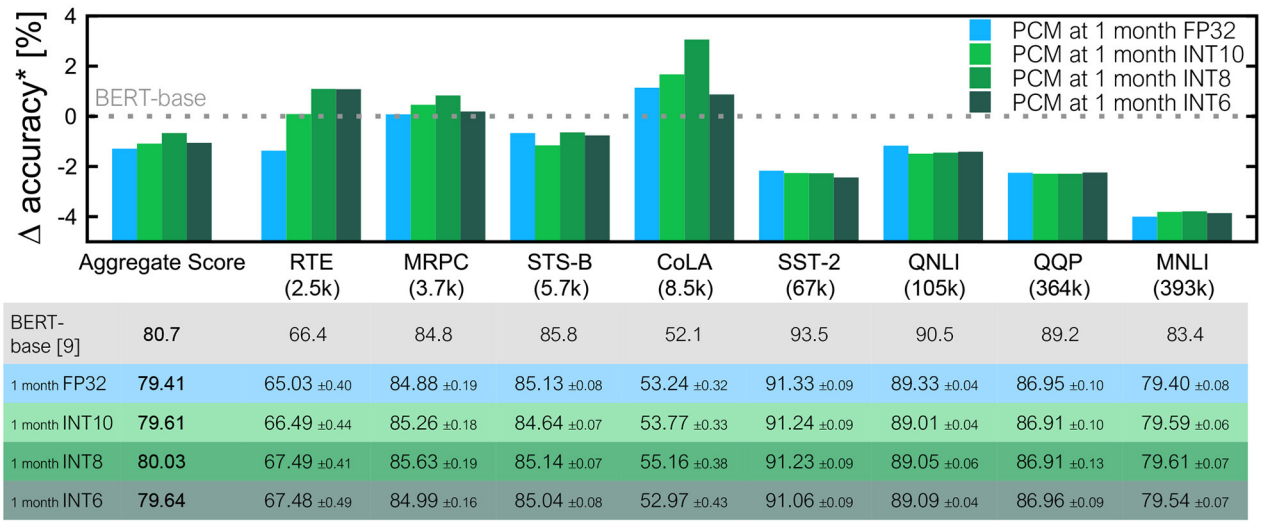
In the self-attention block, there are two batch matrix-multiplies, one for $Q*K$ and one for $softmax(Q*K)*V$ (**Figure 3C(i,ii)**). In this paper, we propose to compute batch matrix-multiplication with various integer precisions in order to reduce energy and area costs for these attention computation units, while keeping softmax operations at full precision. When compared to bfloat16 multiply-and-add (BFLFMA), integer multiply-and-add (IMA) units are much more energy and area efficient. **Figure 7B**, simulated in a 14 nm FinFET technology, shows a 11.3× energy benefit and a 4.7× area benefit from BFLFMA to INT6 (including a wide-enough adder for multiply-accumulate operations across the 64 terms in an attention-head). Next, we explore the impact of these attention quantization options on inference accuracy in BERT.

3.3. Results on BERT With Quantized Attention

Figure 8 summarizes GLUE task inference results with our analog tile models for the fully-connected layers with four different precision settings—FP32, integer 10 bit (INT10), integer 8 bit (INT8), and integer 6 bit (INT6)—for the batch matrix-multiplications in self-attention. The scaling factor used for quantization is initialized from a small set of training data and then learned during the training process. BERT inference performance is comparable among all four quantization schemes. For smaller datasets, INT10, INT8 and INT6 quantized attention models sometimes outperform the FP32 versions because of the additional regularization and noise in the attention layers during training. For the four larger datasets (SST-2, QNLI, QQP, and MNLI), no significant differences in inference accuracy at 1 month were observed down to INT6 quantized attention.

4. DISCUSSION

While we have clearly demonstrated the potential for iso-accuracy with Transformer-based neural networks on fast and energy-efficient analog hardware, there are numerous areas for future work.



BERT-base [9]	80.7	66.4	84.8	85.8	52.1	93.5	90.5	89.2	83.4
1 month FP32	79.41	65.03 ±0.40	84.88 ±0.19	85.13 ±0.08	53.24 ±0.32	91.33 ±0.09	89.33 ±0.04	86.95 ±0.10	79.40 ±0.08
1 month INT10	79.61	66.49 ±0.44	85.26 ±0.18	84.64 ±0.07	53.77 ±0.33	91.24 ±0.09	89.01 ±0.04	86.91 ±0.10	79.59 ±0.06
1 month INT8	80.03	67.49 ±0.41	85.63 ±0.19	85.14 ±0.07	55.16 ±0.38	91.23 ±0.09	89.05 ±0.06	86.91 ±0.13	79.61 ±0.07
1 month INT6	79.64	67.48 ±0.49	84.99 ±0.16	85.04 ±0.08	52.97 ±0.43	91.06 ±0.09	89.09 ±0.04	86.96 ±0.09	79.54 ±0.07

* All datasets report accuracy, CoLA reports Matthew Correlation Coefficient, STS-B reports Pearson Correlation

FIGURE 8 | Quantization inference results for all 8 GLUE tasks and the average score. Shown is a comparison to our FP32 noise-aware model from **Figure 6** at 1 month of drift for various levels of precision: INT10, INT8, and INT6, all of which perform at least as well as our full precision model for most tasks. On some tasks, including the aggregate score, the reduced precision seems to serve as additional regularization and performs better than our FP32 model. The table reports mean values across trials and standard errors of the mean.

4.1. Software-Equivalent accuracy

We have shown that full software-equivalent accuracy will require continued improvement in both PCM devices and in hardware-aware training techniques. However, we have been reasonably conservative in our accuracy report, presenting results at 1 month of inference. We note that some workloads may only require results at 1 day or 1 week of drift, for example when models are weekly updated. We project that current PCM devices can comfortably support software-equivalent accuracy on many GLUE tasks on such timescales. For tasks where models are less frequently updated, another approach would be to incur slightly more frequent in-place reprogramming of the same model – this would be a tradeoff between model availability, the time needed for model programming, device endurance, temperature variation and other factors.

4.2. Model Size

While we have focused on BERT, which has 110 M parameters, new Transformer-based networks are emerging that attempt to reduce model size while maintaining accuracy. DistilBERT (Sanh et al., 2019) uses knowledge distillation to reduce the number of parameters in half, and ALBERT (Lan et al., 2020) uses cross-layer parameter reuse, reducing the number of unique parameters to a fraction of the original. However, we note that these smaller models may present a challenge to analog hardware, since fewer unique weights can make models less robust to noise. Hardware-software co-optimization that can strike a good balance between model size and robustness to PCM-based noise could lead to future Transformer-based networks that are highly optimized for accuracy, energy-efficiency, and speed on Analog-AI hardware.

5. CONCLUSION

We show that despite their various noise sources, PCM-based analog accelerators are a sensible choice for deep learning workloads, even for large natural language processing models like BERT. Our simulation results using a comprehensive noise model demonstrate that BERT can be expected to be close to software-equivalent accuracy even with existing PCM devices. Other Transformer-based models with the same building blocks can be similarly evaluated with our approach. We have shown that expected improvements in programming noise variability provide a consistent trend toward software-equivalent accuracy. Finally, in preparation for high energy efficiency on the fully-connected layers, we provide a potential solution to the next biggest energy cost: the activation processing from the attention block. We show that 11.3× energy improvements should be feasible by quantization to INT6, with no significant loss in accuracy.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

KS, HT, MS, and GB conceived the original ideas. KS, HT, AC, and MS implemented and ran the simulations. All authors contributed during data analysis. KS, HT, AC, MR, SA, and GB drafted the manuscript.

REFERENCES

Agirre, E., Marquez, L., and Wicentowski, R., (2007). *Proceedings Fourth International Workshop on Semantic Evaluations (SemEval)*. (Prague).

Ambrogio, S., Gallot, M., Spoon, K., Tsai, H., Mackin, C., Wesson, M., et al. (2019). "Reducing the impact of phase-change memory conductance drift on the inference of large-scale hardware neural networks," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 6.1.1–6.1.4.

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., di Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* 558:60. doi: 10.1038/s41586-018-0180-5

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., et al. (2006). "The second PASCAL recognising textual entailment challenge," in *Proceedings Second PASCAL Challenges Workshop on Recognising Textual Entailment* (Venice).

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. (2009). "The fifth PASCAL recognizing textual entailment challenge," in *Proceedings Text Analysis Conference (TAC)* (Gaithersburg, MD).

Burr, G. W., Brightsky, M. J., Sebastian, A., Cheng, H.-Y., Wu, J.-Y., Kim, S., et al. (2016). Recent progress in PCM technology. *IEEE J. Emerg. Sel. Top. Circ. Sys.* 6, 146–162. doi: 10.1109/JETCAS.2016.2547718

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys. X* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Burr, G. W., Shelby, R. M., Sidler, S., di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large–scale neural network (165,000 synapses), using phase–change memory as the synaptic weight element. *IEEE Trans. Electron Dev.* 62, 3498–3507. doi: 10.1109/TED.2015.2439635

Chang, H. ., Narayanan, P., Lewis, S. C., Farinha, N. C. P., Hosokawa, K., Mackin, C., et al. (2019). Ai hardware acceleration with analog memory: microarchitectures for low energy at high speed. *IBM J. Res. Dev.* 63, 8:1–8:14. doi: 10.1147/JRD.2019.2934050

Dagan, I., Glickman, O., and Magnini, B. (2006). "The PASCAL recognising textual entailment challenge," in *ML Challenges: Evaluating Predictive Uncertainty, visual Object Classification, and Recognising Textual Entailment*, (Milan), 177–190.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dolan, W. B., and Brockett, C. (2005). "Automatically constructing a corpus of sentential paraphrases," in *Proceedings International Workshop on Paraphrasing*, (Jeju Island).

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. (2007). "The third PASCAL recognizing textual entailment challenge," in *Proceedings ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Prague, 1–9.

Giannopoulos, I., Sebastian, A., Le Gallo, M., Jonnalagadda, V., Sousa, M., Boon, M., et al. (2018). "8-bit precision in-memory multiplication with projected phase-change memory," in *2018 IEEE International Electron Devices Meeting (IEDM),* 27.7.1–27.7.4.

Gokmen, T., Rasch, M. J., and Haensch, W. (2019). "The marriage of training and inference for scaled deep learning analog hardware," in *2019 IEEE International Electron Devices Meeting (IEDM)* , 22–3.

Ielmini, D., Lacaita, A. L., and Mantegazza, D. (2007). Recovery and drift dynamics of resistance and threshold voltages in phase-change memories. *IEEE Trans. Electron Dev.* 54, 308–315. doi: 10.1109/TED.2006.888752

Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S. R., Piveteau, C., et al. (2020). Accurate deep neural network inference using computational phase-change memory. *Nat. Comm.* 11:2473. doi: 10.1038/s41467-020-16108-9

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). ALBERT: a lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

Matsukura, F., Tokura, Y., and Ohno, H. (2015). Control of magnetism by electric fields. *Nat. Nanotechnol.* 10, 209–220. doi: 10.1038/nnano.2015.22

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: an imperative style, high-performance deep learning library. *NIPS* 32, 8026–8037.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, TX, 2383–2392.

Rasch, M. J., Moreda, D., Gokmen, T., Gallo, M. L., Carta, F., Goldberg, C., et al. (2021). A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays. *arXiv*.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). "DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter," in *NeurIPS EMC^2 Workshop* (Vancouver, BC).

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., et al. (2013). "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, WA, 1631–1642.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2017). Efficient processing of deep neural networks: a tutorial and survey. *Proc. IEEE* 105, 2295–2329. doi: 10.1109/JPROC.2017.2761740

Tsai, H., Ambrogio, S., Mackin, C., Narayanan, P., Shelby, R. M., Rocki, K., et al. (2019). "Inference of long-short term memory networks at software-equivalent accuracy using 2.5M analog phase change memory devices," in *2019 Symposium on VLSI Technology*, T82–T83.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Neurips* (Long Beach, CA).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). "GLUE:a multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of ICLR* (New Orleans, LA).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2020). *GLUE Benchmark*. Available online at: gluebenchmark.com/leaderboard

Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural network acceptability judgments. *arXiv preprint 1805.12471*.

Williams, A., Nangia, N., and Bowman, S. (2018). "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, LO.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). "Transformers: State-of-the-art natural language processing," in *Proceedings Conference Empirical Methods in NLP: System Demonstrations*, 38–45.

Wong, H.-S. P., Lee, H.-Y., Yu, S., Chen, Y.-S., Wu, Y., Chen, P.-S., et al. (2012). Metal-Oxide RRAM. *Proc. IEEE* 100, 1951–1970. doi: 10.1109/JPROC.2012.2190369

Conflict of Interest: The authors were employed by IBM Research.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 20 July 2021
doi: 10.3389/fnins.2021.684113

Always-On Sub-Microwatt Spiking Neural Network Based on Spike-Driven Clock- and Power-Gating for an Ultra-Low-Power Intelligent Device

*Pavan Kumar Chundi[1], Dewei Wang[1], Sung Justin Kim[1], Minhao Yang[1], Joao Pedro Cerqueira[1], Joonsung Kang[2], Seungchul Jung[2], Sangjoon Kim[2] and Mingoo Seok[1]**

[1] Department of Electrical Engineering, Columbia University, New York City, NY, United States, [2] Samsung Electronics, Seoul, South Korea

OPEN ACCESS

Edited by:
Alexantrou Serb,
University of Southampton,
United Kingdom

Reviewed by:
Shuangming Yang,
Tianjin University, China
Dan Hammerstrom,
Portland State University,
United States
Vivek Mangal,
Apple, United States

***Correspondence:**
Mingoo Seok
ms4415@columbia.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: 22 March 2021
Accepted: 21 June 2021
Published: 20 July 2021

Citation:
Chundi PK, Wang D, Kim SJ, Yang M, Cerqueira JP, Kang J, Jung S, Kim S and Seok M (2021) Always-On Sub-Microwatt Spiking Neural Network Based on Spike-Driven Clock- and Power-Gating for an Ultra-Low-Power Intelligent Device. Front. Neurosci. 15:684113. doi: 10.3389/fnins.2021.684113

This paper presents a novel spiking neural network (SNN) classifier architecture for enabling always-on artificial intelligent (AI) functions, such as keyword spotting (KWS) and visual wake-up, in ultra-low-power internet-of-things (IoT) devices. Such always-on hardware tends to dominate the power efficiency of an IoT device and therefore it is paramount to minimize its power dissipation. A key observation is that the input signal to always-on hardware is typically sparse in time. This is a great opportunity that a SNN classifier can leverage because the switching activity and the power consumption of SNN hardware can scale with spike rate. To leverage this scalability, the proposed SNN classifier architecture employs event-driven architecture, especially fine-grained clock generation and gating and fine-grained power gating, to obtain very low static power dissipation. The prototype is fabricated in 65 nm CMOS and occupies an area of 1.99 mm^2. At 0.52 V supply voltage, it consumes 75 nW at no input activity and less than 300 nW at 100% input activity. It still maintains competitive inference accuracy for KWS and other always-on classification workloads. The prototype achieved a power consumption reduction of over three orders of magnitude compared to the state-of-the-art for SNN hardware and of about 2.3X compared to the state-of-the-art KWS hardware.

Keywords: always-on device, spiking neural network, event-driven architecture, neuromorphic hardware, clock and power gating

INTRODUCTION

An spiking neural network (SNN) classifier is an attractive option for ultra-low-power intelligent internet-of-things (IoT) devices. It is promising especially for always-on functions due to their spike-based operation for computation and communication, allowing their switching activity and power to scale smoothly with the input activity rate. An SNN, therefore, is suitable for

applications like keyword spotting (KWS) or face recognition in surveillance, thanks to its event-driven operation.

Spiking neural network based hardware work so far, however, focused on either the acceleration of neural simulations or the improvement of both performance and energy efficiency. In other words, they are not designed for always-on function. For example, Neurogrid (Benjamin et al., 2014) targets large-scale neural simulations. It employs analog neurons and address event representation (AER) for communication, the latter using a multi-bit bus. SpiNNaker (Painkras et al., 2013) also targets neural simulation and employs an array of embedded digital processors communicating asynchronously. Yang et al. presented multiple works that targeted large scale neural simulations. In CerebelluMorphic (Yang et al., 2021b) they simulated portions of the cerebellum related to motor learning using 6 field programmable gate array (FPGA) chips that communicate using a multicast router. In BiCoSS (Yang et al., 2021c) they presented a platform with 35 FPGA chips connected to realize real-time computation of biological activities in multiple brain areas. In another work (Yang et al., 2021d), they presented an event-based processing algorithm that used piecewise linear approximation and binarization for efficient implementation of credit assignment to neurons in neuromorphic hardware. On the other hand, TrueNorth (Akopyan et al., 2015) was designed to be a scalable low power neurosynaptic inference engine for SNNs. The architecture was event-driven and employed synchronous circuits for computation blocks and asynchronous circuits for communication. Also, Tianjic chip was designed to support inference only with both neuromorphic and deep-learning models (Pei et al., 2019). Some works proposed architectures for both the training and inference of SNNs. Koo et al. (2020) introduced the implementation of a stochastic bit and used it in the realization of a neuron and synapse. They support on-chip training and inference with the synapse being stochastic in training and neuron being stochastic in both training and inference. Chen et al. (2018) presented an SNN accelerator with on-chip spike-timing-dependent plasticity (STDP) based learning. This chip has 64 cores that communicate using a network-on-chip (NoC) with each core supporting 64 leaky integrate and fire (LIF) neurons. Also, Loihi (Davies et al., 2018) was designed to support a variation of the current based dynamics LIF neuron model and a wide range of synaptic learning rules for both supervised and unsupervised learning. It is built for performance. It has 128 cores, three x86 cores, off-chip interfaces and an asynchronous NoC for communication between cores. Also, Seo et al. (2011), implemented a scalable architecture with a set of 256 neurons and transposable memory for synapses in near-threshold voltage (NTV) circuits. It mapped an auto-associative memory model. Some other works implemented different learning rules for on-chip training. Knag et al. (2015), implemented a feature extractor based on a sparse coding algorithm using LIF neurons. Park et al. (2019), developed a new neuromorphic training algorithm and hardware which supports low overhead on-chip learning. Some of these chips e.g., (Akopyan et al., 2015; Davies et al., 2018) employ asynchronous logic such as quasi-delay-insensitive (QDI) dual-rail dynamic logic or bundled data communication. Asynchronous logic circuits are, however, generally bulkier and power-hungrier than the single-rail static counterpart and also not very voltage-scalable (Chen et al., 2013; Liu et al., 2013) and bundled data communication incurs significant overhead because of the handshake. Some other chips employ power-efficient static logic (Chen et al., 2018; Davies et al., 2018; Park et al., 2019; Pei et al., 2019), but they target high throughput, not always-on function. As a result, they exhibit a power consumption of more than tens of mW, which makes it difficult to use them for always-on functions.

In this work, we focus on ultra-low-power *always-on* inference hardware and propose an SNN classifier consuming less than 300 nW. Our architecture uses fully spike-based event-driven operation and only static logic operating at a NTV to achieve such low power. Specifically, our design is centered around the neurosynaptic core. It is implemented using static gates and spike-driven (i) spatiotemporally fine-grained clock generation, (ii) clock-gating, and (iii) power-gating. Also, the communication between neurosynaptic cores is free from information loss due to the collision of spikes, despite using only wires to connect the cores. The architecture exhibits active power consumption that is proportional to the input rate due to its event-driven nature.

We also employ the technique in Cao et al. (2014) to train a neural network with binary weights and use the weights for the SNN we intend to deploy. The use of binary weights is a recent development in deep learning for making inference efficient (Courbariaux et al., 2015). They are of special interest because of their reduced memory footprint and simple computations. They are well suited for low power hardware and attain close to state of the art accuracy on datasets like MNIST. On the other hand, we keep the activations as spike-rate-coded multi-bit values, which improves the model's inference accuracy.

We prototyped an SNN classifier in 65-nm LP CMOS technology. It has 5-layers and a total of 650 neurons and 67,000 synapses. It consumes 2.3–6.8X lower power at state-of-the-art accuracies on two well-known KWS benchmarks, i.e., Google Speech Command Dataset (GSCD) for multi-keyword recognition (Warden, 2018) and HeySnips for single-keyword spotting (Coucke et al., 2019).

In the remaining portion of this manuscript, we will present our SNN hardware architecture and the experimental results. In section "Materials and Methods," we discuss the high-level SNN classifier architecture, elaborate on each of the components of the neurosynaptic core and introduce the experiment setup. In section "Results," we present the results and finally conclude in section "Discussion."

MATERIALS AND METHODS

The SNN classifier in our proposed design is depicted in **Figure 1**. It can support a fully connected network as large as 256-128-128-128-10 with binary weights onto five neurosynaptic cores which is sufficient to support the KWS task. We map each layer of the network to a different neurosynaptic core. The neuron block in the neurosynaptic core for the input layer has 256 neurons while the ones for the hidden layers, each contains 128 neurons.

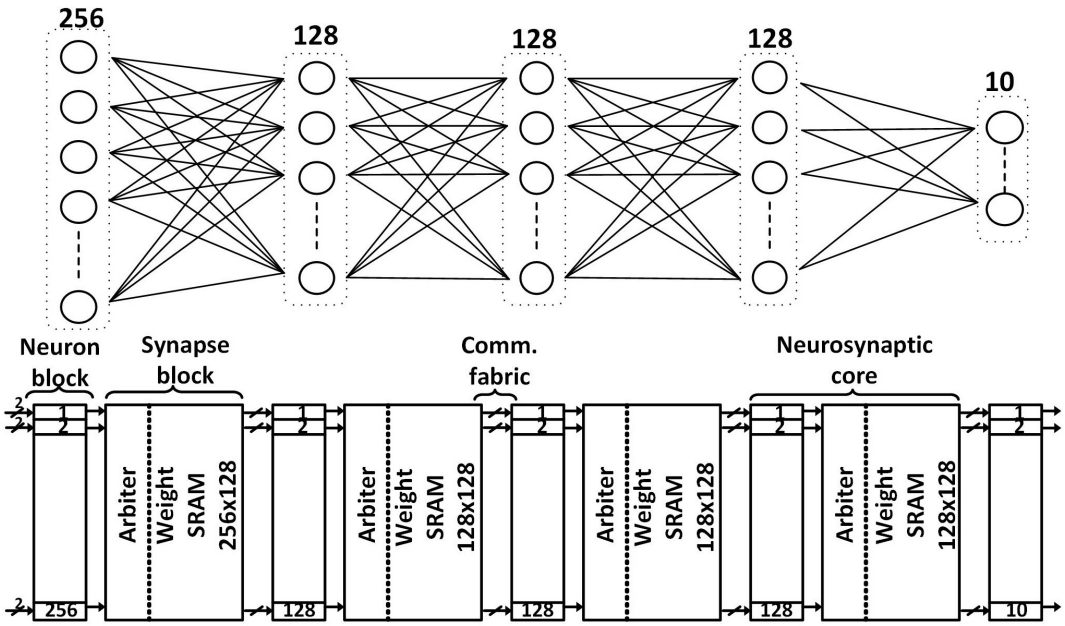


FIGURE 1 | The proposed SNN classifier architecture **(bottom)** with the maximum supported network size **(top)** for always-on functions like keyword spotting.

Each neuron has its own hardware and thus they can process in parallel. The size of each layer can be altered to make it smaller by configuring the neurosynaptic core using the scan chain. The architecture needs to change if a much larger network needs to be supported while not increasing the area for tasks like object identification in a security video, necessitating time-sharing of neuron hardware.

The input and hidden neurosynaptic cores have a neuron block and a synapse block while the output neurosynaptic core has only a neuron block. A neuron block contains all the IF neurons in that layer, a synapse block has (i) an arbiter, (ii) an SRAM storing up to 256-by-128 binary weights for the input neurosynaptic core and up to 128-by-128 binary weights for the hidden neurosynaptic cores, and (iii) a spike generator that simultaneously generates 128 spikes.

Neuron Block

We propose a spike-event-driven architecture. **Figure 2** shows the neuron block based on that architecture. Each neuron has (i) asynchronous wake-up circuits and (ii) a synchronous finite state machine (FSM). Also, all the neurons in a neuron block share a clock generator based on a ring oscillator. The architecture contains fine-grained clock-generation and clock-gating circuits based on spike input as an event. In the absence of input spikes, each neuron gates its clock and also power-gates the non-retentive parts of the neuron using zigzag power-gating switches (PGSs) (Cerqueira and Seok, 2017), to reduce static power dissipation. In zigzag power-gating, if the circuit in the power down state, the gates are left in alternate states by default, reducing the capacitance that needs to be charged while transitioning into power on state.

The wake-up circuit of each neuron (**Figure 2A**, left) has the static flip-flops, FF_{+1} and FF_{-1}, which detect the rising edge of the incoming spikes from two inputs, Spk_{+1} and Spk_{-1}. Positive spikes which increase the potential of the neuron are directed to Spk_{+1} and negative spikes which decrease the potential to Spk_{-1}. As shown in **Figure 3A**, the detection of a spike makes the output of the clock-enable flip-flop (FF_{clk-en}) high. It also un-gates the PGS of the neuron. Thanks to the zigzag PGS, the ungating (i.e., wake-up) is done in a single clock cycle.

This process starts up the shared clock generator in the neuron block if it was not already started by another neuron. The shared clock generator contains a configurable ring oscillator and a clock divider. The length of the ring oscillator and the divisor for the clock divider are determined during testing to obtain the desired clock frequency. The first falling edge of the clock generator's output after an active FF_{clk-en} sets the un-gate flip-flop (FF_{clk-ug}) to high, ungating the clock signal that goes into the FSM. The use of FF_{clk-ug} ensures that there is a complete low phase of the clock signal before the rising edge at the clock input of the FSM, giving sufficient setup time to the flip-flops in the FSM.

Once awoken, the neuron FSM gets executed. The FSMs are slightly different for the input core, hidden cores, and output core (**Figures 2B–D**). In the case of hidden neurons, the FSM, as shown in **Figure 2B**, enters the *Potential Update* state on receiving the positive edge of the clock. The neuron's potential is increased or decreased by one based on the input spike's type. Then, the neuron's potential is compared with the preset threshold (TH) in *Check Pot.* State. The neuron contains a 9-bit adder/subtractor to increment/decrement potential and to compare the potential with the threshold. If the potential is less than the threshold, the FSM goes back to the *Start/Standby* state while resetting all the flip flops in the wake-up circuit (FF_{+1}, FF_{-1}, FF_{clk-en}, and

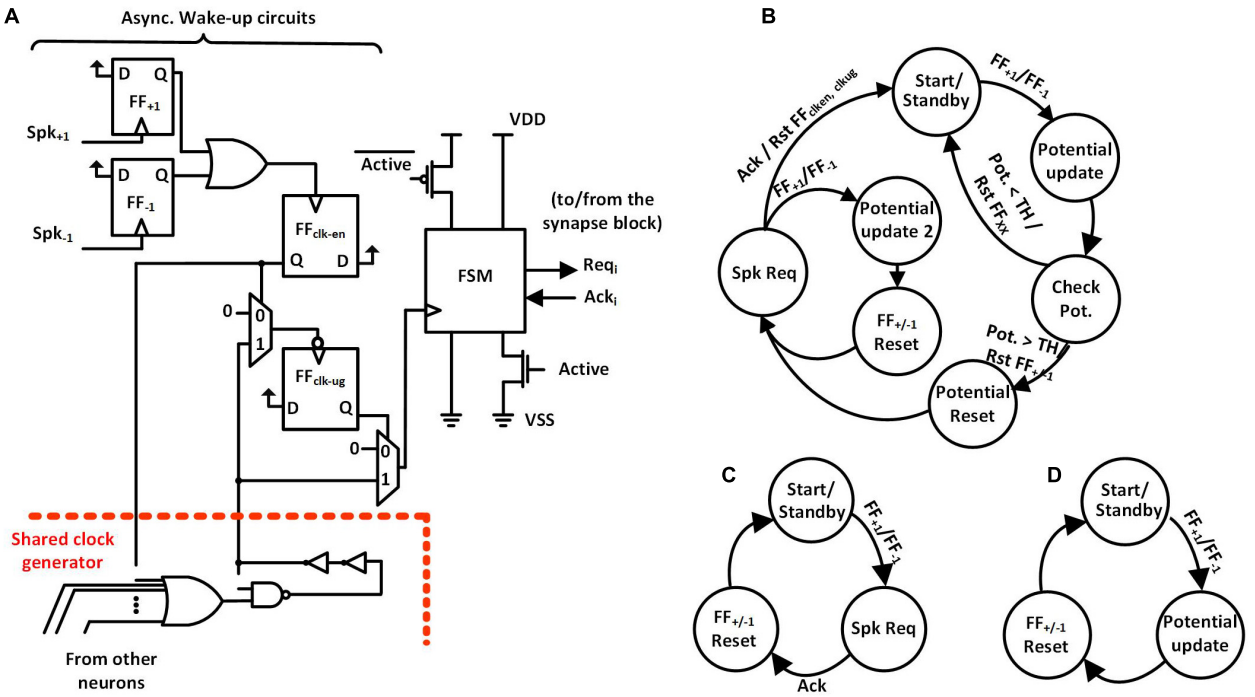


FIGURE 2 | (A) Neuron Block Architecture with wake-up circuits on the left, FSM with zigzag power gating on the right and Shared Clock generator on the bottom. **(B)** Hidden Neuron FSM. **(C)** Input Neuron FSM. **(D)** Output Neuron FSM.

FF_{clk-ug}; find them in **Figures 3A,B**). Otherwise, it resets the neuron's potential to zero and also FF_{+1} and FF_{-1} in the *Potential Reset* state, allowing for receiving the next spike (**Figure 3C**). The FSM then enters the *Spk Req* state, asserts the firing request (Req_i) and waits for the acknowledgment (Ack_i) from the arbiter in the synapse block. While waiting for Ack_i, if the FSM receives a new spike it enters another state, *Potential update 2*, where the neuron's potential is calculated. Once Ack_i from the arbiter is received, the neuron's FSM goes back to the *Start/Standby* state after resetting the flip-flops (FF_{+1}, FF_{-1}, FF_{clk-en}, and FF_{clk-ug}) in the asynchronous wake-up circuits. This cuts off the clock and power to the neuron.

The operation of input and output neurons are slightly different. The input neuron's FSM is depicted in **Figure 2C**. On receiving a spike, the FSM directly enters the *Spk Req* state, asserts a firing request (Req_i) and waits for an acknowledgment (Ack_i) from the arbiter in the synapse block. On receiving Ack_i from the arbiter, the FSM resets FF_{+1}, FF_{-1}, FF_{clk-en}, FF_{clk-ug} and goes back to the *Start/Standby* state. Again, in this state, the clock and power to the neuron are gated. The output neuron's FSM is depicted in **Figure 2D**. Upon receiving a spike, the FSM enters the *Potential Update* state, then in the next state, it resets FF_{+1}, FF_{-1}, FF_{clk-en}, FF_{clk-ug} and then goes back to the *Start/Standby* state. The output neuron does not generate any spikes and only keeps track of the potential. The neuron with the highest potential determines the classification result.

This spike-based event-driven operation enables large power reduction and energy savings. First, if the input has no activity, which is common for always-on applications, the proposed neuron architecture can enjoy a very long sleep time. The hidden neuron without spike-event-driven power management would consume 1.16 nW as shown in **Figure 4A**. The proposed clock-generation/-gating enables 74.6% power savings and the zigzag power gating provides an additional 17.68%, resulting in an overall power reduction of 4.8X when the circuit is not processing any spikes.

If the input has non-zero activity, the proposed neuron will experience shorter sleep time but it still saves a considerable amount of energy. For the targeted benchmarks, the shortest idle time between two spikes per neuron is estimated to be around 4 ms at the maximum input rate. **Figure 4B** shows the energy consumption of the hidden neuron as a function of sleep time obtained using SPICE simulation. The energy consumed includes the overhead of transitioning in to and out of the power down state and the energy consumed during sleep. We consider the hidden neurons with no low power technique used, with only clock gating used, and with both clock and power gating used. We can observe that the neuron with clock and power gating can save energy consumption by 4.35X for 4 ms sleep time. Also, if the sleep time of the neuron is greater than 1.3 ms, we stand to gain due to the proposed fine-grained clock and power gating. The shortest idle time between two spikes would be much smaller for SNN accelerators that target high throughput, making it challenging to obtain any benefit from fine-grained power gating.

Synapse Block

The synapse block was also designed based on the event-driven architecture. **Figure 5A** shows its microarchitecture. The synapse

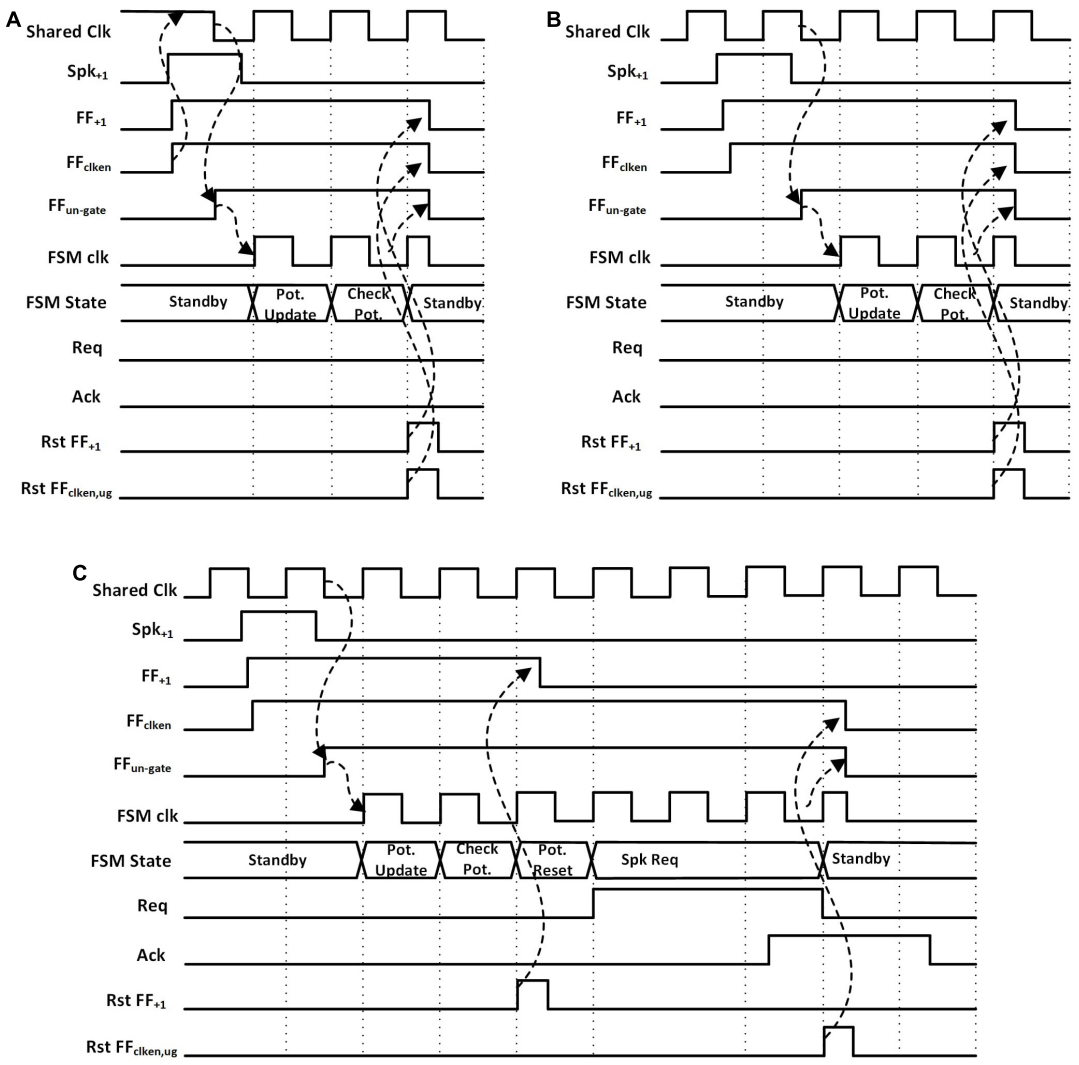


FIGURE 3 | Waveforms for a hidden neuron FSM when **(A)** potential is less than the threshold and shared clock was disabled, **(B)** potential is less than the threshold and shared clock is running (assume other neurons in the same neuron block are active), **(C)** potential is greater than the threshold and shared clock is running.

block has an arbiter FSM, an SRAM array, spike generators, and its own clock generator. A request signal (Req_i) from the neurons within the same neurosynaptic core starts the local clock generator of the synapse block, which makes the arbiter FSM get executed. In case multiple neurons assert Req_i, the arbiter handles the requests, i.e., grants access to the single-port weight SRAM based on a fixed priority. To serve n-th neuron's request, the arbiter asserts the n-th wordline (WL_n) and loads the binary weights on the read-bitlines (RBLs) whose values are captured by the flip flops. Each row of the SRAM contains 128 binary weights which are equal to the number of neurons in the neurosynaptic core. This means all the weights needed to serve a neuron's request are obtained in a single access. The spike generator uses these weight values to generate 128 positive or negative spikes to the neuron in the next layer. It is to be noted that the spike generator is connected to the neurons in the next neurosynaptic by wires only. The arbitration among the neurons also has the effect of managing access to these wires by allowing only one spike per wire at once. Therefore, we avoid the loss of information due to the collision between two (post-synaptic) spikes traveling to a single neuron at the same time.

When the local clock generator is enabled, the arbiter FSM gets executed (**Figure 5B**). The FSM starts in the *Start/Standby* state and when the positive edge of the clock arrives the FSM moves to one of the *Ack[i]* states. The exact *Ack[i]* state is determined based on the indices of the neurons making the request. The neurons with a smaller index have a higher priority.

The waveforms in **Figure 5C** show an exemplary operation of the circuit when neuron 1 and neuron 2 generate a request at the same time. We can see from the figure that once the requests are generated, the FF_{clk-en} flip-flop is set. This turns on the local clock generator and disables power gating. Acknowledgment (Ack_1) is provided to neuron 1 because it has a higher priority

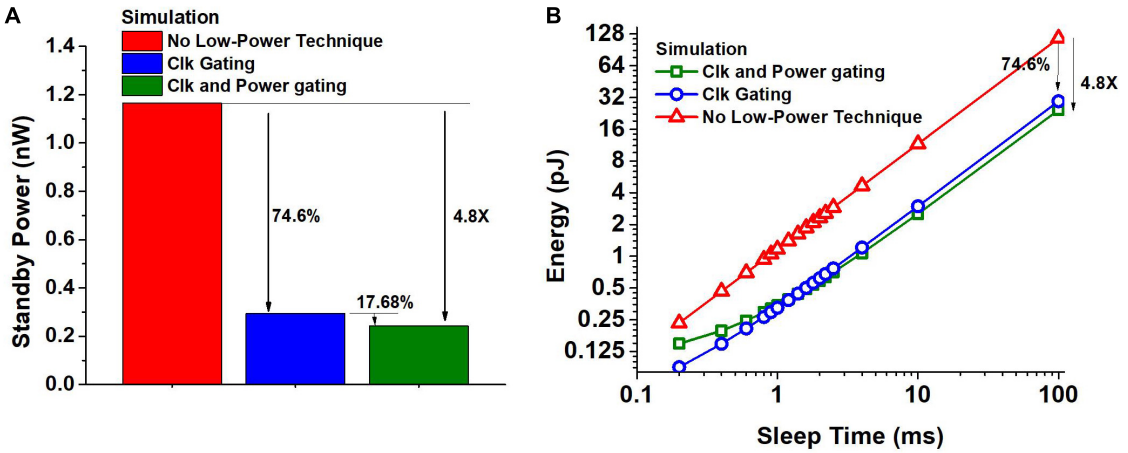


FIGURE 4 | (A) Impact of spike-driven clock gating and a combination of clock and power-gating on the standby power consumption of a hidden neuron obtained using SPICE simulation. **(B)** Energy consumption of the hidden neuron obtained using SPICE simulation as a function of sleep time between two spikes, when the clock is free running and when clock gating and power gating are used.

determined in design time. The same acknowledgment signal acts as the read WL_n for the SRAM.

The arbiter then starts executing the spike generation sub-FSM (*Spkgen*). The *Spkgen* waveform in **Figure 5C** shows the state of the sub-FSM. When Spkgen is in the state *St1*, weight values are captured in flip-flops and when Spkgen enters state *St2*, 128 positive or negative spikes ($spk_{+/-1}$) are generated for all the neurons in the next layer based on the weight values. The arbiter acknowledges back to neuron 1 by asserting Ack_1 while the spike generator goes through the states *St0*, *St1*, and *St2*. Ack_1 stays high until the request from the neuron is high or the spike generation completes, whichever is later. If there are any outstanding Req_i, the arbiter FSM continues to serve, otherwise, the clock and power are disabled.

We chose the fixed priority arbiter instead of a round-robin one as the area saving is about 17X for 128 inputs. **Figure 5D** shows the area of the round-robin arbiter and fixed priority relative to a fixed priority arbiter with 32 inputs. We can see from **Figure 5D** that the area required for a round-robin arbiter is superlinear as a function of the input size while the area for a fixed priority arbiter increases approximately linearly with the number of inputs.

The fixed priority scheme, however, could cause the neuron with the lowest priority to starve, i.e., its requests may not be served if the arbiter is busy serving the requests of the neurons with higher priority. We can address the problem of starvation by increasing the bandwidth of the SRAM or reducing the requests that neurons make. In our design process, we ensure the fixed priority arbiter starves no neuron. We improved the bandwidth of SRAM using supply boosting which is discussed in section "On-Chip SRAM." We chose the thresholds of the neurons and the clock frequencies of the neuron and synapse blocks so that spikes are not missed while the neurons are waiting for acknowledgment from the arbiter.

The process to determine those key design parameters is as follows. As shown at the bottom of **Figure 6**, we have considered a case where a neuron receives spikes from $N_{\mathrm{nrn,i}}$ neurons and produce spikes, where we can formulate the number of requests in the i-th layer ($N_{\mathrm{req,\ i}}$), which is:

$$N_{req,i} = \frac{N_{spk,i} \times N_{nrn,i}}{TH_i}, \tag{1}$$

where TH_{i} is the threshold of the neurons, $N_{\mathrm{spk,\ i}}$ is the number of incoming spikes in a particular time period (called a frame) and per neuron, $N_{\mathrm{nrn,\ i}}$ is the number of neurons, all in the i-th neurosynaptic core. On the other hand, the number of requests that the arbiter in the i-th layer can serve ($N_{\mathrm{serve,\ i}}$) can be formulated as:

$$N_{serve,i} = \frac{f_{clk,a} \times T_{frame}}{N_{cyc,a}} \tag{2}$$

where $N_{\mathrm{cyc,\ a}}$ is the number of cycles that the arbiter consumes to serve one request, T_{frame} is the frame size, $f_{\mathrm{clk,\ a}}$ is the arbiter's clock frequency.

If $N_{\mathrm{req,\ i}}$ (Eq. 1) exceeds $N_{\mathrm{serve,\ i}}$ (Eq. 2), starvation occurs. When starvation occurs, incoming spikes can get dropped as the arbiter is not fast enough to serve all the requests. We ensure by design no spike is dropped, i.e., by making $N_{\mathrm{req,i}}$ not exceed $N_{\mathrm{serve,i}}$. This is done by increasing TH_{i} or increasing $f_{\mathrm{clk,\ a}}$. The former, however, can incur a degradation in the accuracy. This is because the increase of TH_{i} would reduce the number of output spikes generated in the i-th layer. It has the same effect as reducing the precision of the activations in a binary-weight neural network that has a similar network structure. On the other hand, increasing $f_{\mathrm{clk,a}}$ increases the power consumption of the synapse block. Therefore, we swept TH_{i} and $f_{\mathrm{clk,\ a}}$ values to find optimal operating points for the chip. **Figure 7A** shows a curve obtained using RTL simulation with 1000 MNIST test samples (LeCun et al., 1998) and 8-bit activations. The curve separates the regions where the neurons starve and where they do not. We choose design points so that the average number of spikes per neuron is roughly the same for each of the hidden layers and

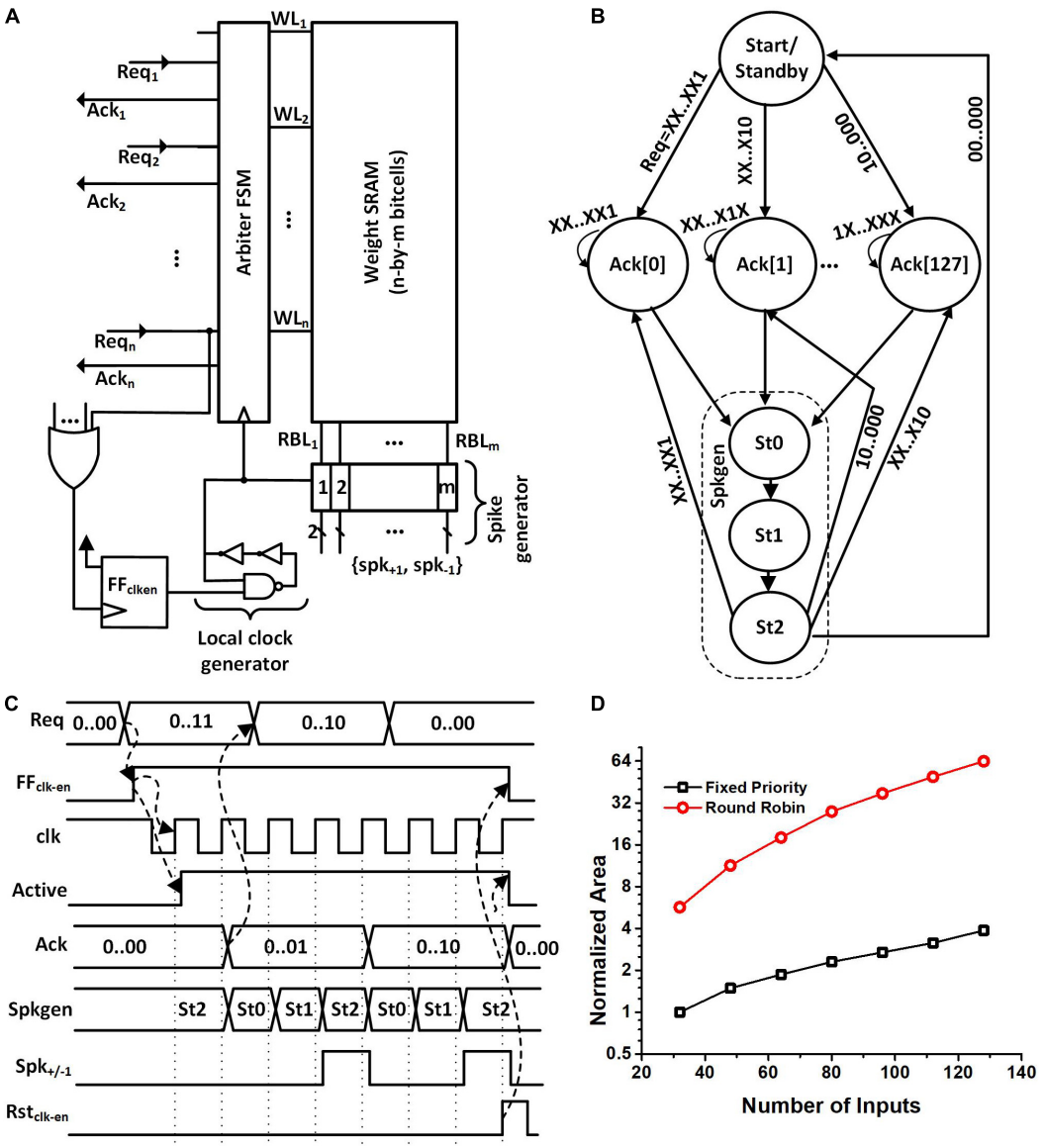


FIGURE 5 | (A) Proposed synapse block architecture. **(B)** Arbiter FSM showing the fixed priority and Spkgen sub-FSM. **(C)** Waveforms showing the operation of the synapse block when neuron 1 and 2 generate a request. Acknowledgment is given to neuron 1 because of higher priority. Spkgen sub-FSM executes while the acknowledgment is high. **(D)** Normalized area comparison between round-robin arbiter and fixed priority arbiter for a different number of inputs. The normalized area is obtained by dividing the cell area with the area of the fixed priority arbiter with 32 inputs.

hence the curve that separates the starvation and non-starvation region is the same for all of them. The design point, i.e., the neuron threshold and the synapse block clock frequency for each of the hidden layers is indicated as a red star in **Figure 7A**. The threshold values we chose for the hidden layers for the MNIST dataset are (32, 16, and 14) and the threshold values we chose for the KWS datasets with 6-bit activations are (28, 18, and 10). The threshold values are not very different for the two kinds of datasets despite the difference in the desired spiking rate because of the dependency on weight and input data.

Indeed, the threshold value affects the number of spikes generated in a layer and this affects the inference accuracy. Recall that the activations are spike-rate-coded multi-bit values and the reduction of the number of spikes leads to fewer bits. We can observe the impact of the choice of threshold values for different hidden layers on the accuracy of the SNN classifier in **Figures 7B–D**. It shows through a Python simulation the accuracy obtained on 300 test samples of the MNIST dataset. The Python simulation models the neuron and arbiter and uses a set of time series vectors as input spike train whose entries are either 1,−1, or 0 indicating the presence and the sign of the spike. We chose the time resolution so that the results mimic the RTL simulation. For **Figure 7B**, we varied the threshold for the first hidden layer while the thresholds of the second and

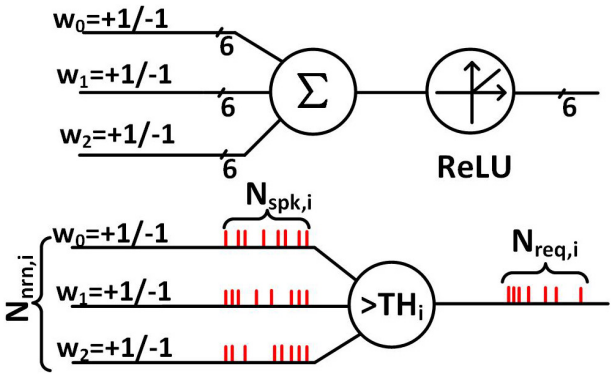


FIGURE 6 | Binary coding in a BNN and spike-rate coding in an SNN.

third hidden layers are chosen to be 16 and 8. For **Figure 7C** we varied the threshold of the second hidden layer and kept the threshold of the first hidden layer to be 24 and that of the third hidden layer to be 8. For **Figure 7D** we varied the threshold of the third hidden layer and kept the threshold of the first hidden layer to be 24 and that of the second hidden layer to be 16. From **Figure 7B** we can observe that the accuracy of the classifier is worse for the small (roughly < 5) and the large threshold values (roughly > 40). **Figure 7B** also shows the total number of neuron requests dropped across layers as a function of the threshold of the first hidden layer. It indicates that if the threshold is too small, too many spikes are produced, causing starvation, which leads to too many neuron requests being dropped, resulting in a deterioration in the accuracy. **Figures 7C,D** show a similar trend when the threshold for the second hidden layer and the third hidden layer is varied. But we can also observe that the impact of the threshold of the second and third hidden layer on accuracy is relatively small if a proper threshold is chosen for the first hidden layer. This is because the number of spikes and hence the number of neuron requests are large in the first hidden layer. The threshold of the first hidden layer determines the number of requests dropped in the first hidden layer which is also a large portion of the total number of requests dropped.

On-Chip SRAM

The chip has 65.25 kb of SRAM and so it was important to minimize SRAM leakage power dissipation. We designed the SRAM based on the circuit described by Cerqueira et al. (2019) for ultra-low-power operation. High threshold voltage (HVT) transistors with three times minimum length were used for the bitcell to reduce leakage. The buffer in the peripheral circuits employed zig-zag power gating with cut-off transistors separate for each row, ensuring fast wake-up.

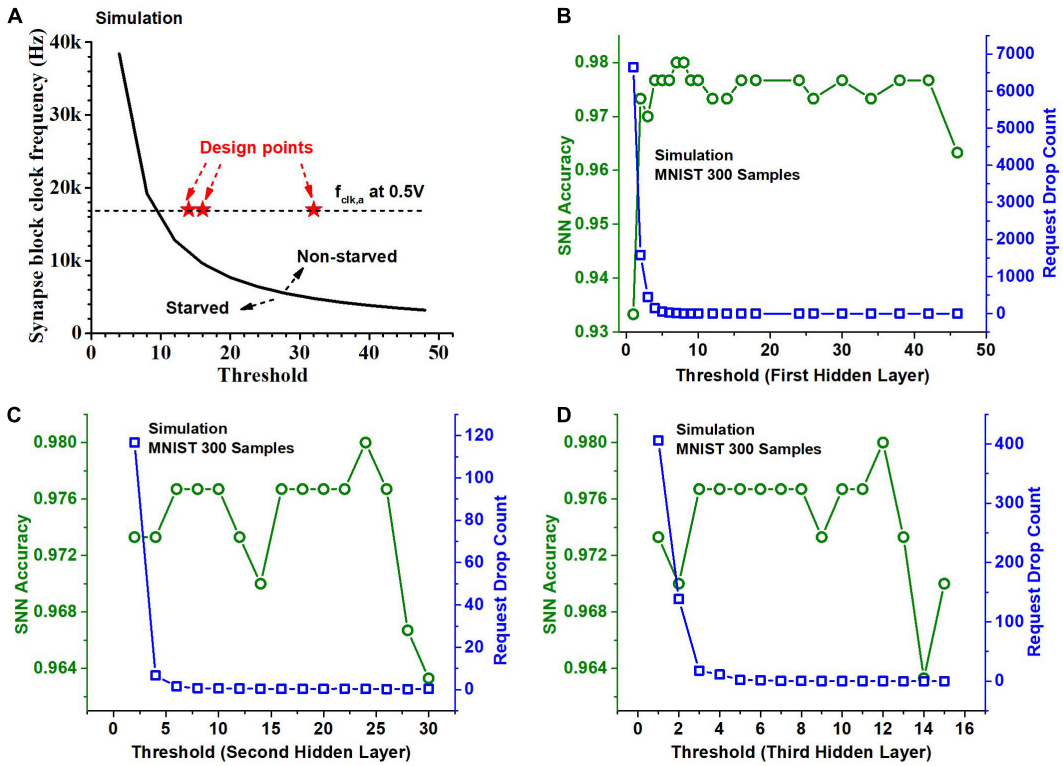


FIGURE 7 | (A) Threshold and clock frequency optimization for no starvation during operation. Neurons are not starved when the arbiter clock frequency and the threshold for the layer are high enough. **(B)** SNN accuracy on 300 samples of the MNIST dataset and the total number of neuron requests dropped (request drop count) per sample when the threshold value for the 1st hidden layer is varied. **(C)** SNN accuracy and Request drop count when the threshold value for the 2nd hidden layer is varied. **(D)** SNN accuracy and Request drop count when the threshold for the 3rd hidden layer is varied.

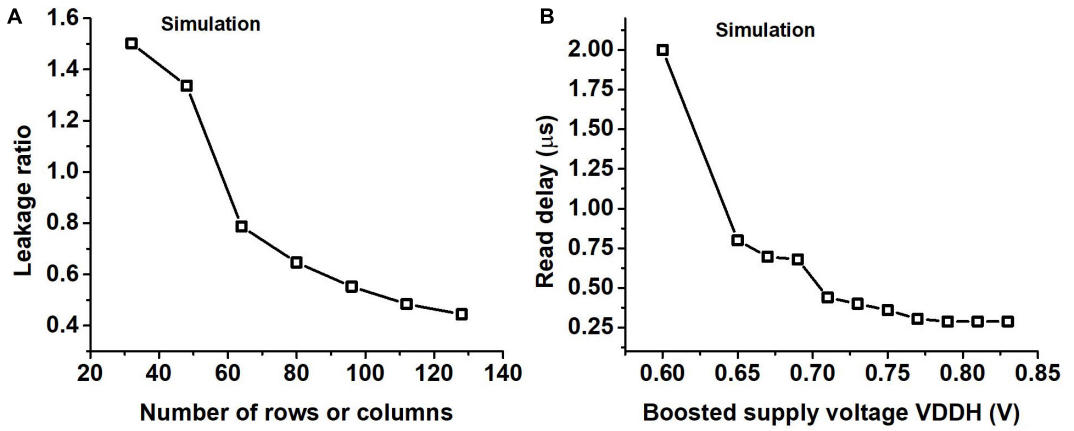


FIGURE 8 | (A) The ratio of the leakage of the peripheral circuits to that of the bitcells (leakage ratio) for different SRAM sizes where the number of rows is the same as the number of columns obtained using SPICE simulation. **(B)** Read delay of the SRAM for different boosted supply voltages (VDDH) obtained using SPICE simulation.

We chose to have all the weights for a layer in a single SRAM macro size instead of smaller banks. **Figure 8A** shows the ratio of the leakage of the peripheral circuits to the leakage of the bit cells as a function of the number of rows obtained using SPICE simulation. From **Figure 8A** we can see that because of this the leakage of peripheral circuits would get amortized among more bitcells, helping us in the overall objective of reducing the leakage.

We use supply voltage boosting during a read operation to speed up the charge or discharge of the read bitline. The delay in the read operation arises mostly from charging the read WL_n and charging or discharging the read bit line. Supply voltage boosting was needed to improve the speed of read operation which took a hit due to the use of a single SRAM macro for storing all the weights in a layer. We performed a transistor-level SPICE simulation to observe the read delay of SRAM for different values of the boosted supply voltage and the results are shown in **Figure 8B**. In **Figure 8B** we can observe that on increasing the boosted supply voltage (VDDH) we will eventually be limited by the time taken to charge the WL_n. We operate our circuit so that read delay is not the critical path in the design by choosing a high enough VDDH, which is roughly 0.8 V if the regular VDD is set to 0.52 V.

Experiment Setup

Chip Prototype

We prototyped the test chip in a 65 nm LP CMOS process. **Figure 9A** shows the die photo with the boundaries of different cores marked. The input and the hidden cores have the dimensions 0.7 mm × 0.7 mm. The output neurons take an area of 0.0276 mm^2. Each of the hidden cores is logically equivalent but have a different layout to simplify the routing. The total core area is around 1.99 mm^2. The area breakdown of the chip can be seen from the pie chart in **Figure 9B**. The chip also contains the input decoder and the output encoder for reducing the number of I/O the chip would require. Those decoder and encoder convert the spike I/Os to the binary address in AER, reducing the spike I/O pin count from 512 to 9. Also, the chip contains a scan chain to configure the thresholds of the neurons in different neurosynaptic cores, set the clock frequency of the neuron and synapse blocks and write the weights into the SRAMs.

Input Preparation

We envisioned the SNN to interface directly with a spike-generating feature-extraction front end such as the ones discussed in Yang et al. (2015, 2019, 2021a). For our experiments, we used the software model (Yang et al., 2019) to generate features for the KWS task. The software model makes use of post-layout Spectre simulations for tuning its parameters and has been validated using chip measurements. In the analog front end, the spikes are generated when the voltage on a capacitor exceeds a certain threshold (Th_{analog}). The finite bandwidth of the comparator and Th_{analog} together control the spike frequency. We do not alter the value used for Th_{analog} across the HeySnips and the GSCD datasets.

The software model generates features of size 16. Each dimension of the feature captures the energy at a central frequency in the form of the number of spikes that are generated by the analog front end in a certain time period. We call this time period as frame size in case of audio input. The central frequencies of the 16 channels are geometrically scaled from about 100–5 kHz. We configure the front end so that the number of spikes can be represented by 6-bits, i.e., each element in the feature has 6-bit precision. We set a frame size T_{frame} of 80 ms with no overlap between successive frames, based on the length of the audio clip of the datasets and the dimension of the input layer that the chip supports. In GSCD and HeySnips datasets, each keyword audio sample is roughly 1s. We put together the feature vector of the current frame along with the feature vectors of the past 15 frames to obtain a vector of size 256 that contains the number of spikes associated with each input neuron. We evenly spread out the spikes for each input neuron within a time period equal to the frame size. The FPGA then sends these spikes to the input decoder inside the SNN chip in the form of AER.

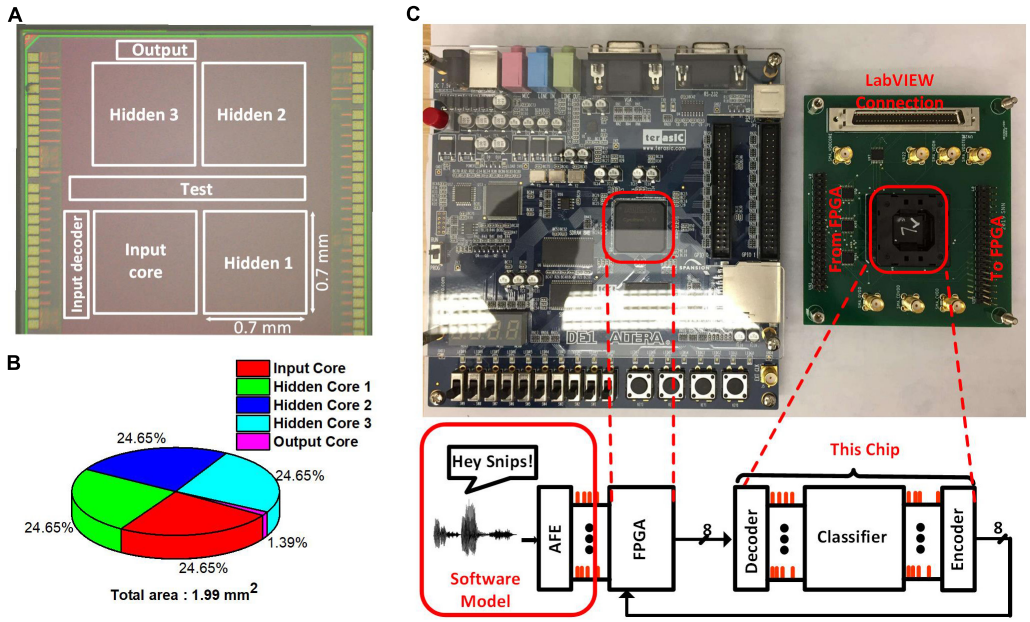


FIGURE 9 | (A) Die photo of the SNN Classifier with the Neurosynaptic Cores along with the Test Circuits. **(B)** Area break-down of the SNN Classifier. **(C)** Test chip with its connection to the FPGA board and LabVIEW. FPGA interface is used for sending the inputs to the chip and reading out the potential of the output neurons. LabVIEW is used for configuring the thresholds and write to the SRAMs.

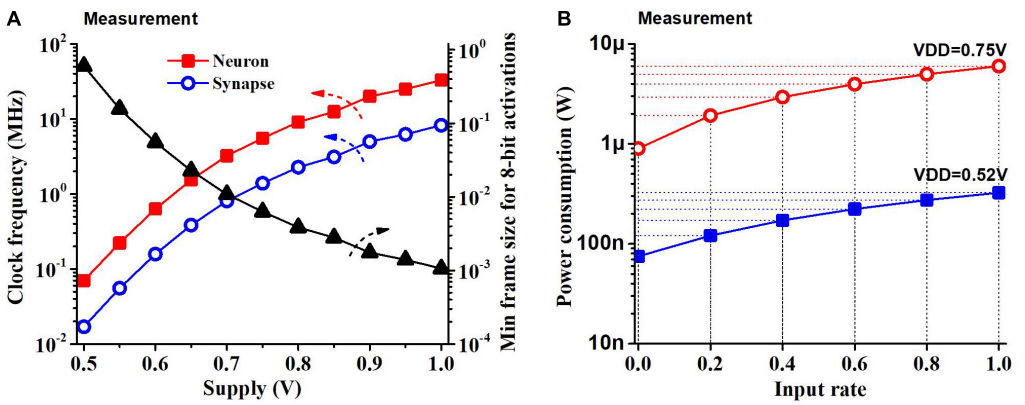


FIGURE 10 | (A) Clock frequency measurement and min frame length for 8-bit activations as a function of the supply voltage. Frame size is constrained by the latency of the SNN classifier (i.e., longer latency increases the minimum frame size). **(B)** Power consumption of the chip as a function of the input rate at two supply voltages. Power consumption increases linearly with the input rate.

In the case of the MNIST grayscale dataset, we downsample the image size to 16 × 16 by utilizing 2 × 2 max-pooling so that we can match the image with the size of the input layer of the chip. For each input sample, we generate a set of time series vectors (spike trains) for the input layer of the chip in a time duration equal to the frame size, which is chosen based on the latency of the chip.

Training

We train a binary neural network (BNN) model that uses binary weights (+1, −1), has no bias and 6-bit ReLU activation (Cao et al., 2014) for the KWS task. The network structure is equivalent to the SNN model we deploy. The BNN provides the weights for the SNN model. The 6-bit activations in the BNN are encoded for the SNN using spike-rate, e.g., $010000_{(2)}$ is mapped to 16 spikes/frame. We set the threshold of the neurons in each layer such that each neuron generates at most 63 spikes/frame (i.e., $N_{spk,i}$ in **Figure 6** is less than 63), which matches the 6-bit activation of the BNN model. This is possible because in the SNN model, as spikes pass through the neurons in a layer, the number of spikes scales roughly by the ratio of the threshold. We can easily change the activation precision after deployment for different models by configuring the thresholds. For example,

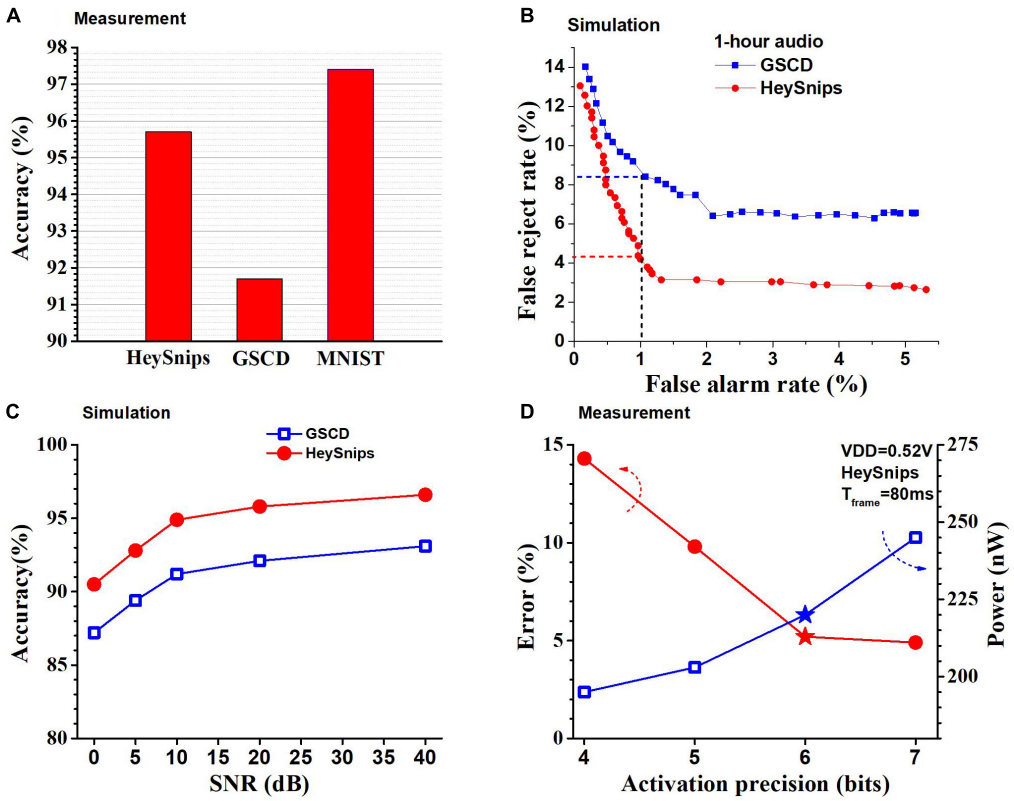


FIGURE 11 | (A) Accuracies of the SNN chip measured across multiple benchmarks. **(B)** ROC curves from KWS benchmarks obtained using RTL simulation. **(C)** Accuracy of the SNN chip on KWS datasets across 0–40 dB SNR levels obtained using RTL simulation. **(D)** Measured power consumption of chip and error for the HeySnips dataset as a function of activation precision. Stars denote the operating point used for comparison with other works.

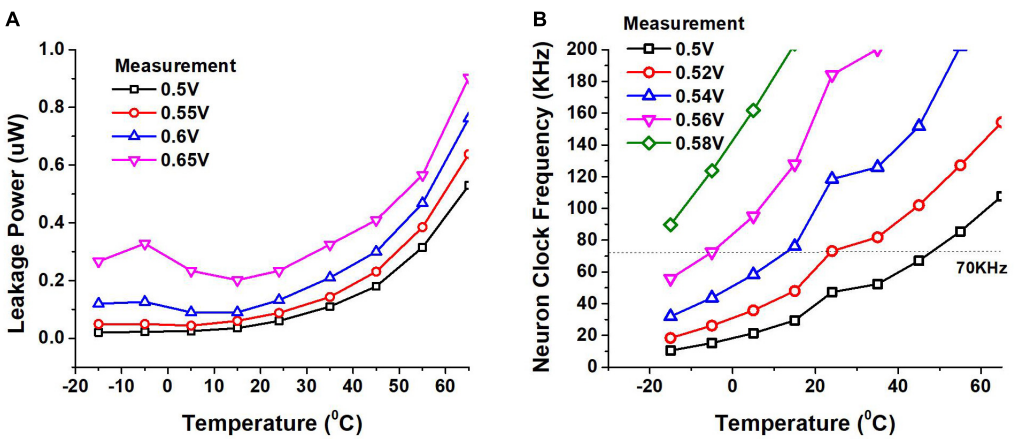


FIGURE 12 | (A) Leakage power of the chip measured as a function of temperature at different supply voltages. **(B)** Variation of neuron clock frequency measured as temperature and supply voltage vary.

we use 8-bit activation with the T_{frame} of 0.5 s for the MNIST grayscale.

Inference Testing

Altera DE1 board containing a Cyclone II FPGA chip is used to interface with the input decoder and the output encoder in the SNN chip as shown in **Figure 9C**. The SNN chip along with the FPGA is globally synchronous but locally asynchronous. The global clock comes from the FPGA and is used to send new inputs and read out the potential of output neurons at regular intervals. The input decoder and the output encoder are synchronized to the global clock but are asynchronous to the clock of the input

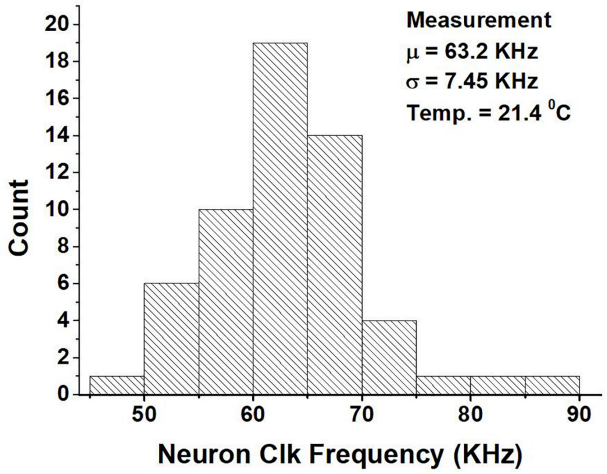


FIGURE 13 | Measurement of the neuron clock frequency of around 50 neurosynaptic cores across 10 chips at a supply of 0.52 V.

core and the output core. All the neuron and synapse blocks within the SNN chip are asynchronous to each other. LabView is used to configure the scan-chain and write weights into the SRAMs in the neurosynaptic cores.

The FPGA board reads out the input data from its memory. It sends an 8-bit AER code to the input decoder identifying the neuron which is supposed to receive a spike and another signal identifying whether the spike is an incrementing spike or decrementing spike. The input decoder then sends a pulse to the appropriate neuron. Spikes that arrive at the input of the hidden layer arrive at all the neurons simultaneously. There is separate hardware for each neuron and hence they can process spikes simultaneously and compete for access to the SRAM. After an interval greater than or equal to the latency of the chip, the FPGA deactivates the output core's clock so that no more spikes are processed. It then enables the output encoder to read out the potential of the output neurons in a serial fashion. The SNN chip is not pipelined, so at the end of the readout, the FPGA resets the potential in all the neurons and sends in the next set of spikes to the input decoder.

For our experiments, we define the latency of the SNN chip as the time needed to process enough spikes to achieve the desired accuracy. If the latency of the chip is less than or equal to the frame size, we can achieve real-time operation. We can stream a new input to the classifier at the end of each frame, which is typically done in audio processing systems. The time period we allow the chip to process is equal to the frame size. The frame sizes are large enough to process most of the spikes and not hurt the accuracy of the task.

RESULTS

Most of the results are based on a supply voltage of 0.52 V and the clock frequency of the neuron block of 70 kHz and that of the synapse block of 17 kHz, while the chip can operate at other supply voltages and achieve different frequencies. **Figure 10A** shows the measurement results of the neuron block frequency, synapse block frequency and latency of the chip (minimum frame size) at different supply voltages when 8-bit activations are used. An off-chip instrument (NI LabView) was used to measure the clock frequency. The latency of the chip was measured by comparing the potential of the output neurons with the results from RTL simulation for 50 samples of the MNIST test set with 8-bit activations. The minimum frame size we can use to operate the chip with 8-bit activations reduces with a supply voltage as the speed of the circuit increases.

We measured the power consumption of the chip during standby and when continuously running (100% input rate) KWS datasets like GSCD or HeySnips. The power consumption of the chip would scale based on the amount of activity at the input. **Figure 10B** shows what the SNN chip power consumption would be at different input rates. We obtain the maximum switching power by subtracting the standby power from the power consumption at a 100% input rate. We obtain the power consumption at an intermediate input rate by scaling the switching power and summing it up with the standby power. The SNN chip dissipates a power of 75 nW when there is no input and power of 220 nW when running a KWS dataset at a supply of 0.52 V.

We *physically measured* the accuracy of the chip and we can see the accuracy of the chip across the different classification tasks in **Figure 11A**. We read out the output neurons' potential to the FPGA and pick the index of the neuron with the highest potential as the predicted class. In GSCD, the SNN can recognize four keywords ("yes," "stop," "right," and "off," arbitrarily chosen) and fillers with an accuracy of 91.8%. The SNN architecture we use is 256-128-128-128-5 and configure the thresholds to be (1, 28, 18, 10) where 1 is the threshold for the input layer (fixed) and the rest are for the hidden layers. For the HeySnips dataset, the chip can recognize one keyword ("Hey Snips") and fillers with an accuracy of 95.8%. For the MNIST grayscale dataset, the trained SNN structure is 256-128-128-128-10 with the thresholds of (1, 32, 16, 14) and it gives an accuracy of 97.6%.

Figure 11B shows the receiver operating characteristic (ROC) curve for GSCD and HeySnips. It shows the false reject rate (FRR) as a function of the false alarm rate (FAR) for 1-h-long audio obtained by concatenating the test set samples and running an RTL simulation. FAR indicates the number of false positives while FRR indicates the number of false negatives. We obtained the ROC by calculating the softmax of the output neurons' potential and varying the discriminating threshold. If the softmax value of the keyword class is greater than the discriminating threshold then the prediction is a keyword otherwise it is a non-keyword. If the discriminating threshold is large (close to 1) most of the audio frames will be classified as non-keyword which will increase the number of false negatives (FRR). If the discriminating threshold is small (close to 0) then most of the audio frames will be classified as keyword thereby increasing the number of false positives (FAR). In the case of GSCD, we take the average of the pairs (FAR and FRR) we obtain for each keyword at a certain discriminating threshold. In addition, we

ran an RTL simulation to obtain the accuracy of the chip in the presence of noise by mixing the speech audio with white noise at various SNRs. We adopted noise-dependent training for this experiment (Yang et al., 2019), i.e., we use the same SNR for both train data and test data. The SNN classifier chip achieves reasonably high accuracy across 0 to 40 dB SNR levels for both GSCD and HeySnips datasets as shown in **Figure 11C**. The configurability of the thresholds of different layers in the SNN classifier architecture allows us to change the data precision after deployment. Recall that changing the threshold in the hidden layers of the SNN has the same effect as changing the precision of the activations in a deep neural network with the same network structure and that activation is spike rate coded in our SNN. This can be used to trade-off accuracy for power savings. **Figure 11D** shows the measured accuracy and the power consumed by the SNN chip when the precision of the activations is varied for the HeySnips dataset. At higher activation precision the error is lower, but the power consumption is higher and at lower activation precision the error is higher, but the power consumption is lower. We chose a precision of 6-bit which is a good compromise between the power consumption and the accuracy.

We also measured the impact of temperature on the leakage power dissipation and the speed of our circuits. We placed our SNN chip and other testing hardware in a temperature chamber for our measurements. **Figure 12A** shows the leakage power of the circuit while **Figure 12B** shows the clock frequency of the circuit at different supplies as the temperature is varied. The margin we provide to the length of the ring oscillator helps us avoid timing failure due to temperature, supply and process variation to a certain extent. While our design does not have a mechanism to dynamically tune the supply voltage or frequency, it is beneficial to operate the circuit at a lower supply when the temperature is high and at a higher supply when the temperature is low, to obtain the needed performance while keeping the power consumption low.

On the other hand, **Figure 13** shows the variation in the neuron clock frequency among approximately 50 cores across 10 chips at a supply of 0.52 V. From the figure we can see that the mean is 63.2 KHz and the standard deviation is 7.45 KHz. The variation in the clock frequency is due to both the difference in the layout of the ring oscillators across cores and chip-to-chip variation. The chip-to-chip variation among the cores is not uniform. The standard

TABLE 1 | Comparisons with recent KWS hardware.

	This work	Shan et al. (2020)	Guo et al. (2019)	Giraldo and Marian (2018)
Technology (nm)	65	28	65	65
Algorithm	SNN	DSCNN	RNN	LSTM
Area (mm^2)	1.99	0.23	6.2	1.035
VDD (V)	0.52–1	0.41	0.9–1.1	0.575
Clock frequency	70 kHz @ 0.52 V	40 kHz	75 MHz	250 kHz
Benchmark 1	GSCD (4 Keywords)	GSCD (2 Keywords)	GSCD (10 Keywords)	TIMIT (4 Keywords)
Accuracy (%)	91.8	94.6	90.2	92.0
Benchmark 2	HeySnips (1 Keyword)	GSCD (1 Keyword)	HeySnips (1 Keyword)	N/A
Accuracy (%)	95.8	98.0	91.9	N/A
Power	75–220 nW*	510 nW**	134 μW	5 μW

**Power consumption scales with input rate; **feature extraction circuits included.*

TABLE 2 | Comparisons with recent SNN hardware.

	This work	Koo et al. (2020)	Park et al. (2019)	Chen et al. (2018)	TrueNorth
Technology (nm)	65	90	65	10	28
Neuron count	650	1	410*	4096	1M
Synapse count	67k	1	N/A	1M	256M
Area (mm^2)	1.99	0.15	10.08	1.72	430
Clock frequency	70 kHz @ 0.52 V	37.5 MHz	20 MHz	105 MHz @ 0.5 V	N/A
			MNIST classification		
Power	305 nW	282.8 mW†	23.6 mW	9.42 mW**	63 mW
Accuracy (%)	97.6	92.3	97.8	97.9	97.6***
Throughput (inf/s)	2	N/A	100K	N/A	N/A
Energy per inference (nJ)	195	N/A	236	1700	N/A
Energy per SOP (pJ)	1.5	8.4 pJ/1.84 pJ††	N/A	3.8	26

**Input layer not included; **estimated from neuron's power dissipation; ***estimated from Hsin-Pai Cheng et al., IEEE DATE 2017; †power reported in Koo et al. (2020) based on network size and power for one neuron and synapse; ††energy with sequencing circuits / Energy without sequencing circuits.*

deviation of the clock frequency varies from 5.6 to 9.5 KHz based on the specific core. If we use the average of the clock frequency of cores within a chip as being indicative of the chip's performance, the chip used for comparison and reporting other measurements has a performance that is about average.

DISCUSSION

Prior works on SNN hardware have focused on non-always-on application (Akopyan et al., 2015; Chen et al., 2018; Davies et al., 2018), support for on-chip training (Chen et al., 2018; Davies et al., 2018; Park et al., 2019) and support for both deep learning and neuromorphic workloads (Pei et al., 2019). The absence of any prior work on SNNs for targeting always-on hardware motivated us to explore a new architecture for SNNs.

We presented a fully spike-event-driven SNN classifier for an always-on intelligent function. We employed a fine-grained clock and power-gating to take advantage of the input signal sparsity, low leakage SRAM and a fixed priority arbiter to achieve a very low standby power of 75 nW. We trained the SNN for multiple always-on functions, notably multi- and single-keyword spotting benchmarks, achieving competitive accuracies. The average power consumption of the SNN chip scales with the input activity rate. It ranges from 75 nW with no input activity and 220 nW with the maximum input activity for the KWS benchmarks.

Table 1 summarizes the comparison of our work with other recent KWS accelerators. Our design achieves 2.3–6.8X power savings compared to Shan et al. (2020) among KWS accelerators. If we scale the area of our design to 28 nm it would be 0.37 mm^2 which is still slightly higher than Shan et al. (2020). The higher area usage of our work is possibly because it does not adopt time-sharing in neuron hardware.

Our work does not have feature extraction circuits. They would increase the area and power when included. We can consider two feature extraction circuits (Yang et al., 2019, 2021a), as candidates for the analog front end for our chip. Yang et al. (2021a) is the improved version of Yang et al. (2019). We used the software model of the analog front end presented in Yang et al. (2019). The power consumed by the analog front end and the feature extraction circuits is 50 nW in the improved version and 380 nW in the older version.

The use of multiple supplies (VDD = 0.52 V and VDDH = 0.8 V) in our work can add some hardware and power overhead. There would be a significant increase in power consumption if we use only 0.8 V as the power supply for our chip. For example, if we assume that power consumption increases quadratically with VDD, then the power increases by 2.4X. We can consider two scenarios that can provide two different supplies and avoid a large increase in power. In one scenario, we assume an external DC-DC converter provides VDD while we can generate VDDH using a capacitor-based charge pump circuits (Kim et al., 2021). The current load of the VDDH is not high since it is used in only a small part of SRAM. Therefore, even if the charge pump efficiency is not high, the overall impact is small. In the other scenario, we assume an external DC-DC converter provides VDDH and then we can generate VDD using an on-chip digital LDO. This LDO would have a power efficiency of 65% (VDD/VDDH), which increases total chip power dissipation by 53.8%.

Table 2 summarizes the comparison of our design with other SNN hardware work (TrueNorth's power is estimated from Cheng et al., 2017). Our design achieves over 30,000X power savings compared to Chen et al. (2018) in **Table 2**. Our design is optimized for ultra-low power always-on functions while others are optimized for a balance between higher throughput and energy efficiency. High-performance SNN accelerators generally assume that input will be presented at a much higher rate, therefore, the time interval between spike events would be much smaller, limiting the benefit of fine-grained clock gating. Our design achieves competitive accuracies among both KWS and SNN hardware works and contributes to a growing body of literature that supports SNNs as an attractive low-power alternative to deep learning based hardware architectures.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

PC and DW designed the chip with PC focussing on the synapse block. DW focussing on the neuron. SK contributed to improving the tape-out flow. JC designed the bitcell for use in the SRAM block while MY contributed to the software that simulates the analog front end. MS supervised the project and is the principal investigator. JK and SJ contributed to the technical discussions. All authors contributed to the article and approved the submitted version.

FUNDING

This research is in part supported by the Samsung Electronics, DARPA (the μBrain program), and SRC TxACE (Task 2810.034).

ACKNOWLEDGMENTS

This article was an extension of the conference manuscript (Wang et al., 2020).

REFERENCES

Akopyan, F., Jun, S., Andrew, C., Rodrigo, A. I., John, A., Paul, M., et al. (2015). TrueNorth: design and tool flow of a 65 mW 1 Million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/tcad.2015.2474396

Benjamin, B. V., Peiran, G., Emmett, M., Swadesh, C., Anand, R. C., Jean-Marie, B., et al. (2014). "Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *IEEE* 102, 699–716. doi: 10.1109/jproc.2014.2313565

Cao, Y., Yang, C., and Deepak, K. (2014). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vision* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Cerqueira, J. P., and Seok, M. (2017). Temporarily fine-grained sleep technique for near- and subthreshold parallel architectures. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 25, 189–197. doi: 10.1109/tvlsi.2016.2576280

Cerqueira, J. P., Thomas, J. R., Yu, P., Shivam, P., Martha, A. K., and Mingoo, S. (2019). "Catena: A 0.5-V Sub-0.4-mW 16-core spatial array accelerator for mobile and embedded computing," in *Proceedings of the 2019 Symposium on VLSI Circuits*, C54–C55. doi: 10.23919/vlsic.2019.8777987

Chen, G. K., Raghavan, K., Ekin Sumbul, H., Phil, C. K., and Ram, K. K. (2018). A 4096-Neuron 1M-Synapse 3.8PJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10NM FinFET CMOS. *IEEE Symp. VLSI Circ. Honolulu HI* 2018, 255–256. doi: 10.1109/VLSIC.2018.8502423

Chen, Y., Mingoo, S., and Steven, M. N. (2013). "Robust and energy-efficient asynchronous dynamic pipelines for ultra-low-voltage operation using adaptive keeper control," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, (Beijing), 267–272. doi: 10.1109/islped.2013.6629307

Cheng, H. P., Wei, W., Chunpeng, W., Sicheng, L., Hai, H. L., and Yiran, C. (2017). "Understanding the Design of IBM neurosynaptic system and its tradeoffs: a user perspective," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Vol. 2017, (Lausanne), 139–144. doi: 10.23919/date.2017.7926972

Coucke, A., Mohammed, C., Thibault, G., David, L., Mathieu, P., and Thibaut, L. (2019). "Efficient keyword spotting using dilated convolutions and gating," in *Proceedings of the ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Brighton), doi: 10.1109/icassp.2019.8683474

Courbariaux, M., Yoshua, B., and Jean-Pierre, D. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv [Preprint]*.

Davies, M., Narayan, S., Tsung-Han, L., Gautham, C., Yongqiang, C., Sri Harsha, C., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/mm.2018.112130359

Giraldo, J. S. P., and Marian, V. (2018). "Laika: A 5uW Programmable LSTM accelerator for always-on keyword spotting in 65nm CMOS," in *Proceedings of the ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, (Dresden), doi: 10.1109/esscirc.2018.8494342

Guo, R., Yonggang, L., Shixuan, Z., Ssu-Yen, W., Peng, O., Win-San, K., et al. (2019). "A 5.1pJ/Neuron 127.3us/Inference RNN-based speech recognition processor using 16 computing-in-memory SRAM Macros in 65nm CMOS," in *Proceedings of the 2019 Symposium on VLSI Circuits*, (Kyoto), C120–C121. doi: 10.23919/vlsic.2019.8778028

Kim, S. J., Soo, B. C., and Mingoo, S. (2021). A High PSRR, low ripple, temperature-compensated, 10-μA-Class Digital LDO Based on current-source power-FETs for a Sub-mW SoC. *IEEE Solid-State Circuits Letters* 4, 88–91. doi: 10.1109/LSSC.2021.3070556

Knag, P., Jung, K. K., Thomas, C., and Zhengya, Z. (2015). A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding. *IEEE J. Solid State Circ.* 50, 1070–1079. doi: 10.1109/jssc.2014.2386892

Koo, M., Srinivasan, G., Shim, Y., and Roy, K. (2020). "sBSNN: stochastic-bits enabled binary spiking neural network with on-chip learning for energy efficient neuromorphic computing at the edge. *IEEE Trans. Circ. Syst. I Regular Papers* 67, 2546–2555. doi: 10.1109/TCSI.2020.2979826

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceed. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Liu, J., Steven, M. N., and Mingoo, S. (2013). "Soft MOUSETRAP: a bundled-data asynchronous pipeline scheme tolerant to random variations at ultra-low supply voltages," in *Proceedings of the 2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, (Santa Monica, CA), doi: 10.1109/async.2013.29

Painkras, E., Luis, A. P., Jim, G., Steve, T., Francesco, G., Cameron, P., et al. (2013). SpiNNaker: A 1-W 18-Core System-on-chip for massively-parallel neural network simulation. *IEEE J. Solid State Circ.* 48, 1943–1953. doi: 10.1109/jssc.2013.2259038

Park, J., Juyun, L., and Dongsuk, J. (2019). "7.6 A 65nm 236.5nJ/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *Proceedings of the 2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, (San Francisco, CA), 140–142. doi: 10.1109/isscc.2019.8662398

Pei, J., Lei, D., Sen, S., Mingguo, Z., Youhui, Z., Shuang, W., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Seo, J. S., Bernard, B., Yong, L., Benjamin, D. P., Steven, K. E., Robert, K. M., et al. (2011). "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Proceedings of the 2011 IEEE Custom Integrated Circuits Conference (CICC)*, (San Jose, CA), doi: 10.1109/cicc.2011.6055293

Shan, W., Minhao, Y., Jiaming, X., Yicheng, L., Shuai, Z., Tao, W., et al. (2020). "14.1 A 510nW 0.41V low-memory low-computation keyword-spotting chip using serial FFT-Based MFCC and binarized depthwise separable convolutional neural network in 28nm CMOS," in *Proceedings of the 2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, (San Francisco, CA), 230–232. doi: 10.1109/isscc19947.2020.9063000

Wang, D., Pavan, K. C., Sung, J. K., Minhao, Y., Joao, P. C., Joonsung, K., et al. (2020). "Always-On, Sub-300-nW, event-driven spiking neural network based on spike-driven clock-generation and clock- and power-gating for an ultra-low-power intelligent device," in *Proceedings of the IEEE Asian Solid-State Circuits Conference (A-SSCC)*, (Hiroshima), doi: 10.1109/a-sscc48613.2020.9336139

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv [Preprint]*.

Yang, M., Chung-Heng, Y., Yiyin, Z., Joao, P. C., Aurel, A. L., and Mingoo, S. (2019). Design of an always-on deep neural network-based 1-μW voice activity detector aided with a customized software model for analog feature extraction. *IEEE J. Solid State Circ.* 54, 1764–1777. doi: 10.1109/jssc.2019.2894360

Yang, M., Hongjie, L., Weiwei, S., Jun, Z., Ilya, K., Sang, J. K., et al. (2021a). Nanowatt acoustic inference sensing exploiting nonlinear analog feature extraction. *IEEE J. Solid State Circ.* 1–11. doi: 10.1109/JSSC.2021.3076344

Yang, M., Shih-Chii, L., and Tobi, D. (2015). A Dynamic Vision Sensor With 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding. *IEEE J. Solid State Circ.* 50, 2149–2160. doi: 10.1109/jssc.2015.2425886

Yang, S., Jiang, W., Nan, Z., Bin, D., Yanwei, P., and Mostafa, R. A. (2021b). CerebelluMorphic: large-scale neuromorphic model and architecture for supervised motor learning. *IEEE Trans. Neural Netw.* 1–15. doi: 10.1109/TNNLS.2021.3057070 [Epub ahead of print].

Yang, S., Jiang, W., Xinyu, H., Huiyan, L., Xile, W., Bin, D., et al. (2021c). BiCoSS: toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw.* 1–15. doi: 10.1109/TNNLS.2020.3045492 [Epub ahead of print].

Yang, S., Tian, G., Jiang, W., Bin, D., Benjamin, L., and Bernabe, L. B. (2021d). Efficient spike-driven learning with dendritic event-based processing. *Front. Neurosci.* 15:97. doi: 10.3389/fnins.2021.601109

Conflict of Interest: JK, SJ, and SK are employed by Samsung Electronics.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 03 August 2021
doi: 10.3389/fncom.2021.674154

Accelerating Inference of Convolutional Neural Networks Using In-memory Computing

Martino Dazzi[1,2]*, *Abu Sebastian*[1], *Luca Benini*[2] *and Evangelos Eleftheriou*[1]

[1] IBM Research Europe, Rüschlikon, Zurich, Switzerland, [2] Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland

In-memory computing (IMC) is a non-von Neumann paradigm that has recently established itself as a promising approach for energy-efficient, high throughput hardware for deep learning applications. One prominent application of IMC is that of performing matrix-vector multiplication in $\mathcal{O}(1)$ time complexity by mapping the synaptic weights of a neural-network layer to the devices of an IMC core. However, because of the significantly different pattern of execution compared to previous computational paradigms, IMC requires a rethinking of the architectural design choices made when designing deep-learning hardware. In this work, we focus on application-specific, IMC hardware for inference of Convolution Neural Networks (CNNs), and provide methodologies for implementing the various architectural components of the IMC core. Specifically, we present methods for mapping synaptic weights and activations on the memory structures and give evidence of the various trade-offs therein, such as the one between on-chip memory requirements and execution latency. Lastly, we show how to employ these methods to implement a pipelined dataflow that offers throughput and latency beyond state-of-the-art for image classification tasks.

Keywords: convolutional neural network, in-memory computing, computational memory, AI hardware, neural network acceleration

OPEN ACCESS

Edited by:
Oliver Rhodes,
The University of Manchester,
United Kingdom

Reviewed by:
Shimeng Yu,
Georgia Institute of Technology,
United States
Rishad Shafik,
Newcastle University, United Kingdom

***Correspondence:**
Martino Dazzi
daz@zurich.ibm.com

Received: *28 February 2021*
Accepted: *23 June 2021*
Published: *03 August 2021*

Citation:
Dazzi M, Sebastian A, Benini L and Eleftheriou E (2021) Accelerating Inference of Convolutional Neural Networks Using In-memory Computing. Front. Comput. Neurosci. 15:674154. doi: 10.3389/fncom.2021.674154

1. INTRODUCTION

In recent years, Deep Neural Networks (DNNs) have revolutionized the field of Machine Learning by reaching unprecedented accuracy in a large number of cognitive data analysis tasks. DNNs are currently being used in a wide variety of applications, ranging from image classification (He et al., 2016) to autonomous driving (Bojarski et al., 2016) and natural language interpretation (Vaswani et al., 2017). As the applications increase in number and complexity, so do DNN architectures, and along with them the hardware architectures for their execution and training.

Historically, DNNs run on general purpose processors, such as CPUs and GPUs. While this solution is still widely employed and GPUs are constantly improving their metrics of energy consumption and execution time for training and inferencing, they are inadequate for application areas with power envelopes of sub Watt or of very few Watts, which are generally referred to as the IoT or edge computing realm. To this end, custom hardware platforms such as application-specific integrated circuits (ASICs) are being designed (Chen et al., 2016) for low power and efficient execution of DNNs. ASICs are quite energy efficient in terms of TOPS/W and can reach state-of-the art accuracy and throughput in a variety of tasks (Jouppi et al., 2017), albeit at the expense

of time consuming circuit design and, to a certain extent, limited scope of execution. Moreover, a hardware-aware training of the DNNs (Han et al., 2015; He et al., 2017; Jacob et al., 2018) is often needed.

Independently from the hardware platform, be it general purpose processors or ASICs, different performance can be obtained on the basis of the computational paradigm being used. In general, von Neumann architectures are inherently limited in performance by the need to move data from the memory to the computational units: in modern DNN models, with a parameter count that can reach hundreds of millions (Huang et al., 2017; Vaswani et al., 2017) retrieving them from a memory can severely hinder performance. This phenomenon, also known as the von Neumann bottleneck, has led to many research efforts aiming at alternative, non-von Neumann paradigms. Among these, we look in depth into IMC (Prezioso et al., 2015; Burr et al., 2017; Hu et al., 2018; Ielmini and Wong, 2018; Le Gallo et al., 2018; Xia and Yang, 2019; Sebastian et al., 2020), a computational paradigm showing promise for unprecedented performance and energy efficiency, targeting specifically the main computational load of DNNs: matrix-vector multiplications. With IMC, we take advantage of a set of resistance-based or charge-based memory devices, such as memristive (Sebastian et al., 2019; Joshi et al., 2020) or CMOS-based devices (Valavi et al., 2019). By organizing these devices in a crossbar array configuration, based on their physical properties, a matrix-vector multiplication can be carried out with $\mathcal{O}(1)$ time complexity, contrary to the $\mathcal{O}(N^2)$ time complexity of this operation on traditional architectures. However, in order to fully exploit this new computing paradigm, in the design of IMC-based hardware we must rethink well-established architectural choices and provide novel methodologies for optimizing the dataflow. Specifically, while the number and size of synaptic weights can vary greatly within the layers of a DNN, the IMC crossbar arrays on which they are mapped have pre-determined, fixed shapes. Consequently, the mapping of synaptic weights is a pivotal problem to optimize in order to fully exploit the potential of IMC in DNN applications.

Moreover, while IMC obviates the need to communicate synaptic weights, the intermediate results must be cached on the local memories of the IMC cores. Also in this case, new approaches must be developed for handling the data efficiently and according to the dataflow.

In this work, we focus on the problem of executing image classification tasks on IMC-based hardware architectures. Specifically, we focus on Convolutional Neural Networks (CNNs), which represent the state-of-the-art for a variety of image processing applications. In section 2, we propose a novel IMC core architecture for inference of CNNs. Moreover, we present novel methods for mapping weights on the IMC crossbar array and activations on the local memory of the IMC core. These methods enable high-throughput, efficient execution of CNNs on the IMC hardware. Note that while these contributions are presented as different parts that organically belong to a IMC-based accelerator for inference of CNNs, the methodologies and approaches developed here, have universal applicability regardless of the overall architectural configuration. In section 3, we introduce the dataflow of the IMC-based accelerator and present as a case study the execution of ResNet-32 on the CIFAR-10 dataset. This section gives evidence of how our proposed methodologies applied to IMC yield beyond state-of-the-art performance compared to non-IMC ASICs targeting the same dataset. Lastly, section 4, compares our approach with previous works in the field and concludes the paper.

2. METHODS

2.1. Hardware Architecture Overview

We identify the In-Memory Computing core (IMC core) as the unit building block of the IMC hardware architecture. The IMC core is built around a crossbar array executing the matrix-vector multiplication and features peripheral circuitry for the additional functionality required by CNNs.

Figure 1 shows the mapping of the CNN layers on the IMC cores and the tasks carried out during execution by each component. We start by describing the operation required by CNNs in order to understand how these are employed on the hardware architecture and how the hardware itself is designed in order to facilitate their execution.

A CNN architecture is typically comprised of a series of interconnected layers, each layer defined by learnable parameters and typically one activation function. The result of the activation function is referred to as activation. As the name suggests, every layer of a CNN performs a convolution on the activations. A convolutional layer transforms an input volume of feature maps of size $H_{in} \cdot W_{in} \cdot C_{in}$ into an output volume of feature maps of size $H_{out} \cdot W_{out} \cdot C_{out}$. Unless specified, we will consider in our study input and output volumes with $H_{in} = H_{out} = H$ and $W_{in} = W_{out} = W$. Note that these assumptions do not affect generality, since for the layers in which pooling, striding, and padding are involved, the approach we follow in designing the various architectural choices would remain the same. Also, we will refer to the activations along the plane marked by H and W as lying on the (H, W) plane.

Each convolutional layer consists of many dot products. Specifically, every element of the output volume in the (H, W) plane is calculated as the dot product between a patch of the input volume of size $F_1 \cdot F_2 \cdot C_{in}$ with an equally sized matrix of parameters of that layer, also called kernel weights. The dot product is executed C_{out} times by applying C_{out} different kernel weights in order to obtain all C_{out} output channels of the output volume. The results of the matrix-vector multiplication are called pre-activations. In matrix form, the matrix-vector multiplication for one pre-activation at layer $L + 1$ across all channels at a generic position (u, v) is expressed by Equation (1), where $\alpha_{i,j}^{n,L}$ indicates an activation at layer L and position *(i, j)* at channel n, $\gamma_{i,j}^{n,L}$ denotes a pre-activation layer L, position (i, j) and channel n, and $\boldsymbol{K}$ represents the kernel matrix. One element of the kernel matrix for a position *(a,b)*, input channel v, and output channel w

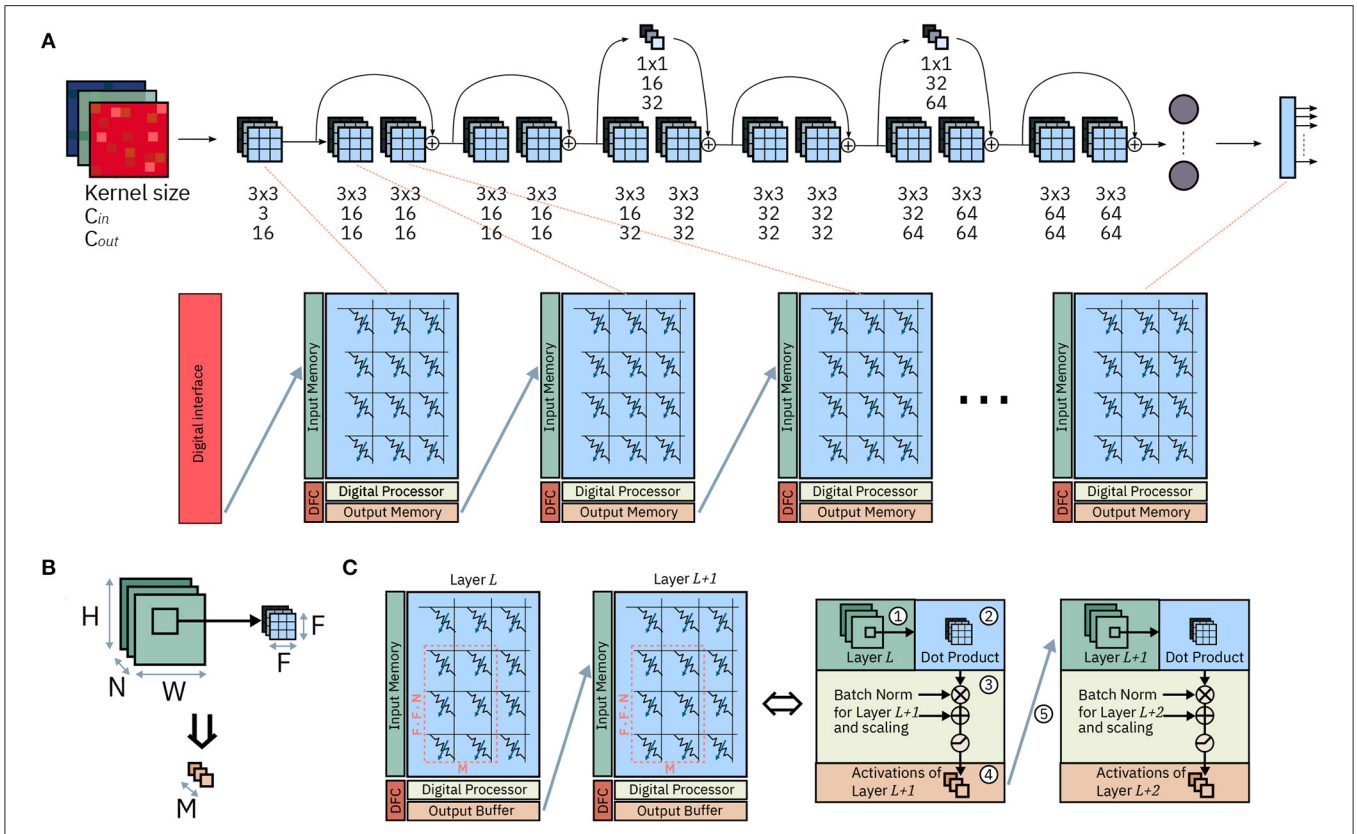


FIGURE 1 | Mapping of ResNet-32 on an array of IMC cores. **(A)** Mapping of the network on the array: each layer is mapped onto one IMC core. The synaptic weights are mapped on the computational memory elements, while operations on pre-activations or activations are carried out by the Digital Processor. **(B)** One matrix-vector multiplication for a convolutional layer. **(C)** Dataflow between two IMC cores; the dataflow is scheduled by the Dataflow Controller. The system level representation (left) and, color-coded, the corresponding functionality of each component (right).

is indicated as $k_{a,b}^{v,w}$. We consider $C_{in} = N$, $C_{out} = M$, and $F_1 = F_2 = F$.

$$\gamma_{u,v}^{L+1} = \boldsymbol{\alpha}_{i:i+F-1,j:j+F-1}^{L} \cdot \boldsymbol{K} = \begin{bmatrix} \alpha_{i,j}^{0,L} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N/2-1,L} \\ \alpha_{i,j}^{N/2,L} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N-1,L} \end{bmatrix}^{T} \begin{bmatrix} k_{0,0}^{0,0} & \cdots & k_{0,0}^{0,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N/2-1,0} & \cdots & k_{F-1,F-1}^{N/2-1,M} \\ k_{0,0}^{N/2,0} & \cdots & k_{0,0}^{N/2,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N,0} & \cdots & k_{F-1,F-1}^{N,M} \end{bmatrix} \quad (1)$$

Subsequently to the operation on the kernel weights, the resulting pre-activations undergo a post processing that typically involves additive and multiplicative scalings, known as batch normalization (Ioffe and Szegedy, 2015) and a non-linear activation function. Given the operations described above, the matrix-vector multiplication that transforms activations from one layer into pre-activations of another is particularly suited for IMC, while the subsequent operations on pre-activations are better suited for standard digital processing units.

Figure 1A shows the mapping of a CNN, ResNet-32, on an array on IMC cores. The network is mapped on the physical array so that at most one layer is mapped on each IMC core. In the cases in which the size of the layer is greater than that of the IMC core, the synaptic weights of the layer are split between multiple IMC cores. The topology of the network is mapped on the IMC cores, implemented as crossbar arrays, so that edges of the dataflow graph of the CNN correspond to communication channels in the physical array. While the design of the communication fabric for such an architecture is not the subject of this work, we will assume that any IMC core can communicate in one timestep to any other IMC core with which communication is required, as presented in Dazzi et al. (2019). **Figure 1B** shows figuratively the matrix-vector multiplication of Equation 1 for a convolutional layer where $F_1 = F_2 = F$. A subset of the input volume (in green) is multiplied by the kernel weights (in blue); after batch normalization and execution of the activation functions, the results are the output activations across all output channels C_{out} and one position on the H, W dimensions (in orange). **Figure 1C** shows the hardware architecture of two successive IMC cores and, color coded, the role of every component in the execution of the CNN. The IMC core consists of the Crossbar Array, an

Input Memory, a Digital Processor, an Output Memory and a Dataflow Controller.

The Input Memory stores the input activations of one layer, which during execution are either received from an input scratchpad or from another IMC core executing another layer. When needed for computational purposes, the activations are read from the Input Memory and provided to the interface of crossbar array ①. The crossbar array maps the kernel weights of one layer to its IMC devices and executes the matrix-vector multiplication between a patch of the input volume and the kernel weights themselves expressed in Equation (1) ②. The result of this computation are C_{out} pre-activations, i.e., the results of one matrix-vector multiplication before any post-processing and application of the non-linear activation function. The Digital Processor executes the remainder of the computation that transforms pre-activations into activations ③. Namely, it takes care of all computations in CNNs not adequate for IMC. In our example, these include batch normalization and the application of activation functions. Lastly, the output memory collects the output activations that are the result of the computation ④; These results are then delivered, via communication links, to the next layers (cores) for further processing ⑤.

Figure 2 shows the detailed dataflow of two IMC cores executing two subsequent layers *L* and *L+1* and the dataflow chart for the IMC core executing layer *L*. In the example, we assume a 3x3 kernel and no padding. Firstly, rows of input pixels to layer *L* are fetched from the Input Memory and loaded to the Crossbar Array ①. In this phase, the pixels are fetched from the local SRAM and loaded on the local buffers of the Crossbar Array, which temporarily store pixels for the subsequent computation to be executed. The operation is repeated for timesteps $t_0 - 2$ (red row of pixels), $t_0 - 1$ (yellow row of pixels), and t_0 (green row of pixels), after which enough data for one dot product has been loaded to the buffers of the Crossbar Array. Consequently, the computation in the crossbar array ② can take place at timestep t_0. Finally, timestep t_0 concludes with the delivery of the pixel that was computed to the local memory of the subsequent core ③. In can be seen how, aside from the initial loading of pixels at timesteps $t_0 - 2$ and $t_0 - 1$, the IMC core assigned to layer *L* requires a new row of pixels to be fetched for computation. Indeed, at timestep $t_0 + 1$, one new row of pixels (in blue) is loaded, followed by computation of one dot product and delivery of the pixel to the IMC core assigned to layer *L+1*.

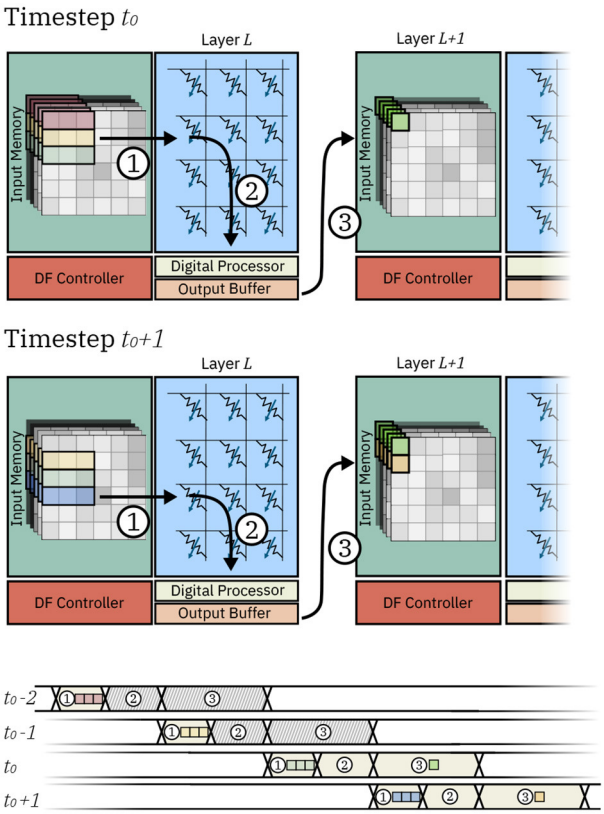


FIGURE 2 | Data movement in two IMC cores at two consecutive timesteps t_0 and $t_0 + 1$. Execution steps are represented as (1) fetching and loading of the data, (2) computation, and (3) communication of the pixel to other IMC cores. The timechart shows the execution steps for the IMC core executing layer *L*. At timesteps $t_0 - 2$ and $t_0 - 1$ input pixels are loaded on the crossbar array. As data is not yet sufficient for the computation of any pixel, no computation or communication takes place. At timestep t_0, sufficient pixels are available to compute on the crossbar array and one input pixel is computed and communicated to other IMC cores. Likewise at timestep $t_0 + 1$, a new row of pixels is loaded and the IMC core executed the computation and communication of another output pixel.

2.2. Mapping of Weights

In CNNs, the synaptic weights, also referred to as kernel weights and responsible for the matrix-vector multiplications, comprise a convolutional layer. In this section, we will refer to the overall matrix of kernel weights of size $F_1 \cdot F_2 \cdot C_{in} \times C_{out}$ as the kernel matrix. Also, for the sake of brevity, we will refer to the activations for one position in the (H, W) plane across all channels as one *pixel*. **Figure 3** shows the mapping of one kernel matrix on a crossbar array. Without loss of generality, it is assumed that one kernel weight is mappable on one computational memory device. Note that this is a reasonable assumption given the common precision requirements of weights in DNNs (Joshi et al., 2020) and the effective precision offered by the memory devices used for IMC. Our approach can however be easily extended to cases where a single weight is mapped on multiple devices. According to this mapping approach, the kernel matrix is unrolled so that the kernel weights associated to one output channel are placed along one column of the crossbar. In this way, the C_{out} columns of the kernel matrix are mapped on the same amount of adjacent columns of the crossbar array. Based on this physical placement, one matrix-vector multiplication is performed as follows: first, the input data-patch, retrieved from the Input Memory, is unrolled and provided at the input rows of the crossbar, matching the corresponding rows of the kernel matrix. Then, from each column, the pre-activation for a single channel is read back and converted to a digital value. The pre-activations will subsequently undergo digital processing, which in principle implements the non-linear function and any other digital processing that is required.

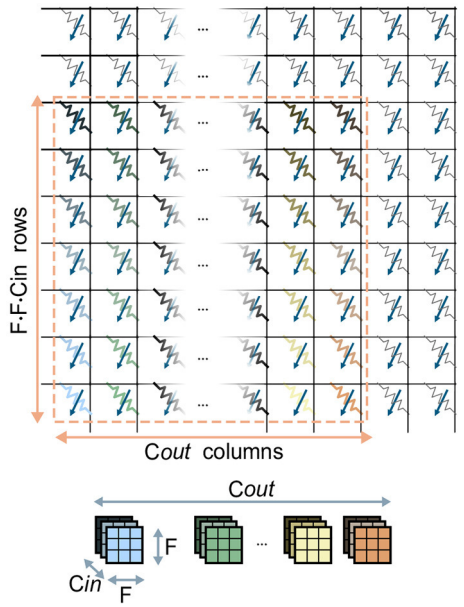


FIGURE 3 | Mapping of one kernel matrix of size $F \cdot F \cdot C_{in} \cdot C_{out}$ on a crossbar array. The kernel weights **(top)** are color coded to reflect their placement on the devices **(bottom)**. The $F \cdot F \cdot C_{in}$ kernel weights responsible for one of the C_{out} output channels are mapped on one column of the array.

Fundamentally, the problem of mapping a kernel matrix to a crossbar array constitutes the mapping of a shape of $(F_1 \cdot F_2 \cdot C_{in}) \times C_{out}$ elements onto a grid of fixed size composed of computational memory elements. Although the F_1, F_2, C_{in}, C_{out} parameters change from layer to layer and from network to network, state-of-the-art CNNs for image classification (He et al., 2016; Huang et al., 2017) typically feature kernels of size $F_1 = F_2 = 3$, and channel sizes in powers of two, commonly ranging from 16 to 1024. Furthermore, usually $C_{in} = C_{out}$ for the majority of the internal layer of the CNNs (Krizhevsky et al., 2012; He et al., 2016). In light of this assessment, we will focus on the case where $F_1 = F_2 = 3$ and $C_{in} = C_{out}$, as this is the set of hyperparameters which is the most common within the architectures of the CNNs we take into consideration. Regarding the dimensionality of the crossbar arrays, it appears that practical implementations adopt square sizes, primarily for generality and for the possibility of performing the reverse read operation in the case the array was designed also for training DNNs (Nandakumar et al., 2018). Based on these considerations, the kernel matrices typically appear to have an aspect ratio equal to $F_1 \cdot F_2 = 9$ (i.e., they require 9 times more rows than columns), and are mapped on a crossbar array grid with aspect ratio equal to one. This implies that a large number of the crossbar array devices will end up being unmapped, and therefore unutilized, during the execution of CNNs. We try to avoid this issue by proposing two methods of parallelization of the computation of the convolutional layers by replication of the kernel matrix. In these methods, we map multiple replicas of the kernel matrix on the crossbar array for one convolutional layer in order to execute in parallel multiple matrix-vector multiplications of the same layer. Note that the number of kernel replicas mapped on the crossbar array effectively represent the number of dot products that can be executed in parallel on the array. Consequently, the number of kernel replicas is equivalent to the degree of parallelism of the dot products.

2.2.1. Parallelizing Computation Across One Direction

In this method, we parallelize the computation so that the outputs are pre-activations along one direction of the (H, W) plane of the output volume. **Figure 4** shows the parallelization of three matrix-vector multiplications for a single convolutional layer with a kernel of size $F_1 = F_2 = 3$.

Figure 4A shows the input and output feature map volumes. In the input volume, we highlight the input patches required for each of the matrix-vector multiplications we parallelize and in the output volume we highlight the positions of the resulting pre-activations. All the activations from the various input patches will be provided in parallel to the rows of a crossbar array, and consequently all the pre-activations of the output volume will be calculated in parallel on the columns of the crossbar array. **Figure 4B** shows the single input patches of the input volume, colored and patterned, and the corresponding pre-activations on the output volume with the same kernel matrix. It can be noted in **Figure 4A** that, while each of the three input patches comprise $27 \cdot C_{in}$ activations, they overlap and overall comprise $(5 \times 3) \cdot C_{in}$ unique activations. **Figure 4C** shows the mapping of the kernel replicas on the crossbar array. Three replicas of the kernel matrix are mapped on the crossbar, each kernel matrix by itself occupying an area of $F_1 \cdot F_2 \cdot C_{in}$ rows and C_{out} columns. Different color patterns represent input activations and output pre-activations for different matrix-vector multiplications on the same input volume in **Figure 4A**. Note that, because of the overlapping in the input patches, some input activations belong to more than one input patch. Specifically, in **Figure 4C**, input activations with overlapping color patterns represent activations of the input volume that belong to various input patches. Consequently, as the input pixels for different matrix-vector multiplications are shared, some rows of the crossbar are shared between different kernels. Specifically, assuming $F_1 = F_2 = F$, while one kernel matrix occupies a $(F \cdot F \cdot C_{in}) \times C_{out}$ area on the crossbar, any additional kernel matrix adds $F \cdot C_{in}$ rows and C_{out} columns.

2.2.2. Parallelizing Computation Across Two Directions

In this method, we parallelize the computation so that the output are pre-activations along on both directions of the (H, W) plane of the output volume. In **Figure 5A**, we color-code and pattern the input patches for the parallelized matrix-vector multiplications on the feature maps. We parallelize overall six matrix-vector multiplications, with the input patches covering collectively a 5-by-4 area on the (H, W) plane of the input volume. **Figure 5B** shows the physical mapping of the kernel replicas on the crossbar and the position of the activations provided as inputs. It can be noted that, contrary to the previous

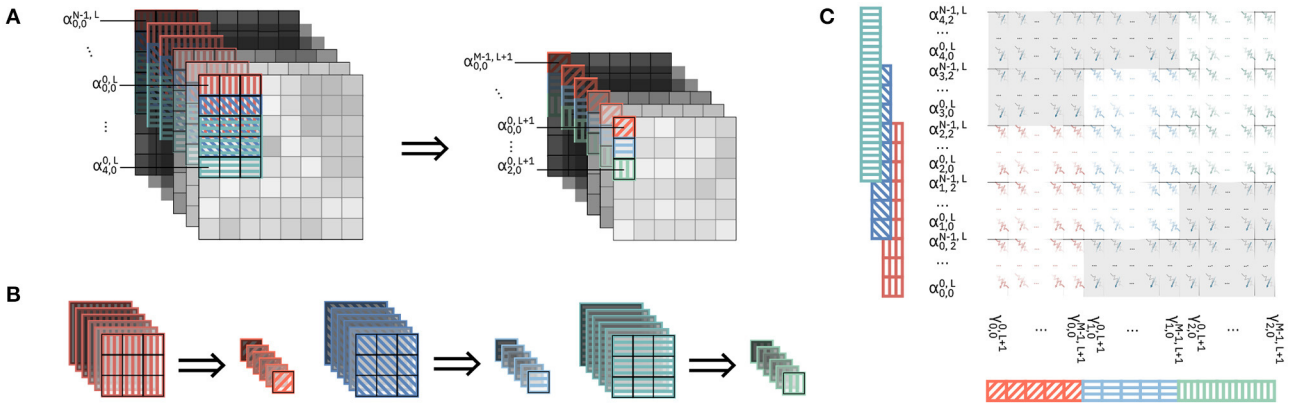


FIGURE 4 | Parallelization of the computation along one direction through mapping of kernel matrix replicas on the crossbar array. In this example, we display the parallelization of three matrix-vector multiplications of the convolutional layer. Devices outside the colored patches are unmapped. **(A)** Input volume (left) and output volume (right) of feature maps for a given layer, where three matrix-vector multiplications are executed in parallel. The patches required for every matrix-vector multiplications are highlighted in the input volume and color-patterned. The resulting output activations in the output volume are color-patterned in a matching color. **(B)** Correspondence between the input patches in **(A)** and the resulting output activation. **(C)** Mapping of the kernel matrix replicas for the three matrix-vector multiplications on the crossbar array. The color of the devices corresponds to the similarly colored input and output activations in **(B)**. Input activations are provided to the crossbar array in one timestep in an order that is highlighted by the same color pattern as in **(A,B)**.

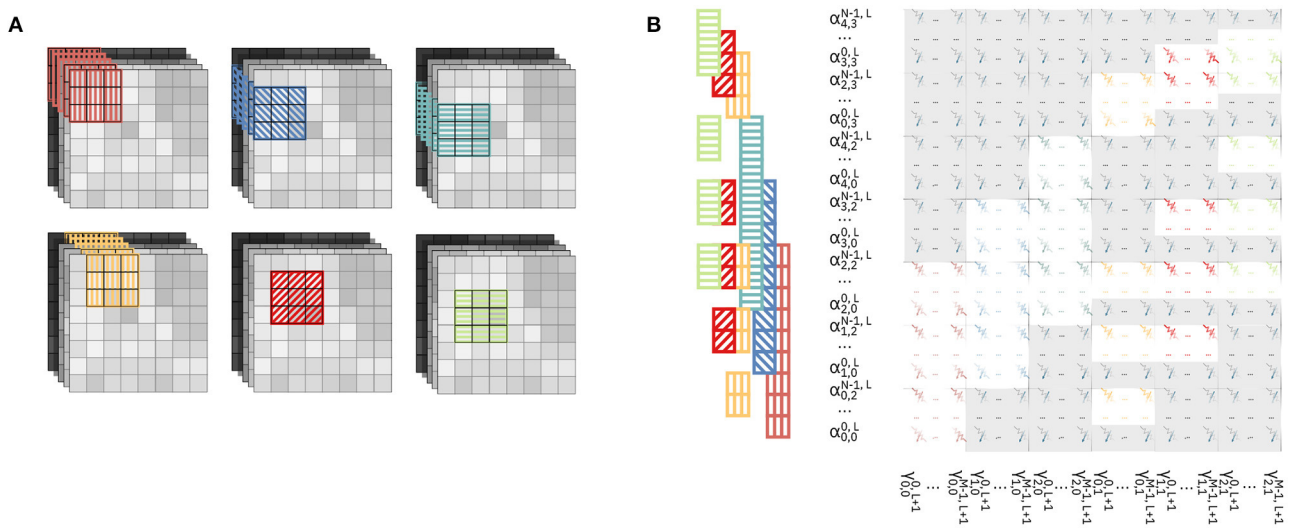


FIGURE 5 | Parallelization of the computation along two directions by mapping of kernel matrix replicas on the crossbar array. **(A)** Color-patterned position of the six input patches for the six matrix-vector multiplications executed in parallel on the crossbar. **(B)** Mapping of the kernel matrix replicas on the crossbar array. Input activations are provided to the crossbar array in an order that is highlighted by the same color pattern as in **(A)**.

method, the sharing of rows among different kernels is more scattered, and generates a less regular pattern of mapping of both activations to the rows of the crossbar and kernel weights to the devices. With this method, for every replica of the kernel matrix mapped onto the crossbar, C_{out} columns are assigned to the new output to be computed, while the number of additional rows depends on the position of the patch on which the kernel is supposed to perform the computation. In particular, based on the position, the activations required by one kernel replica may or may not have been already in use by rows in the crossbar. For example, the kernel associated with the yellow patch, i.e., the one with a vertical stripe pattern in **Figure 5A**, requires $F \cdot C_{in}$ rows to be assigned. However, the kernels associated with the red diagonal stripe patterned and light green horizontal stripe patterned patches only require C_{in} new rows to be assigned for each.

2.2.3. Comparison of Parallelization Methods

Given the many variables in the mapping of kernel matrices to crossbar arrays (e.g., crossbar size, kernel size, and number of input and output channels that might change even within the same CNN), a reasonable *figure of merit* of a mapping scheme

should be based on the aspect ratio of the kernel matrices that are mapped on the crossbar. Indeed, assuming crossbars of aspect ratio equal to 1 and given a number of kernel matrices to be mapped on that crossbar, then the greatest number of kernel matrices that can fit on the crossbar is achieved by a method that makes the aspect ratio of the shape of the overall replicas of the kernel matrix closer to 1. In this section, we will refer to the method proposed in section 2.2.1 as Method 1, and the method proposed in section 2.2.2 as Method 2. Also, we will refer to the overall matrix comprising a number of kernel matrix replicas to be mapped onto one crossbar as the collective kernel matrix. In Method 1, the aspect ratio of n kernel replicas (collective kernel matrix) mapped on the crossbar can be expressed in closed form and, for $F_1 = F_2 = F$, is equal to

$$A(n) = F^2 \cdot \frac{C_{in}}{C_{out}} \cdot \frac{1 + (n-1) \cdot \frac{1}{F}}{1 + (n-1)} \tag{2}$$

It can readily be seen that for $C_{in} = C_{out}$, then $A(1) = F^2$, and $A(\infty) = F$. Thus, as the number of kernel matrix replicas increases, the aspect ratio of the collective kernel matrix decreases. Specifically, from a quadratical dependency when $n = 1$, we reach asymptotically a linear dependency.

For Method 2, given that the way in which the number of new rows to be assigned for every replica strongly varies from the position of the input volume patch that the kernel takes as input, we did not derive a closed formula expression. We note however that as Method 2 parallelizes computations across both directions of the (H, W) plane of the input volume, the theoretical minimum aspect ratio is:

$$A(n = H_{out} \cdot W_{out}) = \frac{C_{in} \cdot H_{in} \cdot W_{in}}{C_{out} \cdot H_{out} \cdot W_{out}} \tag{3}$$

which, for layers with $C_{in} = C_{out}$, $H_{in} = H_{out}$, $W_{in} = W_{out}$ and stride equal to 1, is equal to 1.

Figure 6 shows at comparison of the two methods for different parameters. In this comparison we consider $C_{in} = C_{out} = 16$ and $F_1 = F_2 = F = 3$, which are realistic parameters for CNNs targeting IoT-like datasets. We perform the comparison in the following way: given an output volume, we fix a maximum number of pixels on the width direction W_{out} in its (H, W) plane (In **Figure 6** it is called ΔW). We then parallelize the computations in order to obtain adjacent pixels with maximum width equal to ΔW. By comparing the two methods in this way, we observe that the case with $\Delta W = 1$ corresponds to Method 1, while any other value of ΔW corresponds to Method 2. **Figure 6A** shows a comparison of the aspect ratios of the collective kernel matrix as a function of the number of kernel replicas n. As foreseen by Equation 2, the minimum aspect ratio reachable with $\Delta W = 1$ is 3. Method 2, with $(\Delta W \geq 2)$ is able to further reduce the aspect ratio, and while the theoretical minimum is equal to 1, it tends to decrease very little at around $A = 2$. Also, it is apparent that for $\Delta W \geq 3$ the parallelization seems to perform similarly. **Figure 6B** shows the number of rows required for a given ΔW, as function of the number n of kernel replicas. For a given number of replicas,

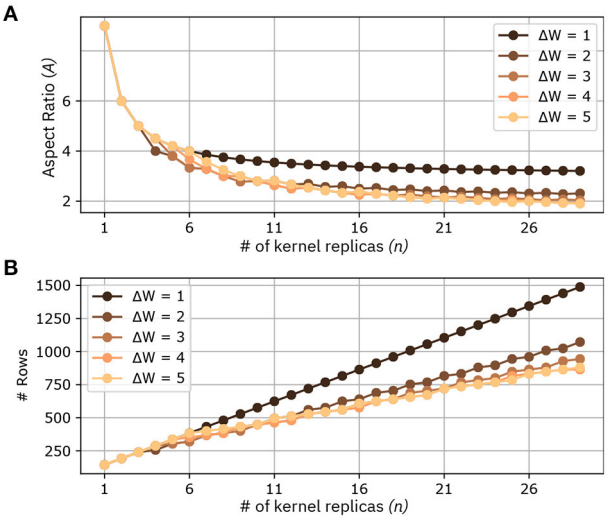


FIGURE 6 | Comparison between different mappings of kernel replicas at the increase of the number of kernels n. Different methods are marked by different ΔW. **(A)** Aspect ratio of the collective kernel matrix (# crossbar rows/# crossbar columns) for a given number of kernel replicas. **(B)** Number of Rows at the increase of the number of kernel replicas.

Method 2, with $\Delta W \geq 2$ clearly requires less rows compared to Method 1. For example, note that a collective kernel matrix of 20 kernels requires 1,056 rows with Method 1, and only 672 with Method 2 and $\Delta W = 5$. Lastly, we discuss the benefits of our methodologies on device utilization. Mathematically, device utilization u can be formulated as:

$$u = \frac{\sum_{i=0}^{N-1} D_K}{D_A}, \tag{4}$$

where D_K denotes the number of devices required for one kernel matrix, N represents the number of kernels, and D_A indicates the number of devices of the crossbar array. As our proposed mapping methods maximize N, they also enable higher device utilization, which increases linearly with N. However, note that u is not a function of ΔW. Consequently, for a given number N of kernel matrices that are mapped on the array, the device utilization is the same, regardless of the ΔW used in the mapping.

2.3. Mapping of Activations

As presented in section 2.1, the activations for one layer are stored on the Input Memory of the IMC core. The purpose of the Input Memory is to cache the pixels, which will serve as the input vectors for the matrix-vector multiplications that will be executed on the crossbar array. We start by describing the pattern by which pixels need to be stored and fetched from the input memory. At every timestep, if the conditions for computation are met, each IMC core must fetch a given number of pixels and provide them to the crossbar array interface for computation of the matrix-vector multiplications. Further, the IMC core may have to store the pixels that it receives from other IMC cores. We assume that the Input Memory is a static random-access memory (SRAM),

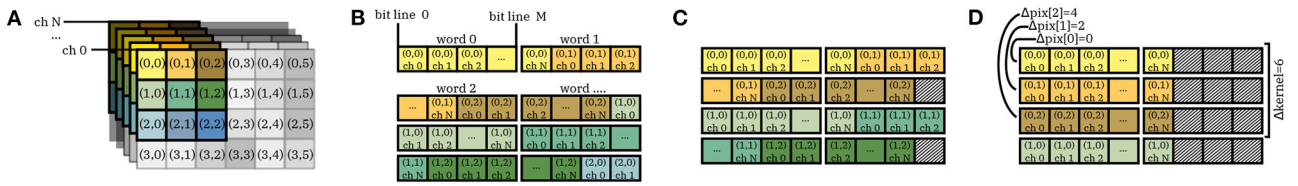


FIGURE 7 | Mapping of pixels on the Input Memory. In the memory mapping, colored-in boxes represent data, while diagonal patterned boxes represent empty bitlines. **(A)** Portion of a feature map volume, in which we highlight the pixels to be mapped on the Input Memory. **(B)** Inter-word adjacent placing of pixels. In the figure, we display the numbering of words and bit lines in the Input Memory. **(C)** Kernel-level interleaved placing. Pixels belonging to one kernel row are placed contiguously to each other [e.g., pixels (0, 0) to (0, 2))]; the remaining bit lines for the word containing the (0, 2) pixel as left empty. **(D)** Activation-level interleaved placing. No data from different pixels in stored in the same word. Note that $\Delta pix[i]$ indicates the index of word with the beginning of the $i-th$ pixel. $\Delta kernel$ denotes the overall number of words that store one row of pixels.

which is commonly used to the end of caching operands in ASICs (Shafiee et al., 2016).

In general, for the pipelined operation of layers with stride equal to 1, at every timestep a IMC core must fetch a number of pixels equal to the kernel size and store one pixel received from a previous layer (IMC core). We note the difference between the logical placement of pixels, which are part of the input volume of pixels as shown in **Figure 2**, and their physical instantiation in the local memory. Given that the fetching pattern is tied to the logical placement of the pixels in the input volume, in principle we would prefer to store pixels in the Input Memory in a way that mimics their logical position. Unfortunately, this task poses several problems. First, for a single network, different layers can typically have different channel depth, i.e., pixels for different layers are represented by a different number of bits. Moreover, the channel depth can vary from network to network. On the other hand, memory arrays such as on-chip SRAM, have fixed physical structures organized in words, each word comprising a given number of bits. The overall number of words is defined as the word depth. Below, we present three methods for mapping the varying and flexible data structure of pixels onto the fixed and predetermined memory array. We also present appropriate metrics for evaluating these methods.

2.3.1. Memory Mapping Evaluation Metrics

In standard on-chip memories, the granularity for fetching the data is the memory word. This means that in one clock cycle, the memory can be accessed one word at a time. The number of cycles required to store (i.e., write) and fetch (i.e., read) one pixel depends on the size of the memory word and of the pixel itself. Furthermore, in the case where different pixels are stored in a single word, the writing operation is possible through write-masks, which are a common feature in SRAMs. On the other hand, the reading operation will require bit-slicing logic to dissect the word and extract the logical units required. Although possible, both the reading and the writing of different data in the same word require the accounting of the bit-line index where one data ends and another begins.

In the following subsections, we use as a metric the number of cycles required to write one pixel and to read one kernel-row of pixels. Moreover, we consider the requirement of storing bit-line indexes for reading and writing data. Lastly, we assess the amount of memory needed to store the same number of data by the various methods.

2.3.2. Intra-Word Adjacent Placing (IWAP)

Using this method, we store pixels adjacent to each other in the memory so that different pixels could be stored on the same word. The placing mimics the logical organization of the data, so that adjacent pixels on the same row in the input volume are stored adjacent to each other on the Input Memory. **Figure 7B** shows inter-word adjacent mapping for a number of pixels highlighted in **Figure 7A**. Each pixel is mapped so that different channels are stored sequentially and according to their logical order in the feature map volume. If space is available within one word, different pixels can be stored in the same word; this is for example the case of pixel $(0, 0)$ and pixel $(0, 1)$ in word 1. For a given number of pixels to be cached, this method requires the minimum possible amount of memory. However, it requires bit-line indexes for both storing and fetching the pixels, and can require different number of cycles for writing and reading the pixels depending on the timestep.

2.3.3. Kernel-Level Interleaved Placing (KLIP)

Using this method, we store different pixels within the same word, only if they belong to a single row of an input patch for which matrix-vector multiplications must be computed. Assume that, in **Figure 7A**, the feature map volume is to be convolved with a 3x3 kernel. Among those highlighted in the figure, the rows of pixels belonging to input patches for the convolution are $[(0, 0), (0, 1), (0, 2)]$; $[(1, 0), (1, 1), (1, 2)]$; and $[(2, 0), (2, 1), (2, 2)]$. **Figure 7C** shows the mapping on the Input Memory. The pixels of each row are stored contiguously to each other in the memory. However, the remaining bit lines within the word containing the last pixel of the feature map volume are left empty. This practice of leaving part of the memory empty in order to allow the mapping of data in a way that is closer to its logical positioning is somewhat similar to the concept of memory interleaving typically employed in memory management for CPUs.

As this method leaves part of the memory empty to accommodate the positioning of pixels in memory, it clearly does not use the minimum amount of memory possible for one volume of pixels. As with the previous method, it may require different number of cycles for writing the pixels depending on the

timestep. Moreover, given that different pixels can still be stored within one word, bit line indexes are required for writing. As described in **Figure 2**, at every timestep where computations are executed, a new row of an input patch is loaded in the memory. Since we place contiguously the pixels of the rows of the input patches, the number of read cycles is constant at every timestep, and no bit line index is required for reading one row.

2.3.4. Pixel-Level Interleaved Placing (PLIP)

Using this method, we do not allow storing different pixels within the same word. **Figure 7D** shows one such memory mapping. In the example in **Figure 7A**, each pixel is stored in two memory words. The unoccupied part of the last word storing each pixel is left empty. This is very similar to the concept of kernel-level interleaved placing, this time applied to single pixels across the channel depth.

This method has the finest granularity of logical units to be stored, and thus makes the most inefficient use of the local memory. Since every pixel is individually interleaved, there is no need for bit line index for either writing or storing words. Also contrary to previous methods, both the number of read and write cycles are constant regardless of the pixels being read or stored.

2.3.5. Comparison of the Methodologies

In this subsection, we compare the various memory-mapping methodologies, discussed above, using the metrics introduced in section 2.3.1. Consider the storing of a portion of the feature map volume of size $H \cdot F \cdot N$, where H denotes the height of the volume, F the kernel size of that layer and N the channel depth. For the cases considered in this comparison, we use a fixed kernel size $F = 3$, whereas H and N are varying parameters. Furthermore, for this comparison we consider 8-bit precision for the pixels. As explained in section 2.3.1, at each cycle one pixel across all channels must be written on the Input Memory, while a number of pixels equal to one row of input patch must be fetched to execute the matrix-vector multiplication. Thus, in our comparison, write cycles refers to the number of cycles to write one pixel, while read cycles refers to the number of cycles to read $F = 3$ pixels.

Table 1 shows the results for various memory mappings. Firstly, we investigate the case of writing and reading the input image of a CNN, which typically comprises $N = 3$ channels. The size of the image is based on input images for the CIFAR-10 dataset, i.e., $H = 32$. Word length (WL) is set to 128 bits. It can be noted how inter-word adjacent placing (IWAP)requires the minimum amount of memory, but has a varying number of read and write cycles based on the pixels to be read. In this case, owing to the small size of the pixels in comparison to the word length, kernel-level interleaved placing (KLIP) and Activation-level interleaved placing (PLIP) cause a severe overhead of empty memory because of interleaving, i.e., 43 and 82.1%, respectively.

Secondly, we look at the mapping of a volume with $N = 56$ and $H = 8$. This corresponds, for example, to the parameters of the last set of layers of the ResNet-32 architecture with some channels being pruned, as in Joshi et al. (2020). For $WL = 160$, IWAP has varying read and write cycle counts. KLIP guarantees constant read cycle count, but the write cycles can still vary between 3 and 4 based on the pixel to be written, with a small overhead of empty memory, which is about 6%. PLIP guarantees a constant read and write cycle time, with an almost identical empty memory overhead in the case of KLIP. Note that PLIP also guarantees the minimum number of read and write cycles compared to both IWAP and KLIP.

Lastly, we look at the mapping with of a volume with $N = 56$ and $H = 8$, but with $WL = 128$. While IWAP does not guarantee constant read and write cycle count, in this scenario it outperforms all other methods by providing constant and minimum read and write count, other than of course minimum memory requirements. This can be accounted for by the fact that the number of bits of one pixel is exactly 3.5 times the word length. KLIP performs identically to IWAP on the metrics considered, while adding a 4% empty memory overhead. Lastly, PLIP performs the worst by requiring one additional read more compared to the other two methods, and 12% memory overhead. The comparison gives evidence of how different methodologies perform differently for different pixel sizes and word length of the memory, and no methodology outperforms the others a priori. In general terms, IWAP and KLIP perform better for pixels that have a small size compared to the word length, but require the use of write masks to store pixels and, once read, post processing for separating the bits in the word that are required from those that are unwanted. Also, the fact that neither method guarantees a constant number of reading and writing cycles, may be problematic to the design of the dataflow. Conversely, PLIP typically provides better performance for pixel sizes that are greater than the word length, and gives constant read and write cycles count regardless of word length and pixel size. It also does not require the use of write masks, and it may only require the use of post processing for selecting non-zero bits inside the words. These advantages come at the expense of a greater memory requirement. No method, in principle, guarantees a lower read and write cycle count. Lastly, we note that for pixel sizes that are exact multiples of the word size, the three methods map pixels and perform in the exactly the same way.

2.4. Dataflow and Memory Control

Contrary to von-Neumann computing paradigms, IMC considers one operand of the matrix-vector multiplication to be physically instantiated and ready for execution. In the case of inference of CNNs, the two operands of the matrix-vector multiplication are, as presented above, activations and kernel weights. Inherently to the functionality of hardware based on IMC, the kernel weights for every layer are physically instantiated on the devices of the computational memory in the IMC cores, and stationary on their assigned core during execution. This contrasts with von-Neumann hardware models, where the kernel weights for each layer have to be fetched from a memory, and would not be ready for use at the same time. Because of this, in order to increase the utilization of the IMC cores, we would like in principle to parallelize the operation across layers. CNNs are a class of neural networks particularly suited for this specific parallelization because, as detailed in the previous sections and contrary to other classes of neural networks, the matrix-vector operation per layer depends only on a subset of the overall

TABLE 1 | Memory mapping metrics for different channel depth of the pixels (in table, *N*) and word length (*WL*).

		Memory [KB]	% Empty	# Read	# Write
N = 3	IWAP	0.288	0	[1, 2]	[1, 2]
H = 32	KLIP	0.505	43	1	1
WL = 128	PLIP	1.523	82.1	3	1
N = 56	IWAP	1.344	0	[9, 10]	[3, 4]
H = 8	KLIP	1.427	5.9	9	[3, 4]
WL = 160	PLIP	1.436	6.4	9	3
N = 56	IWAP	1.344	0	11	4
H = 8	KLIP	1.4	4	11	4
WL = 128	PLIP	1.528	12	12	4

For all cases, we consider 8-bit precision. Numbers in brackets indicate that different numbers can be obtained for reading or writing different pixels. N and H are the channel depth and the height of the volume of feature maps, respectively. WL is the word length of the local memory.

volume of pixels produced by the previous layer or layers. In this section, we present a method to execute inference of such networks by pipelining across matrix-vector multiplications of different layers.

Assume an array of IMC cores interconnected by a communication fabric. Fundamentally, the dataflow implies that each IMC core operates independently, receiving pixels from cores interconnected to it and triggering its own computations once enough pixels have been received, according to the parameters of the layer it is executing. Upon completion of computation, it will deliver the data to the cores that require it. It is evident from this first summary of the dataflow that the execution will depend highly from the communication fabric that interconnects the IMC cores array. In this work, we assume the presence of a communication fabric that allows any IMC core to communicate with a point-to-point connection to any other IMC core that would require its data for a given application. One such communication fabric and the principles by which to organize communication between an array of IMC cores for the execution of CNNs are described in depth in Dazzi et al. (2019). In our IMC core architecture, the dataflow is enforced by the Dataflow Controller, which also generates the addresses for writing and reading pixels as described in section2.3. We discuss the dataflow control in section 2.4.1, and memory control in section 2.4.2.

Algorithm 1 Dataflow Control

```
1:  def dataflow_ ctrl():
2:    at every timestep do
3:      if pixel_in then
4:        for i from 0 to (nr_parallel − 1) do
5:          store_data(pointer_in, pixel_in)
6:          pointer_in.x.update()
7:          pointer_in.y.update()
8:        end for
9:      end if
10:     if conditions_to_compute(pointer_in): then
11:       for i from (nr_parallel − 1) downto 0 do
12:         data_pixels=read_data(pointer_in-i)
13:         data_to_crossbar(data_pixels)
14:         pointer_out.x.update()
15:         pointer_out.y.update()
16:       end for
17:       if pointer_out == end then
18:         computation_complete=True
19:       end if
20:     end if
21:     if conditions_to_residual(pointer_in) then
22:       read_res()
23:     end if
24:   end
```

2.4.1. Dataflow Control

Algorithm 1 provides a pseudocode that describes the activity of the Dataflow Controller. At every timestep (line 2) the Dataflow Controller first checks the presence of new incoming pixels and stores them in the Input Memory according to one pixel mapping method (lines 3 to 7). Subsequently, it also updates the pointer (*pointer_in*) that keeps track of the current position of the pixels in the input volume. Secondly (lines 10 to 17), it checks whether the conditions to execute the computation are satisfied. When the computation is parallelized, the operations of storing the pixel and updating the input pointers are repeated a number of times (line 4) equal to the number of pixels that are being received (*nr_parallel*). The conditions to start the computation will depend on the pixels present in the Input Memory (and thus from *pointer_in*), and on the parameters of the layer, for example on whether the layer uses padding or the stride of the convolution. In case these conditions are met, the Dataflow Controller reads the pixels from the Input Memory and makes them available to the crossbar array interface (lines 12, 13). Further, it updates a pointer of the output volume that is being computed (*pointer_out*), which will rise a flag indicating the completion of the computation (lines 17, 18) once all the data of the output volume had been computed. Also for the case of fetching the pixels from the Input Memory, in the case of parallelization, the operations of reading the pixels and updating

the output pointers is repeated a number of times equal to the number of pixels that are being produced (line 11). Lastly, the Dataflow Controller checks whether the conditions to send the residual pixel to other IMC cores are met (lines 21, 22), and in case they are verified reads the data from the Input Memory and makes it available to the inter-core communication fabric.

2.4.2. Memory Control

For the control of the Input Memory, the duty of the Dataflow Controller is to generate the read and write addresses for writing and reading the appropriate data. As discussed in section 2.3, the read and write addresses comprise the word address and may or may not include bitline addresses for write masks and bit slicing. In this section, we will consider the case of PLIP memory mapping, in which write masks are not required and the address represents only the address of the memory words. **Algorithm 2** shows two functions, store_data and read_data. The store_data function is called by the Dataflow Controller each time there is new incoming pixels. In principle, one may want to store the entire volume of pixels in the Input Memory. Nevertheless, by pipelining across matrix-vector multiplications, the computation is executed on the data as soon as this is available in the Input Memory, and the input volume is never needed in its entirety for executing the computation. Namely, for a kernel of size $F_1 = F_2 = F$ with padding P and an image of size H by H, given the dataflow presented above, the minimum number of pixels that need to be stored in order to start the computation is $H \cdot (F - P - 1) + (F - P)$, regardless of the stride of the layer. For the sake of simplicity of the memory mapping, at the cost of a small memory overhead, we decide to store in memory $H \cdot F$ pixels. Note that the choice of the number of pixels to store is simply made on grounds of an easier memory mapping, and it is independent from the memory mapping strategies described in section 2.3.

In the store_data pseudocode, at line 2, the incoming data *pixel_in* is written to the Input Memory starting from the cached memory address *addr_w_0*. For any incoming pixels within one column (lines 5 to 7), the new address is computed by incrementing the current address by a quantity $\Delta kernel$, representing the number of words required to store F pixels. This can be seen clearly in the example in **Figure 7D**, where $\Delta kernel$ is 6 words, so that two consecutive pixels on the same column [e.g., $(0, 0)$ and $(1, 0)$ in **Figure 7A**] are stored $\Delta kernel$ words apart. When changing column of pixels (line 3), the address must be first reset to the first position of that column in the memory. This is done by resetting *addr_w_0* to one position Δpix, which stores the position of the first pixel in one column. With reference to **Figure 7D**, $\Delta pix[0] = 0$, $\Delta pix[1] = 2$, $\Delta pix[2] = 4$. When storing the first pixel of column $y = 1$ in **Figure 7A**, i.e., pixel $(0, 1)$, *addr_w_0* must be reset to $\Delta pix[1]$. Since, as described above, we want to store an $H \cdot F$ portion of the input volume of pixels, we must take into consideration how to write pixels in the memory once we reach a column greater than the $F - th$ column. Again with reference to **Figure 7D**, when storing pixels from column $y = 3$, we would reset the initial address to $\Delta pix[0]$, effectively storing column $y = 3$ in place of column $y = 0$, which by that timestep would have become obsolete in terms of computation. Thus, we would effectively permute the position of the column inside the Input Memory between F columns. This is expressed by the mod function in line 4 of **Algorithm 2**.

Algorithm 2 Memory Control

```
1:  def store_data(pointer_in, pixel_in):
2:    write_to_SRAM(addr_w_0, pixel_in)
3:    if pointer_in.x == 0 then
4:      addr_w_0=Δpix[mod(pointer_in.y;kernel_size)]
5:    else
6:      addr_w_0+=Δkernel
7:    end if

8:  def read_data(pointer_in):
9:    for i = 1, . . . , kernel_size do
10:     addr_r=addr_r_0+Δpix·mod(pointer_in.y +
        i;kernel_size)
11:     data_out.append(read_from_SRAM(addr_r))
12:   end for
13:   if pointer_in.x == 0 then
14:     addr_r_0=Δpix[0]
15:   else
16:     addr_r_0=Δkernel
17:   end if
18:   return data_out
```

As relates to reading back the pixels from the Input Memory, the functionality is expressed by the read_data function. With reference to **Figure 2**, we take now in consideration the case in which one row of pixels is read from the Input Memory (e.g., the blue row of pixels in the timestep $t_0 + 1$ portion of **Figure 1**). The read_data function loops through *kernel_size* pixels (line 9) and generates the addresses from which every pixel starts to be read by the read_from_SRAM function (line 11). The various pixels are appended to one another (line 11) and ultimately returned by the function (line 14). The reordering of the pixels because of the permutation of their positions in the Input Memory that was dealt with in the description of the store_data function is performed during the generation of the addresses. Starting from one address indicating the beginning of a row of contiguous pixels *addr_r_0*, this is incremented in the reading loop in the order defined by the current permutation of the positions of the pixels. Taking again as an example the storing of the pixels in **Figure 7A** as shown in **Figure 7D**, in the case in which column $y = 3$ was the last column to be stored in the Input Memory, the ordering of adjacent pixels in memory would be column $y = 3$, column $y = 1$, and column $y = 2$. Thus, for the first row of pixels, the order would be $(0, 3)$ in words 0 and 1; $(0, 1)$ in words 2 and 3; $(0, 2)$ in words 4 and 5. As expressed by the mod function in line 10, starting in this case from $addr\_r\_0 = 0$, *addr_r* is incremented during the loop so that the pixels are read in the order $(0, 1)$, $(0, 2)$, $(0, 3)$. *addr_r_0* is then incremented to the initial position of the next row, or reset (lines 13 to 17). Note that, in the case of a convolution with padding, it is equivalent to having additional pixels equal to zero at the borders of the

input volume. This can be addressed in two ways, either by pre-emptively storing zero pixels in the Input Memory (thus having the store_data function handle the padding) or by adding the zeros once the pixels are read from the Input Memory (thus having the read_data function handle the padding). In the next sections, we will consider the former option.

2.4.3. Example of the Overall Dataflow

Figure 8 shows an example of the evolution of the state variables in **Algorithm 1** at different timesteps. Consider a convolution as in **Figure 8A**, representing a generic layer L of a CNN. It transforms an input volume (left-hand side) to an output volume (right hand-side) with a kernel size of 3×3, stride =1 and without padding. **Figure 8B** shows the input memory and state variables of the dataflow control at 4 different timesteps. Before timestep N, the Input Memory is empty, the memory addresses *addr_w_*0 and *addr_r_*0 are at zero and the pointer *pointer_in* and *pointer_out* are at an invalid value (-1). At a Timestep N, the first pixel is received. This corresponds to the pixel at position $(0, 0)$ on the (x, y) plane, and the pointer to the input volume are consequently updated to values $pointer\_in.x = 0$, $pointer\_in.y = 0$. According to the PLIP memory mapping, the initial address to store the next pixel is updated to $addr\_w\_0 = \Delta kernel$. Lastly, since a single pixel is not sufficient to perform any computation, the conditions to compute return *False*, and *addr_r_*0 and *pointer_out* are not updated. At the subsequent timestep N+1, another pixel along the column $y = 0$ is received. Similarly to the previous timestep, *pointer_in* is updated to $(1, 0)$, *addr_w_*0 is incremented of $\Delta kernel$, while the other pointers remain as they were. At some timestep N+H, the entire column $y = 0$ has been received and store in the Input Memory. Again, the update of *pointer_in* reflects the position reached in the input volume, which is still not enough to perform any matrix-vector multiplication with a 3x3 kernel without padding. As one column has been received in entirety, *addr_w_*0 will have to be reset to the initial position of the second column with $y = 1$, which is equal to $\Delta pix[1]$. Finally, 2H + 3 timesteps after the initial timestep N, two entire columns and three pixels of the third column have been received, and the input pointer *pointer_in* is updated to position $(2, 2)$. As the layer executes a 3×3 convolution without padding, there is sufficient data to execute the first matrix-vector multiplication. The condition_to_compute at line 7 of **Algorithm 1** returns *True* and the input data, highlighted in red in **Figure 8**, is read from the Input Memory and provided to the interface of the crossbar. This matrix-vector multiplication results in the upper leftmost pixel of the output volume, and thus the output pointer *pointer_out* is updated to the newly computed position $(0, 0)$. Lastly, the initial address for data to be read is updated to the position of the latest pixel to be read, equal to $3\Delta kernel$.

2.4.4. Splitting of Layers Onto Multiple IMC Cores

So far, we assumed the kernel matrix can fit in its entirety on the crossbar array of the IMC core. Nevertheless, because of the variability of kernel matrices even within a single CNN on the one hand and the fixed size of crossbar arrays on the other, we must take into consideration the case in which one crossbar array does not have enough rows or columns to fit one kernel matrix. Without loss of generality, we consider the two cases separately, one where the number of rows in one crossbar is not sufficient to fit one kernel matrix and one where the number of columns is not sufficient. As expressed in section 2.2, because of the typical shape of kernel matrices on computational memory, the former case will be, in principle, the most common.

Assume the case of a crossbar array with R rows and C column, and layer whose kernel matrix requires R_{KM} rows and C_{KM} columns. If $2R \geq R_{KM} > R$ and $C_{KM} \leq C$, we can split the kernel matrix between two crossbar arrays so that they calculate partial accumulations of the same convolution. This is equivalent to splitting the original matrix-vector multiplication between a patch of the input volume $\boldsymbol{\alpha}_{i:i+F-1,j:j+F-1}$ and a kernel matrix $\boldsymbol{K}$ as in Equation (5), with each IMC core executing one matrix-vector multiplication.

$$\begin{aligned}\boldsymbol{\alpha}_{i:i+F-1,j:j+F-1} \cdot \boldsymbol{K} &= \begin{bmatrix} \alpha_{i,j}^{0} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N/2-1} \\ \alpha_{i,j}^{N/2} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N-1} \end{bmatrix}^T \begin{bmatrix} k_{0,0}^{0,0} & \cdots & k_{0,0}^{0,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N/2-1,0} & \cdots & k_{F-1,F-1}^{N/2-1,M} \\ k_{0,0}^{N/2,0} & \cdots & k_{0,0}^{N/2,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N,0} & \cdots & k_{F-1,F-1}^{N,M} \end{bmatrix} = \\ &= \begin{bmatrix} \alpha_{i,j}^{0} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N/2-1} \end{bmatrix}^T \begin{bmatrix} k_{0,0}^{0,0} & \cdots & k_{0,0}^{0,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N/2-1,0} & \cdots & k_{F-1,F-1}^{N/2-1,M} \end{bmatrix} + \\ &+ \begin{bmatrix} \alpha_{i,j}^{N/2} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N-1} \end{bmatrix}^T \begin{bmatrix} k_{0,0}^{N/2,0} & \cdots & k_{0,0}^{N/2,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N,0} & \cdots & k_{F-1,F-1}^{N,M} \end{bmatrix} \end{aligned} \tag{5}$$

This is effectively equivalent to splitting the layer into two different layers, where the IMC core executing the first half of the computation forwards it to the other IMC core, which sums the two partial results and execute the required digital processing (batch normalization, activation function and possibly residual additions). In terms of dataflow, this is not different from having two separate layers.

Assume now the case in which $R_{KM} \leq R$ and $2C \geq C_{KM} > C$. In this case, we can split the kernel matrix between two crossbar arrays so that they calculate different channels of the same convolution. This is equivalent to performing the splitting shown in Equation (6), where $\oplus$ indicates the concatenation operation. This is also equivalent to splitting the layer into two different layers, and the same considerations made for the previous case hold.

$$\begin{aligned}\boldsymbol{\alpha}_{i:i+F-1,j:j+F-1} \cdot \boldsymbol{K} = &\begin{bmatrix} \alpha_{i,j}^{0} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N-1} \end{bmatrix}^T \begin{bmatrix} k_{0,0}^{0,0} & \cdots & k_{0,0}^{0,M/2-1} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N-1,0} & \cdots & k_{F-1,F-1}^{N-1,M/2-1} \end{bmatrix} \oplus \\ &\begin{bmatrix} \alpha_{i,j}^{0} \\ \cdots \\ \alpha_{i+F-1,j+F-1}^{N-1} \end{bmatrix}^T \begin{bmatrix} k_{0,0}^{0,M/2} & \cdots & k_{0,0}^{0,M} \\ \cdots & \cdots & \cdots \\ k_{F-1,F-1}^{N-1,M/2} & \cdots & k_{F-1,F-1}^{N-1,M-1} \end{bmatrix} \end{aligned} \tag{6}$$

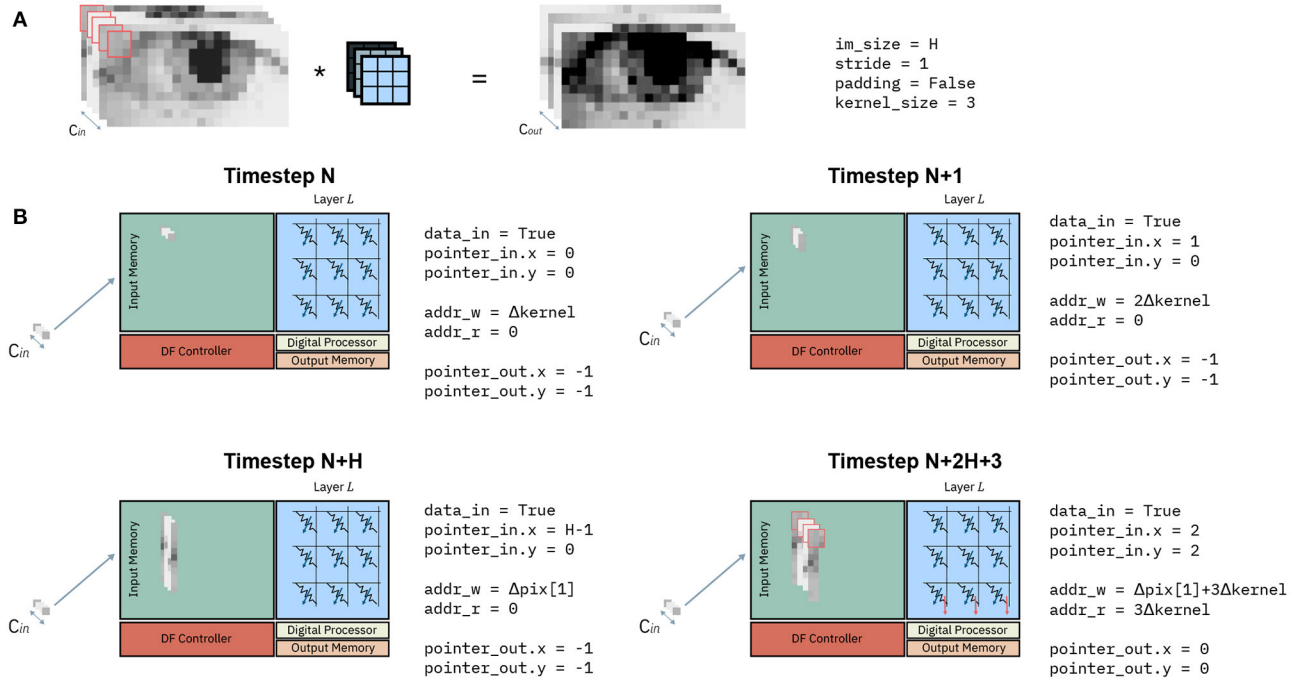


FIGURE 8 | Example of dataflow for one IMC core at subsequent timesteps. **(A)** Example of a convolution. The input volume of depth C_{in} (left) is convolved with the kernel **(in blue)** to obtain an output volume of depth C_{out}. **(B)** Input memory and state variables of an IMC core at different timesteps.

This concept can be of course generalized to a splitting into an arbitrary number of IMC cores, combining both types of splitting, provided connectivity between these IMC cores exists.

3. RESULTS

3.1. Execution of ResNet-32 on an a IMC Accelerator

In this section, we will present the employment of our proposed methodologies in the mapping of a CNN on an array of IMC cores and discuss the behavior of the overall inter- and intra-core dataflow. Specifically, we will consider the inference of ResNet-32 for the CIFAR-10 dataset, which represents a state-of-the art accuracy network for such dataset. While our methodologies are generic with respect to the IMC technology that is employed and are orthogonal to the implementation of the crossbar array and/or the precision of the matrix-vector multiplication operations, in this section we make some assumptions on the hardware. Firstly, we assume the crossbar arrays of each IMC core to have size of 256 × 256. The crossbar arrays operate in a fully parallel mode, and utilizes Phase-Change Memory (PCM) as the IMC devices. Specifically, in this implementation, each weight is mapped onto a differential pair of PCM devices. Input and output data is provided in 8-bit int fixed precision format. Regarding the Digital Processor, this implementation assumes a light digital processing element that can perform batch normalization, residual addition and ReLU. This choice is justified by the fact that the great majority of state-of-the-art CNNs, among which ResNet-32, feature these three operations on activations. Regarding the reduced precision implementation of ResNet-32, it is based on the one presented in Joshi et al. (2020). Such implementation presents a training strategy for the same hardware and numerical precision presented in this work, and reaches 93.7% accuracy on CIFAR-10. Also in this case, note that our proposed methodologies and dataflow are orthogonal to the training process and optimization of the DNNs executed on the IMC hardware.

In this section, we first present the mapping of the network on the IMC core array in section 3.1.1. In section 3.1.2, we present the mapping strategy employed for the activations, and in section 3.1.3, we give display of the dataflow and the overall performance. In section 3.1.4, we discuss the possibility of speedup of the dataflow. Lastly, in section 3.2, we present an implementation of the dataflow controller and input memory implemented in 14 nm CMOS technology.

3.1.1. Mapping of the Kernel Weights on the Array

Figure 1A shows a representation of ResNet-32 and its mapping on an array of IMC cores. ResNet-32 comprises 34 layers. Specifically, 31 convolutional layers with kernel size $F_1 = F_2 = F = 3$, one fully connected layer at the end of the network for classification, and two layers for resampling of the residual connection with kernel size $F_1 = F_2 = F = 1$. Firstly, we must store the kernel matrices on the computational memory of the IMC core array. We proceed in the following way: we map

TABLE 2 | Layer specifications for ResNet-32 network.

#Layers	$H_{in} \times W_{in}$	$H_{out} \times W_{out}$	Stride	C_{in}	C_{out}	Kernel Size	#Rows	#Columns	Pixel Size [bits]
1	32 × 32	32 × 32	1	3	16	3 × 3	27	16	24
2/11	32 × 32	32 × 32	1	16	16	3 × 3	144	16	128
RS1	32 × 32	16 × 16	2	16	28	1 × 1	16	28	128
12	32 × 32	16 × 16	2	28	28	3 × 3	252	28	224
13/21	16 × 16	16 × 16	1	28	28	3 × 3	252	28	224
RS2	16 × 16	8 × 8	2	28	56	1 × 1	28	56	224
22	16 × 16	8 × 8	2	56	56	3 × 3	504	56	448
23/31	8 × 8	8 × 8	2	56	56	3 × 3	504	56	448
FC	1 × 1	1 × 1	2	56	10	/	56	10	448

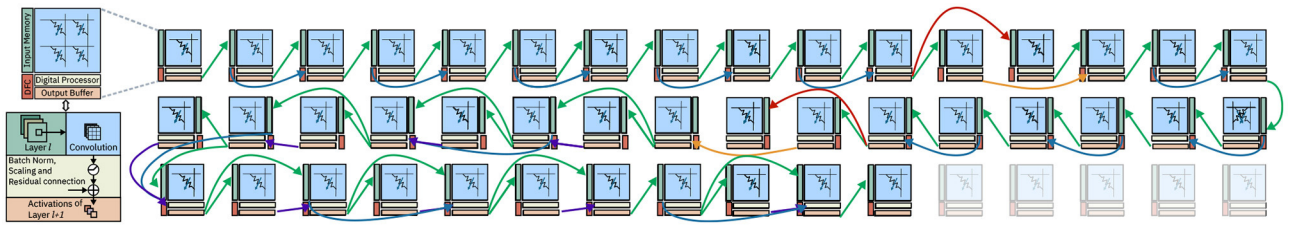


FIGURE 9 | Mapping of ResNet-32 on an array of IMC cores. Different colored arrows represent different functionality of the communication. Specifically, green arrows represents feedforward communication and the blue ones denote residual connections. Moreover, the red arrows indicate communication to resampling layers, whereas the orange ones represent residual connections from a resampling layer. Finally, the purple arrows signify partial summation for layers split on multiple IMC cores.

one layer per IMC core; in case the crossbar array did not have enough rows or columns to fit the kernel matrix for one layer, we split the kernel matrix and map it on multiple IMC cores. For the sake of simplicity, in this first discussion of the dataflow, we will assume the mapping of a single kernel matrix per layer, that is, no kernel replication as in section 2.2. **Table 2** reports the required number of rows and columns for the kernel matrices of each layer. In order to better match the size of the crossbar array, we have trained a version of ResNet-32 with a slightly lower number of channels for the layers RS1, RS2, 12, 32 and FC. In general, all channel depth of 32 have been reduced to 28, and all channel depths of 64 to 56. Such modification is consistent with the implementation in Joshi et al. (2020). Given the number of required rows and columns, all layers except 22/31 can be mapped onto a single crossbar array. Layers 22/31 require more rows i.e., 504, than the 256 rows available in a single crossbar array. Thus, for each of these layers we split the kernel matrix into two smaller matrices of size 256 × 56. In this way, we perform two partial accumulations of the overall matrix-vector multiplication. Note that, for these layers, activations $\alpha^{0,L}$ to $\alpha^{27,L}$ are input to one crossbar array, and $\alpha^{28,L}$ to $\alpha^{56,L}$ are input to another one. We will discuss in more detail, in section 3.1.3, how these two partial accumulations are combined. As a result of this overall mapping strategy, the ResNet-32 network requires 43 IMC cores.

3.1.2. Mapping of Activations on the Input Memory

Table 2 shows the parameters of the layers of ResNet-32 as well as the size in bits of every pixel that needs to be stored in the Input Memory of the corresponding IMC core. We have assumed a word length of 128 bits and a word depth of 512, amounting to a total to 8KB SRAM for the Input Memory of each IMC core. We have used IWAP to store the pixels for layer 1 and ALIP for storing pixels for the remaining layers. In fact, as noted in section 2.3.5, IWAP works best for layers where the size of the pixels is small compared to the word length, as is the case with layer 1, while for the other layers KLIP or PLIP are preferable. From the consideration on the minimum memory requirements in section 2.4.2, the memory requirements for the different convolutional layers are 0.28 KB for layer 1 and 1.5 KB for all the others. Both requirements are lower than the size of the Input Memory.

3.1.3. Dataflow and Performance

Figure 9 shows the mapping of ResNet-32 on an 8-by-6 array of IMC cores. Arrows between cores represent communication channels. We assume a communication fabric as in Dazzi et al. (2019), such that every connection can be satisfied as required by the network. This topology assures that all data delivery occurs within one timestep, guaranteeing that the pipeline never stalls. In **Figure 9** different colored arrows represent different functionality of the communication. Specifically, green arrows represents feedforward communication and the blue ones denote residual connections. Moreover, the red arrows indicate communication to resampling layers, whereas the orange ones represent residual connections from a resampling layer. Finally, the purple arrows signify partial summation for layers split

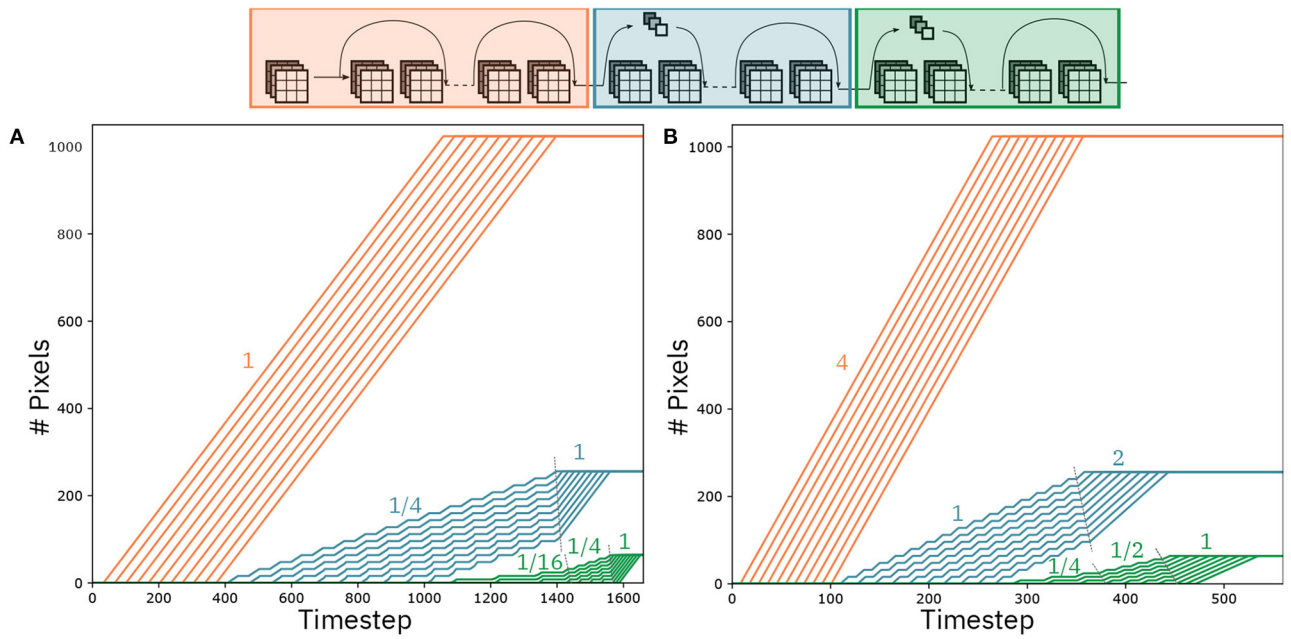


FIGURE 10 | Pixels per layer as a function of the timesteps. Layers with different feature map size are color-coded in different colors. **(A)** Pixels being computed by mapping at most 1 kernel matrix per layer. **(B)** Pixels being computed by mapping multiple kernel matrices per layer.

on multiple IMC cores. **Figure 10A** shows the evolution of the computation per convolutional layer as a function of the timestep. In an ideal pipelining scenario, one would expect that one pixel is computed at every timestep, such that the number of pixels to be computed per layer is equal to the number of timesteps the IMC core is active. This implies that the slope of the curves in **Figure 10A** should be equal to one. The slope is indeed equal to one for all layers that do not need strided convolutions (see red curves in **Figure 10A**). However, the computation of layer 12, with stride equal to two, can only be executed every two rows and every two columns. As the size of the feature map for that layer is 16x16, the IMC core assigned to layer 12 will perform computations every 2 timesteps, while receiving the pixels from layer 11, and stall for 16 timesteps when receiving columns on which no computation is required. Overall, on average, this implies that there will be a computation every 4 cycles, meaning the average slope is 1/4. As all the layers subsequent to layer 12 are connected in a feedforward way, the reduced slope of the computation vs. timesteps curve propagates to all the subsequent layers until the point at which layer 12 is completed. This is evident in **Figure 10A**, where the layers from 13 to 21 progress also at slope 1/4 up to point at which layer 12 is completed, after which the slope progresses at slope 1. Moreover, this is propagated to the layers from 22 to 31 where the slope, which is supposed to be equal to 1/4 (the stride equal to two in layer 22), is reduced of an additional factor 4, i.e., 1/16.

As a result, the overall latency for a single classification is 1,628 cycles. Assuming a timestep of 100 ns, this results in 161.8 us latency for the classification of a single image and 9,650 Images/s throughput.

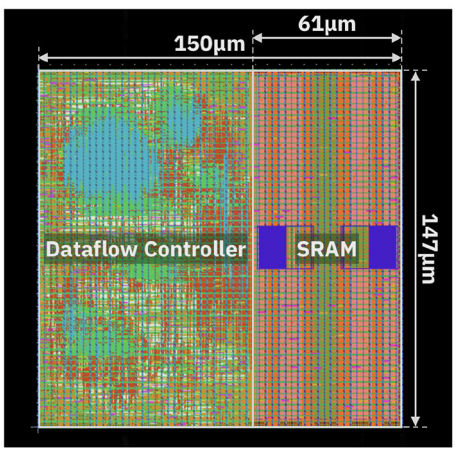


FIGURE 11 | Physical layout of the Dataflow Controller and Input Memory.

3.1.4. Dataflow Speedup

In an architecture as shown in **Figure 9**, the bandwidth of the communication links is tailored on the most expensive data communication. The bandwidth requirements are calculated as

$$B = (n_{\text{bits}} \cdot C_{out}^{max})/T_{timestep} \tag{7}$$

where n_{bits} denotes the number of bits per single activation, C_{out}^{max} indicates the maximum number of output channels of the convolutional layers throughout the network, and $T_{timestep}$ represents the duration in seconds of one timestep. Assuming that we want to deliver all the data from one IMC core at

most within one timestep of 100 ns, the maximum bandwidth required is 4.5 Gbps, from layers 22 to 31. This corresponds to the layers colored in green in **Figure 10**. In ResNet-32 and generally in the design of CNNs, the number of channels per layer doubles for every group of layers. In **Figure 10**, the group of layers marked in blue have twice the number of channels compared to the layers in red, and the layers marked in green have four time the number of channels compared to the layers in red. From Equation 7, it follows that the same bandwidth that allows to send 1 activation in 1 timestep for the last group of layers, allows to deliver in 1 timestep 4 activations for the first group of layers and 2 for the second, respectively. Based on this observation, we present a pipeline speedup method that uses the same hardware communication requirements as the regular dataflow. **Figure 10B** shows the number of activations computed per layer as the timesteps increase, assuming a sped up dataflow. According to this dataflow, we assume that the IMC cores parallelize the computation of matrix-vector operations based on one of the methods presented in section 2.2. Specifically, the first group of layers (red) computes four activations per timestep, the second group of layers (blue) computes two activations per timestep, and the third computes one activation per timestep. As argued before, as long as the kernel matrices can fit on the crossbar arrays of the IMC cores, this does not modify the bandwidth requirements of the hardware implementation. In accordance with the activations per timesteps that are computed per layer, **Figure 10B** shows that the slope of the activation/timestep curve is equal to four for the first group of layers. This seed-up is reflected on the post-stride layers (in blue), where the slope of the layers is increased by 4x compared to **Figure 10A**, as long as its computation depends on the data that is received from the first set of layers, after which it proceeds at the speed set by the IMC core (2 activations/timestep). Similarly for the third group of layers, the speed-up of the preceding layers reflects on its own execution that is sped up as long as it depends on the data that are received by the previous layers, with the slope gradually increasing from 1/4 to 1/2 and ultimately to 1 when the second group of layers have finished the computations. As the number of activations that are produced per layer depend on both the rate of the preceding layers and its own rate, the speed-up is clearly non-linear. For the case of ResNet-32, it results in a latency for a single classification of 526 cycles, resulting to a speed-up of a factor 3.1. The throughput, which depends primarily on the rate of the first layers, is equal to 38,600 Images/s for a batch of 100 images to be classified.

3.2. Hardware Implementation

We demonstrated the dataflow presented in this work via experiments and simulation. Specifically, the Input Memory, Dataflow Controller and communication fabric have been implemented in CMOS 14nm technology. The communication fabric implements the topology presented in Dazzi et al. (2019). These elements represent the digital framework around an array of 8-by-8 IMC cores, constituting an IMC based inference engine. This architecture allows the execution of inference for CNNs through the dataflow described in section 2.4. Note that, in this implementation, we are agnostic of the computational elements, i.e., Crossbar Array and Digital Processor. Based on the application and the desired accuracy, different computational elements can be utilized in such an architecture to build the overall accelerator.

TABLE 3 | Power consumption and area occupation for Input Memory, Dataflow Controller, and communication links implemented in 14 nm CMOS technology.

Area DFC [mm^2]	**Area SRAM [mm^2]**
0.0132	0.0091
Power DFC+SRAM [*mW*]	**Power links [mW]**
10.544	0.775

The design assumes supply voltage equal to 0.8 V and clock frequency of 500 MHz. **Table 3** shows the power consumption and area occupation of the blocks. The Input Memory has been implemented as an SRAM memory with word length equal to 128 bits, comprising 8 KB of memory. **Figure 11** displays the layout of the Input Memory and Dataflow Controller. Overall, the Input Memory occupies 0.0091 mm^2, while the Dataflow Controller occupies 0.0132 mm^2. In order to have a realistic estimation of the power consumption, we evaluate the joint operation of both blocks. Specifically, storing and reading from the Input Memory is only executed based on the dataflow set by the Dataflow Controller. The joint power consumption of Dataflow Controller and Input Memory is equal to 10.54 mW. A breakdown of the power consumption is reported in **Table 4**. Most power is spent in the Sequential elements, which is justified considering the significant amount of buffering of data and pointers inside the Dataflow Controller.

As relates to the communication fabric, the design implements synchronous communication with latch-to-latch links. The average power consumption is 0.775 mW. The energy efficiency of the links depends on their lengths, and ranges between 39.4 and 346.5 fJ/bit.

We now discuss the usage of the computational elements within the architecture described above. In accordance with the case study presented in section 3.1, we assume crossbar arrays of size 256 × 256. We assume the use of phase-change memory (PCM) (Burr et al., 2017) as the computational memory. In order to estimate the performance of such an array, we take into consideration the analog and digital contributions to the energy consumption of the crossbar array. Specifically, as regards the analog contributions, we consider the energy for the voltage regulators for the columns of the array, the energy for the analog-domain computation and the energy of the ADCs for the analog-to-digital conversion. For the analog computation we assume the average conductance of the PCM devices to be 2.5 μS and the read voltage to be 0.2 V. Also, the energy for the ADCs is taken from Kull et al. (2017). As regards the digital contributions, for the crossbar array we also take into consideration the pulse-width modulator (PWM). The PWM applies the input to the array as voltage pulses, performing a digital-to-analog conversion from an 8-bit signed integer to a read voltage pulse. Lastly, we assume a Digital Processor performing batch normalization, ReLU and addition for the residual connection. The power consumption

TABLE 4 | Power breakdown for the internal components of the Dataflow Controller.

	Internal power	Switching Power	Leakage power	Total Power	Percentage (%)
Sequential	4.052	0.3429	0.06095	4.455	42.25
SRAM macro	2.548	0.03460	0.02155	2.604	24.7
Combinational	0.5563	1.625	0.04357	2.225	21.1
Clock	0.1884	1.071	0.001539	1.261	11.96
Total	7.344	3.073	0.1276	10.54	100

of the digital blocks was obtained from RTL simulations of the HDL code of one implementation. Overall, the resulting energy efficiency is 10.5 TOPS/W. Note that, although our chip design is based on 256 × 256 crossbar arrays, it can be readily estimated that by increasing the crossbar array size to 512 × 512 the efficiency becomes 16.3 TOPS/W. Comparing the resulting energy efficiency with digital accelerators targeting similar datasets (Sim et al., 2016) and IMC-based accelerators (Ando et al., 2017), our architecture outperforms them up to a factor 7.4x and 5.2x, respectively. Moreover, our implementation yields very high operations per second, providing 1.008 TOPS. Such performance is, respectively, 7.36x and 409x higher compared to SRAM-based (Jia et al., 2020) and ReRAM-based (Xue et al., 2020) IMC array implementations. In terms of performance density, our proposed PCM-based array provides 1.59 TOPS/mm^2, outperforming the aforementioned implementations by 41.76x and 1322x, respectively. Lastly, comparing our results in terms of thoughput with the ones reported by systolic array-based ASICs on a ResNet of similar depth (Andri et al., 2018), we achieve a throughput that is 206x higher with the dataflow described in section 3.1.3 and 826x higher with the dataflow speedup method described in section 3.1.4. Furthermore, our presented throughputs are, respectively, 7x and 31x higher compared to that of ASICs targeting the same dataset (Esser et al., 2016).

4. DISCUSSION

Various optimized dataflows for efficient hardware deployment of CNNs have been explored in literature. Specifically, pipelined dataflows that exploit parallelization of the computation between layers, have demonstrated mitigation of bandwidth limitations (Goetschalckx and Verhelst, 2019). Similarly, dataflows exploiting the local computation of feature maps between CNN layers have already been employed in a variety of hardware accelerators. For example, dataflows that fuse the computation between subsets of subsequent layers within a CNN (Alwani et al., 2016) have been used in implementations of CNNs on FPGA-based accelerators (Reggiani et al., 2019). Moreover, such advanced dataflows have been proposed for heterogeneous architectures in order to improve throughput and avoid use of off-chip memory (Wei et al., 2018). Dataflows that pipeline the computation across layers in an end-to-end fashion have recently been employed in accelerators based on IMC for inference (Shafiee et al., 2016), and similar concepts have also been used for training dataflows (Song et al., 2017).

All these contributions consider simple feed-forward networks. Furthermore, in all the prior works, the optimization and possible parallelization of the computation performed on the crossbar array has not been considered. In this work we present a fully pipelined, end-to-end inference dataflow based on the proposed optimized mapping of synaptic weights on the crossbars and of activations on the on-chip memory. The dataflow is agnostic of the connectivity of the CNN layers and is not limited to simple feedforward networks. Among the contributions to the mapping of kernel weights on computational memory, it is worth mentioning the results on different ordering strategies presented in Yue et al. (2020). While this work targets easier data handling in the case the kernel matrices were split onto multiple crossbars, we take an orthogonal approach by trying to optimize device utilization when mapping multiple replicas of the kernel weights on the same crossbar array. Clearly, different ordering techniques can in principle be used alongside the optimization strategies presented in this paper. Optimization of the kernel mapping has also been explored in Peng et al. (2019). This work reorders and partitions kernel matrices in order to facilitate their mapping on the crossbar arrays, and proposes a novel dataflow in order to allow DNN inference with such a kernel partition. In our work, we propose kernel mapping methods that do not require *ad-hoc* dataflows for their execution, so that they can be applied to well-established dataflows and compatible with widely used environments for neural network definitions.

To summarize, in this paper we have introduced various methodologies for accelerating convolutional neural networks (CNNs) on hardware based on IMC. We introduced an architecture of a computational memory (IMC) core for execution of CNNs. We presented an exhaustive set of methods for mapping kernel matrices on the crossbar arrays of the IMC cores. For a given number of matrix-vector multiplications to be executed in parallel, these methods allow the size of the crossbar array required to map the kernel matrices to be minimized. Secondly, we presented three novel methods for storing activations on the local memory of the IMC cores. Based on the overall size of the activations and on the parameters of the local memory used for storing them, these different methods for storing activations can minimize

the number of read and write cycles required. Using these mapping schemes for both kernel matrices and activations, we then proposed an end-to-end dataflow for execution of CNNs on an array of IMC cores. We studied the performance of this dataflow on ResNet-32 for CIFAR-10 and presented the implementation of the dataflow control logic in 14nm CMOS technology, demonstrating a throughput that is at least 7x higher than achieved by ASICs targeting the same neural network and dataset.

DATA AVAILABILITY STATEMENT

Some of the datasets presented in this article are not readily available because simulators used for some results are proprietary. Other datasets presented in this study can be found in online repositories. CIFAR-10 dataset is available for download at https://www.cs.toronto.edu/~kriz/cifar.html. ResNet models are typically available in machine learning frameworks such as PyTorch (https://pytorch.org/hub/pytorch_vision_resnet/).

AUTHOR CONTRIBUTIONS

MD conceived the IMC core architecture with support from AS and EE. MD conceived the weight and activation mapping methodologies and developed the python simulator for the experiments. MD developed the dataflow concept for CNNs on IMC with support from AS, LB, and EE. MD designed the hardware implementation of the dataflow controller. MD and EE wrote the manuscript with inputs from LB and AS. AS, LB, and EE supervised the project. All authors contributed to the article and approved the submitted version.

ACKNOWLEDGMENTS

We would like to thank Christophe Piveteau for the fruitful discussions and the help with the simulation of the dataflow. We would also like to thank Matthias Braendli from IBM Research Europe for his technical support in the 14 nm CMOS design. Lastly, we thank our colleagues at IBM Research, in particular those affiliated to the IBM AI Hardware Center.

REFERENCES

Alwani, M., Chen, H., Ferdman, M., and Milder, P. (2016). "Fused-layerCNN accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Taipei: IEEE), 1–12. doi: 10.1109/MICRO.2016.7783725

Ando, K., Ueyoshi, K., Orimo, K., Yonekawa, H., Sato, S., Nakahara, H., et al. (2017). BRein memory: a single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. *IEEE J. Solid State Circ.* 53, 983–994. doi: 10.1109/JSSC.2017.2778702

Andri, R., Cavigelli, L., Rossi, D., and Benini, L. (2018). "Hyperdrive: a systolically scalable binary-weight CNN inference engine for mW IoT end-nodes," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (Hong Kong: IEEE), 509–515. doi: 10.1109/ISVLSI.2018.00099

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys. X* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. (2016). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circ.* 52, 127–138. doi: 10.1109/JSSC.2016.2616357

Dazzi, M., Sebastian, A., Francese, P. A., Parnell, T., Benini, L., and Eleftheriou, E. (2019). 5 parallel prism: a topology for pipelined implementations of convolutional neural networks using computational memory. *arXiv preprint arXiv:1906.03474*.

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Goetschalckx, K., and Verhelst, M. (2019). Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution. *IEEE J. Emerg. Select. Top. Circ. Syst.* 9, 323–331. doi: 10.1109/JETCAS.2019.2905361

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, 770–778. doi: 10.1109/CVPR.2016.90

He, Y., Zhang, X., and Sun, J. (2017). "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, Venice, 1389–1397. doi: 10.1109/ICCV.2017.155

Hu, M., Graves, C. E., Li, C., Li, Y., Ge, N., Montgomery, E., et al. (2018). Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* 30:1705914. doi: 10.1002/adma.201705914

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, 4700–4708. doi: 10.1109/CVPR.2017.243

Ielmini, D., and Wong, H.-S. P. (2018). In-memory computing with resistive switching devices. *Nat. Electron.* 1:333. doi: 10.1038/s41928-018-0092-2

Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., et al. (2018). "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 2704–2713. doi: 10.1109/CVPR.2018.00286

Jia, H., Valavi, H., Tang, Y., Zhang, J., and Verma, N. (2020). A programmable heterogeneous microprocessor based on bit-scalable in-memory computing. *IEEE J. Solid State Circ.* 55, 2609–2621. doi: 10.1109/JSSC.2020.2987714

Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S., Piveteau, C., et al. (2020). Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* 11, 1–13. doi: 10.1038/s41467-020-16108-9

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON: ACM), 1–12. doi: 10.1145/3079856.3080246

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 1097–1105.

Kull, L., Luu, D., Menolfi, C., Braendli, M., Francese, P. A., Morf, T., et al. (2017). "28.5 a 10b 1.5 GS/s pipelined-SAR ADC with background second-stage common-mode regulation and offset calibration in 14nm CMOS FinFET," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), 474–475. doi: 10.1109/ISSCC.2017.7870467

Le Gallo, M., Sebastian, A., Mathis, R., Manica, M., Giefers, H., Tuma, T., et al. (2018). Mixed-precision in-memory computing. *Nat. Electron.* 1, 246–253. doi: 10.1038/s41928-018-0054-8

Nandakumar, S., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., and Eleftheriou, E. (2018). "Mixed-precision architecture based on computational memory for training deep neural networks," in *International Symposium on Circuits and Systems (ISCAS)* (Florence: IEEE), 1–5. doi: 10.1109/ISCAS.2018.8351656

Peng, X., Liu, R., and Yu, S. (2019). "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* (Sapporo: IEEE), 1–5. doi: 10.1109/ISCAS.2019.8702715

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521:61. doi: 10.1038/nature14441

Reggiani, E., Rabozzi, M., Nestorov, A. M., Scolari, A., Stornaiuolo, L., and Santambrogio, M. (2019). "Pareto optimal design space exploration for accelerated CNN on FPGA," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (Rio de Janeiro: IEEE), 107–114. doi: 10.1109/IPDPSW.2019.00028

Sebastian, A., Boybat, I., Dazzi, M., Giannopoulos, I., Jonnalagadda, V., Joshi, V., et al. (2019). "Computational memory-based inference and training of deep neural networks," in *Proceedings of the IEEE Symposium on VLSI Circuits* (Kyoto). doi: 10.23919/VLSIC.2019.8778178

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E. (2020). Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* 15, 529–544. doi: 10.1038/s41565-020-0655-z

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). ISAAC: a convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars. *ACM SIGARCH Comput. Arch. News* 44, 14–26. doi: 10.1145/3007787.3001139

Sim, J., Park, J.-S., Kim, M., Bae, D., Choi, Y., and Kim, L.-S. (2016). "14.6 a 1.42 TOPS/W deep convolutional neural network recognition processor for intelligent IOE systems," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA: IEEE), 264–265. doi: 10.1109/ISSCC.2016.7418008

Song, L., Qian, X., Li, H., and Chen, Y. (2017). "Pipelayer: a pipelined ReRAM-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Austin, TX: IEEE), 541–552. doi: 10.1109/HPCA.2017.55

Valavi, H., Ramadge, P. J., Nestler, E., and Verma, N. (2019). A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute. *IEEE J. Solid State Circ.* 54, 1789–1799. doi: 10.1109/JSSC.2019.2899730

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wei, X., Liang, Y., Li, X., Yu, C. H., Zhang, P., and Cong, J. (2018). "TGPA: tile-grained pipeline architecture for low latency CNN inference," in *Proceedings of the International Conference on Computer-Aided Design*, San Diego, CA, 1–8. doi: 10.1145/3240765.3240856

Xia, Q., and Yang, J. J. (2019). Memristive crossbar arrays for brain-inspired computing. *Nat. Mater.* 18:309. doi: 10.1038/s41563-019-0291-x

Xue, C.-X., Huang, T.-Y., Liu, J.-S., Chang, T.-W., Kao, H.-Y., Wang, J.-H., et al. (2020). "15.4 A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Diego, CA: IEEE), 244–246. doi: 10.1109/ISSCC19947.2020.9063078

Yue, J., Yuan, Z., Feng, X., He, Y., Zhang, Z., Si, X., et al. (2020). "14.3 a 65nm computing-in-memory-based CNN processor with 2.9-to-35.8 TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Diego, CA: IEEE), 234–236. doi: 10.1109/ISSCC19947.2020.9062958

Conflict of Interest: MD, AS, and EE were employed by the company IBM Research Zurich.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

HYPOTHESIS AND THEORY
published: 06 September 2021
doi: 10.3389/fnins.2021.732368

Considerations for Neuromorphic Supercomputing in Semiconducting and Superconducting Optoelectronic Hardware

Bryce A. Primavera[1,2]* *and Jeffrey M. Shainline*[1]

[1] National Institute of Standards and Technology, Boulder, CO, United States, [2] Department of Physics, University of Colorado Boulder, Boulder, CO, United States

Any large-scale spiking neuromorphic system striving for complexity at the level of the human brain and beyond will need to be co-optimized for communication and computation. Such reasoning leads to the proposal for optoelectronic neuromorphic platforms that leverage the complementary properties of optics and electronics. Starting from the conjecture that future large-scale neuromorphic systems will utilize integrated photonics and fiber optics for communication in conjunction with analog electronics for computation, we consider two possible paths toward achieving this vision. The first is a semiconductor platform based on analog CMOS circuits and waveguide-integrated photodiodes. The second is a superconducting approach that utilizes Josephson junctions and waveguide-integrated superconducting single-photon detectors. We discuss available devices, assess scaling potential, and provide a list of key metrics and demonstrations for each platform. Both platforms hold potential, but their development will diverge in important respects. Semiconductor systems benefit from a robust fabrication ecosystem and can build on extensive progress made in purely electronic neuromorphic computing but will require III-V light source integration with electronics at an unprecedented scale, further advances in ultra-low capacitance photodiodes, and success from emerging memory technologies. Superconducting systems place near theoretically minimum burdens on light sources (a tremendous boon to one of the most speculative aspects of either platform) and provide new opportunities for integrated, high-endurance synaptic memory. However, superconducting optoelectronic systems will also contend with interfacing low-voltage electronic circuits to semiconductor light sources, the serial biasing of superconducting devices on an unprecedented scale, a less mature fabrication ecosystem, and cryogenic infrastructure.

Keywords: neuromorphic, superconducting electronics, optoelectronic, large-scale computing systems, spiking network, photonics

OPEN ACCESS

Edited by:
Alexantrou Serb,
University of Southampton,
United Kingdom

Reviewed by:
Dimitra G. Georgiadou,
University of Southampton,
United Kingdom
Hermann Kohlstedt,
University of Kiel, Germany

***Correspondence:**
Bryce A. Primavera
bryce.primavera@nist.gov

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *29 June 2021*
Accepted: *09 August 2021*
Published: *06 September 2021*

Citation:
Primavera BA and Shainline JM (2021) Considerations for Neuromorphic Supercomputing in Semiconducting and Superconducting Optoelectronic Hardware. Front. Neurosci. 15:732368. doi: 10.3389/fnins.2021.732368

1. INTRODUCTION

The foundations of cognition remain a great frontier of science, with potentially enormous ramifications for technology and society. A hardware capable of simulating spiking neural networks with the scale and complexity of the brain or even beyond could be a powerful tool in deciphering this enigma. Achieving such large-scale systems has proven to be non-trivial with established

CMOS hardware (Furber, 2016). A significant challenge will be to enable efficient communication with low-latency amongst billions or trillions of neurons. Optics appears well-matched to the task, as the lack of resistive, capacitive, and inductive parasitics makes optical links more amenable to high fan-out than electrical interconnects (Shainline et al., 2019). While digital systems partially circumvent this issue by leveraging time-multiplexing to artificially increase fan-out (Young et al., 2019), multiplexing introduces latency that scales exponentially above a certain data load (Hennessy and Patterson, 2011). Optical interconnects may enable direct connections between neurons which would eliminate all traffic-induced delays and support larger, faster, and more interconnected networks. However, while the lack of interaction between photons is beneficial for reducing parasitics during communication, it is a detriment to computation. Electronic circuits are better suited to implement complex, nonlinear neuronal functions. It is reasonable to anticipate performance gains from optoelectronic neural systems leveraging optics for communication and electronics for computation, provided the hardware can be realized.

Our proposal to fabricate a direct, physical connection between every pair of connected neurons is known as the fully dedicated axon approach to communication (Segal et al., 2016). While this strategy requires largely fixing network topology in hardware—a chief disadvantage when compared with highly reconfigurable digital systems—the reduced overhead and elimination of communication bottlenecks will greatly benefit performance. We further specify that all synapses, dendrites, and neurons utilize fully dedicated electronic circuits, so that each element of hardware has a one-to-one correspondence with its information-processing role in the neural system. This fully dedicated approach is advantageous if one aspires to create a diverse array of synaptic and dendritic behaviors at each neuron, as observed in biological neural systems (Marder, 1987; Euler and Denk, 2001). For instance, a different time constant or plasticity mechanism could be implemented at every synapse on a single neuron. Perhaps more importantly, fully dedicated components eliminate the auxiliary hardware required to perform multiplexing operations. Further, performing synaptic weighting and temporal dynamics in the electronic domain allows for binary optical communication, which minimizes the amount of optical energy per spike and reduces noise incurred by communication. The scope of this paper is therefore limited to networks meeting these three conditions:

1. Direct, optical connections are utilized for communication between neurons (fully dedicated axons).
2. Optical communication is binary. The amplitude of the optical signal carries no information.
3. All synaptic, dendritic, and somatic computations are performed by fully dedicated electronic circuits.

With these conjectures established, a picture of the hardware under consideration begins to emerge. There is a single optical transmitter at each neuron. This light emitter produces a short pulse of light each time the neuron spikes. The optical pulse is coupled into a waveguide, and optical power is tapped from the waveguide for each downstream synapse. Each synapse contains a photodetector which registers an all-or-nothing synapse event. From there, all synaptic weighting, spike-train filtering, dendritic processing, signal summation, neuronal thresholding, and plasticity mechanisms are implemented in the electronic domain with tailored integrated circuits. A schematic of this general framework is shown in **Figure 1**.

There are potentially multiple ways to physically implement this model. The remainder of this paper will discuss two possible implementations—a superconducting platform and a room-temperature all-semiconductor system. The superconducting platform, known as SOENs (Superconducting OptoElectronic Networks) is discussed in prior work (Shainline et al., 2017b, 2019; Shainline, 2019, 2021). In short, optical links are formed from semiconductor light sources and superconducting nanowire single photon detectors (SNSPDs). Computation is performed with analog Josephson junction (JJ) circuits and memory is implemented with persistent current in superconducting loops. The semiconductor implementation is imagined as an exact analog of the SOENs platform, except without the benefits (or limitations) of cryogenic elements. Traditional photodiodes enable optical communication, analog CMOS circuits provide computation, and emerging memory devices provide synaptic memory.

This paper seeks to analyze the suitability of both platforms for implementing large-scale optoelectronic neuromorphic networks. Despite limiting our discussion only to architectures meeting our three conjectures, there remains a vast space of design choices, making it difficult to draw hard-and-fast conclusions. Nevertheless, interesting guidelines can be obtained by analyzing limits of technologies most likely to be used in each platform. Important benchmarks for device performance are also identified, which may be of use in monitoring the development of this field.

2. COMMUNICATION

2.1. Optical Receivers

We begin analysis of optical interconnects with receivers. There are two ways the receiver influences the power budget of an optical link: (1) The receiver (and the electrical components it must drive) sets the minimum optical signal that must be produced by the light source, and (2) the receiver may require electrical power of its own to run. It is found that the energy per spike may be quite similar in both platforms once cooling is accounted for in the superconducting case. However, the optical power required from light sources is reduced by a factor of 1,000 in the superconducting case, at least when compared to the semiconductor receivers of comparable total efficiency, which omit transimpedance amplifiers (Miller, 2017).

2.1.1. Superconductor Receivers

As stated previously, the SOENs platform utilizes SNSPDs to detect optical signals as faint as a single photon. Physically, an SNSPD is a superconducting nanowire biased with a current source ($I_{spd} \approx 10\ \mu A$). The simple structure makes fabrication and waveguide integration straightforward (Sprengers et al., 2011; Pernice et al., 2012; Akhlaghi et al., 2015; Ferrari et al., 2015,

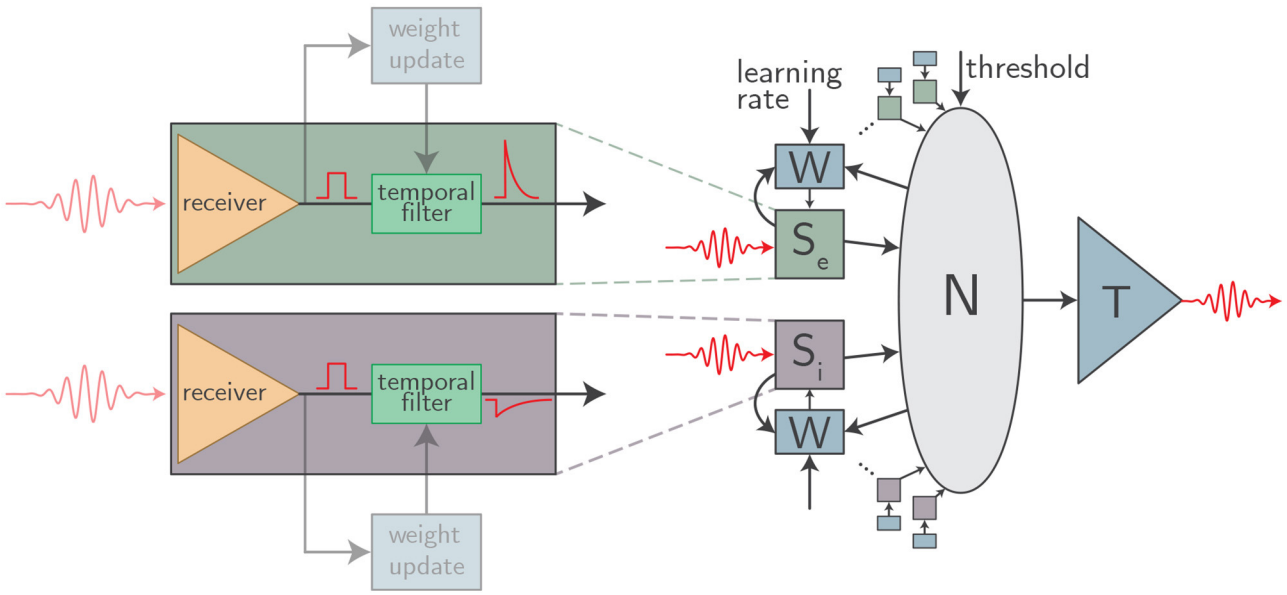


FIGURE 1 | An abstract schematic of the class of optoelectronic neurons meeting our three criteria. Each synapse (S_e and S_i for expiatory and inhibitory synapses, respectively) is implemented with a physical circuit block containing a detector and a temporal filter. The detector produces an all-or-nothing electrical pulse upon receipt of an optical spike which is then processed by the filter. The parameters of the filter (time constant, weight, etc.) can be set individually for each synapse. A local weight update circuit (W) implements plasticity mechanisms at each synapse. Synaptic outputs are integrated in the soma (N) which drives an optical transmitter to downstream connections upon reaching threshold.

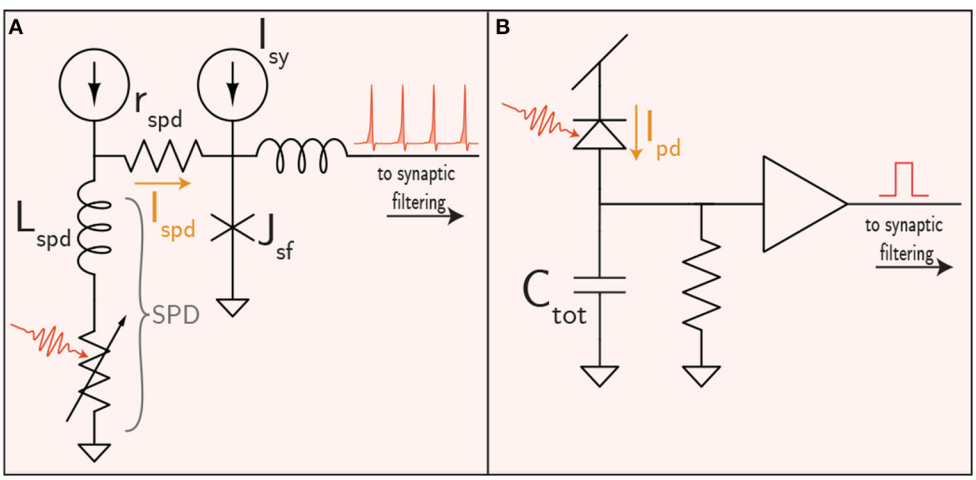


FIGURE 2 | Receivers for the **(A)** superconducting and **(B)** semiconducting platforms. Note that synaptic weighting for the semiconductor case is included in the filtering circuitry, shown in **Figure 4B**.

2018; Sahin et al., 2015; Shainline et al., 2017a; Buckley et al., 2020a). Photons traveling through a waveguide evanescently couple to a nanowire on the surface of the waveguide. A single photon has enough energy to drive the nanowire from the superconducting phase to a resistive state. In SOENs receivers, this momentarily redirects the bias current along an alternate conduction pathway that activates a JJ circuit to register the synapse event and conduct further synaptic processing (**Figure 2A**).

While an SNSPD itself dissipates zero static power, electrical power is still required for superconducting receivers. Current biases will require some power, but should be shared by many devices (section 3), ameliorating the cost. More important is dynamic electrical power consumption associated with detection events. The nanowire has an inductance, L_{spd}, that stores energy from the current bias. During a detection event, this energy is dissipated in the resistor r_{spd}. The electrical energy necessary to detect each photon is then $\frac{1}{2}L_{\mathrm{spd}}I_{\mathrm{spd}}^2$. L_{spd} can be as low as

100 nH, resulting in an electrical energy consumption (E_{spd}) of around 5 aJ/spike.

Since an SNSPD is capable of detecting single photons, it will operate near the quantum limit of optical communication (Razavi, 2012). We assume that the detection of a single photon will be treated as the registering of a synaptic event. The probability of a light source producing a spike with a certain number of photons within a fixed time window is given by a Poisson distribution. We will also conservatively assume a detection efficiency η_D of 70% (higher detection efficiency is certainly possible Marsili et al., 2013; Reddy et al., 2020). The probability of measuring zero photons during a spiking event is then given by:

$$P(0) = \sum_{k=0}^{\infty} \frac{N_{ph}^k e^{-N_{ph}}}{k!}(1-\eta_D)^k = e^{-N_{ph}\eta_D}, \tag{1}$$

where N_{ph} is the average number of photons per spiking event. Neural systems are known for remarkable robustness to and even utilization of noise (Stein et al., 2005; McDonnell and Ward, 2011). Detecting only 99% of spikes may be tolerable and would still represent a significant improvement over biology, wherein synapse reliability is typically in the range of 5–80% (Allen and Stevens, 1994; Lisman, 1997). From Equation (1), this would correspond to roughly 7 photons (0.9 aJ for $\lambda = 1.5\,\mu$m) needed to reach the receiver. The total number of photons produced by the source will need to be higher to account for energy losses in the link. The total optical energy per spike, E_{opt}, will be:

$$E_{\text{opt}} = \frac{N_{\text{ph}} h\nu}{\eta}. \tag{2}$$

$h\nu$ is the energy of a single photon and η is the total energy efficiency of the optical link. η includes all optical losses and the inefficiency of the transmitter. This efficiency factor will be highly dependent on the specifics of the platform, but for now we will leave it as a free variable. The total power consumed by the optical link is the sum of E_{opt} and E_{spd}. Accepting a 1% error rate, these two contributions to the total energy will be roughly equal when $\eta = 20\%$. Such a high efficiency is likely near the limits of physical possibility. For more realistic values of η, E_{opt} will dominate.

Importantly, superconducting electronics come with a cooling overhead (section 5). We conservatively assume that every watt of power produced at low temperature will require 1 kW of refrigeration power. System-level effective optical energy per spike for superconducting links will be no less than 1 fJ.

Fabrication of waveguide-integrated SNSPDs has become commonplace in recent years (Sprengers et al., 2011; Pernice et al., 2012; Akhlaghi et al., 2015; Ferrari et al., 2015, 2018; Sahin et al., 2015; Shainline et al., 2017a; Buckley et al., 2020a). SNSPD materials include NbN, NbTiN, WSi, and MoSi. Superconducting films (3–10 nm) can be sputtered at room temperature atop many substrates and patterned into wires from 50 to 5 μm wide using conventional lithography and etching. Multiple planes of SNSPDs have also been demonstrated (Verma et al., 2012)—a promising development for future large-scale neuromorphic systems (section 5). Waveguide-integrated NbN SNSPDs can reach photon count rates exceeding 1 GHz (Rosenberg et al., 2013; Vetter et al., 2016). However, slower detectors, such as MoSi and WSi SNSPDs with 20 MHz count rates, have demonstrated the best yields to date (99.7% Wollman et al., 2019). Previous statements that SOENs were limited to 20 MHz were motivated by these pragmatic concerns about the current state of fabrication (Shainline et al., 2019).

2.1.2. Semiconductor Receivers

While semiconductor receivers are the predominant technology for long-distance optical communication, intra-chip optical receivers deviate significantly from their long-distance counterparts, as traditional transimpedance amplifiers likely consume too much electrical power, despite impressive optical sensitivities. This has led to the proposal of "receiverless" designs that omit amplifiers altogether (Miller, 2017). Receiverless communication uses a photodetector to directly drive the input of CMOS gates. Photons produce electron-hole pairs in the photodetector, which in turn charge the CMOS gate capcitance up to the switching voltage. A circuit diagram of the scheme is shown in **Figure 2B** in which a photodiode directly drives a CMOS digital buffer. A resistor is also placed in parallel to allow the receiver to reset. In principle the resistor is unnecessary if an optical reset is used as described in Debaes et al. (2003). The resistor would increase the minimum optical power necessary to register a spike and limit the bandwidth of the receiver.

With optical link efficiency η, the necessary optical energy required to drive the receiver to a voltage V is Miller (2017):

$$E_{\text{opt}} = \frac{C_{\text{tot}} V}{\eta \mathcal{R}}. \tag{3}$$

$\mathcal{R}$ is the responsivity of the detector, typically of order 1 A/W. C_{tot} includes the photodiode capacitance, the CMOS gate capacitance, and any wiring capacitance. It is reasonable to consider values for C_{tot} at the femtofarad level. For 1.5 μm photons and a required voltage swing of 0.8 V, $E_{\text{opt}} \approx 0.7$ fJ (5000 photons) for unit efficiency. This is similar to the superconducting case, once cooling is considered. If two optical communications links were identical in all measures (source efficiency, optical losses, etc.) except one was cooled to 4 K with SNSPDs and the other operated at room-temperature with photodiodes, then communicating a spike would cost nearly the same energy at the system level in each link. The power required for cryogenic cooling pays for itself with reduced light levels in the optical link. Cooling semiconductor receivers to 4 K does not appreciably improve the situation, as the number of photons required in the receiverless case is related to charge, capacitance, and voltage, not thermal noise. For capacitances below 1 fF (a difficult task), semiconductor receivers could potentially consume even less energy than their superconducting counterparts. Waveguide-integrated femtofarad photodiodes have been demonstrated in both SiGe and Ge (DeRose et al., 2011). Polysilicon photodiodes are also attractive for increased manufacturability Mehta et al. (2014). Most photodiodes have far better speed than required for neuromorphic applications, reaching up to 45 GHz (DeRose et al., 2011).

Just as with SNSPDs, semiconductor receivers will also require electrical power, even if it is minimized by the receiverless approach. In this case, there will be static power dissipation through the leakage current of the photodiode. Assuming a 1 V bias, a leakage current on the order of 1 nA (Zhang et al., 2020a), and an optical link efficiency of 1%, this static dissipation would dominate power consumption for average spiking rates below 10 kHz. The development of low capacitance, zero-bias photodiodes (Nozaki et al., 2018) would be a major advantage toward making efficient, low frequency networks. Static power consumption is also a major question for many avalanche photodiode (APD) receivers. Avalanche gain could provide a significant (at least one order of magnitude) reduction in the necessary optical power per spike (Miller, 2017). While often associated with higher bias voltages, germanium waveguide-integrated avalanche detectors have been demonstrated to provide 10 dB of gain even at 1.5 V bias (Assefa et al., 2010). However, dark current is still typically in the microamp range for such detectors (Assefa et al., 2010; Virot et al., 2014), meaning that brain-scale networks are likely out of reach due to power constraints (section 5). APDs may be of interest in smaller, faster spiking networks, however. Another intriguing possibility is to reduce static power consumption through cooling, as the dark current could potentially be reduced by orders of magnitude (Pizzone et al., 2020). However, in that case one forfeits a major advantage of the semiconductor approach.

While the receiverless scheme is promising for achieving low energies per spike, it places significant burden on the transmitter side of the link. Neuromorphic applications magnify this burden, as neurons are expected to drive thousands of downstream connections in parallel. Additionally, the receiver capacitance must be charged quickly to maintain high spiking frequencies. The result is that relatively large optical power is required from transmitters. The best case ($\eta = 1$) scenario is shown in **Figure 3**. Semiconductor receivers can be expected to require around one thousand times the optical power of superconducting receivers and the highest spiking frequency of a neuron could very well be limited by the power output of the light source. The ramifications of this result on prospective light sources are discussed in the next section.

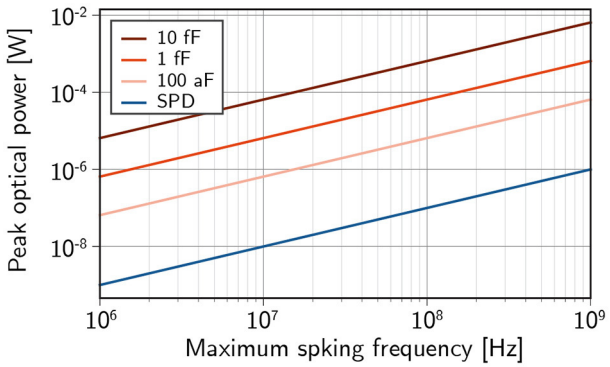


FIGURE 3 | The required optical power to drive 10^3 downstream synapses within one inter-spike interval for a given spiking frequency assuming receiverless photodiodes with optical link efficiency $\eta = 1$.

2.2. Optical Transmitters

The transmitter is expected to dominate the power budget of optical links for both platforms. Room-temperature, CMOS-integrated light sources have been a holy grail for decades, but materials integration issues have kept this prized objective out of reach. For superconducting systems, SNSPDs drastically lower the power requirements of light sources, while cryogenic temperatures improve light source efficiency. Light sources are likely significantly simpler in the superconducting case. However, interfacing low-voltage superconducting electronics with semiconductor light sources (McCaughan et al., 2019) presents an obstacle that is absent from the all-semiconductor platform.

2.2.1. Integrated Light Sources

Optical coherence is not a requirement for the envisioned system. NanoLEDs are thus an attractive option due to their ease of fabrication, lack of threshold current, and improving efficiency with shrinking scale (Romeira and Fiore, 2019). However, nanoLEDs struggle to produce optical power significantly greater than 1 μW Romeira and Fiore (2019). While semiconductor systems targeting spiking frequencies in excess of 1 MHz may be forced to turn to lasing, nanoLEDs should be more than sufficient for superconducting platforms. Either way, integrating millions of light sources on a 300 mm wafer remains highly challenging. The indirect band gap of silicon drastically reduces light emission. Off-chip light sources are used in some applications, but are likely untenable for massive systems, as their high static power consumption is incommensurate with the sparsity of neural activity. Integrated light sources would be a tremendous boon, if not a requirement for the success of large-scale optoelectronic neuromorphic computing. There are two courses of action: (1) force silicon to emit light through either material and/or environmental modifications or (2) integrate direct bandgap materials on silicon.

Many strategies toward silicon light sources have been pursued (Iyer and Xie, 1993; Shainline and Xu, 2007) including quantum confinement in Si-based superlattices (Warga et al., 2008) and nanocrystals (Walters et al., 2005), emission from embedded erbium (Ennen et al., 1985; Palm et al., 1996), point-defect emitters (Brown and Hall, 1986; Bradfield et al., 1989; Bao et al., 2007; Rotem et al., 2007), extended defects (Ng et al., 2001), strain dislocations (Kveder et al., 2004), and engineering of the local density of optical states (Green et al., 2001). Total efficiency from 0.1% (Kveder et al., 2004) to 1% (Green et al., 2001) has been demonstrated at room temperature, but not at powers and areas suitable for the semiconductor receivers introduced in the previous section.

Abandoning silicon as an active optical element, many researchers turned toward epitaxial germanium grown on Si (Sun et al., 2009c). Like silicon, germanium is an indirect-gap semiconductor. However, the direct gap is only 136 meV higher than the indirect gap, and clever implementation of strain (Ishikawa et al., 2003; Ghrib et al., 2012; Tani et al., 2021) and heavy n-type doping (Liu et al., 2007; El Kurdi et al., 2009; Sun

et al., 2009a; Camacho-Aguilera et al., 2013; Virgilio et al., 2013) can lead to appreciable direct, radiative recombination. These efforts have led to Ge-on-Si lasers (Sun et al., 2009b; Liu et al., 2010), but it has proven difficult to reduce the threshold current and increase device efficiency. Another approach is to grow SiGe with a hexagonal lattice on GaAs, leading to a direct gap (Fadaly et al., 2020), but this does little to solve integration problems.

At present, neither Si nor Ge emission has proved satisfactory for the needs of digital communication, so integrating III-V materials on silicon substrates has received significant attention. Pending a watershed moment in silicon sources, III-V integration will be required for the semiconductor platform (although not necessarily in the superconductor case, where low-temperature changes the physical context). Epitaxial growth would be an attractive solution for III-V integration due to the high throughput (Norman et al., 2018), but defects due to lattice mismatch have so far prevented this method from large-scale adoption. III-V quantum dots are more robust to such defects and have demonstrated high optical powers with small footprints (Chen et al., 2016; Jung et al., 2017; Norman et al., 2018), albeit typically grown on offcut Si substrates that are not CMOS compatible or with thick buffer layers that make optoelectronic contact difficult. More work is required to realize scalable, cost-effective integration of III-V quantum dot light sources with CMOS electronics, passive photonic waveguides, and efficient photodetectors. Without epitaxial growth, the semiconductor platform would be less scalable due to the limited size of III-V wafers and the expense of performing wafer bonding. A variety of schemes have been proposed (Norman et al., 2018; Tang et al., 2019), including die-level bonding (Song et al., 2016; Crosnier et al., 2017), wafer-level bonding (Hu et al., 2019; Szelag et al., 2019; Jiao et al., 2020), transfer printing (Justice et al., 2012; Zhang et al., 2018a, 2019), and selective-area epitaxy (Han et al., 2021), but these approaches still appear cumbersome when seeking the scale of integration considered here.

The situation is significantly more favorable for cryogenic systems. Low temperature often reduces non-radiative recombination (Sandiford, 1958; Gurioli et al., 1991; Dolores-Calzadilla et al., 2017), improving efficiency for both silicon and III-V light sources. The case of Ge at low temperature is more subtle due to the pecularities of the pseudo-direct gap and inter-valley scattering that is more prevalent at higher temperatures (Sun et al., 2009c). The benefits are further compounded by the low optical power requirements of SNSPDs. When integrating III-V light sources with CMOS, the light sources must be integrated on top of the electronics after the high-temperature dopant activation steps have been performed. Superconductor electronics have no such high-temperature processing steps, so the light sources can be produced on a Si wafer before the electronics are realized. Problems related to offcut Si wafers and thick buffer layers are eliminated. Additionally, silicon light sources, with their superior potential for integration, demand exploration with the superconducting platform. Several silicon point defects typically quenched at room-temperature emerge as narrow-linewidth candidates for light sources in the telecommunications band (Davies, 1989; Sumikura et al., 2014; Buckley et al., 2017; Beaufils et al., 2018; Chartrand et al., 2018). While single-photon emission (Bergeron et al., 2020; Hollenback et al., 2020; Redjem et al., 2020) is not the objective in the present context, the narrow linewidth is also attractive for further efficiency gains via the Purcell Effect (Romeira and Fiore, 2018). LEDs have already been demonstrated with the W-center defect (Bao et al., 2007; Buckley et al., 2017), albeit with poor (10^{-6}) efficiencies, limited by electrical injection efficiency rather than emitter lifetime. Photoluminescence studies are promising for orders of magnitude improvement (Buckley et al., 2020b), but more work is required to improve emission efficiency in an integrated-circuit context. If cryogenic silicon light sources become viable, the superconducting platform might hold a major scalability advantage over the semiconducting analog.

2.2.2. Driving Circuitry

Both platforms require neurons to drive semiconductor light sources. The transmitter circuitry is thereby required to produce voltages on the scale of the bandgap of the optical source ($\approx$ 1 V). CMOS circuitry, itself a semiconducting technology, naturally operates on this voltage, rendering the driving circuitry a non-issue. Standard MOSFET LED or modulator driving circuits (Halbritter et al., 2014; Bowers et al., 2016) can be straightforwardly adapted for neuromorphic applications. Superconductors, however, operate in an entirely different regime, with signals usually on the order of the superconducting energy gap ($\approx$ 1 mV). The optimal method for interfacing superconducting electronics with semiconductor devices is still an area of active research. Recent progress has been made with devices utilizing the massive change in impedance during a phase transition between superconducting and resistive states. In McCaughan et al. (2019), a resistive element was heated using 50 mV pulses to thermally trigger a transition in a superconducting meander. The meander transitioned to a state with resistance in excess of 10 MΩ and was used to drive a cryogenic silicon light source waveguide-coupled to an SNSPD. While these results are promising, the light source was only pulsed at 10 kHz (due to poor source efficiency) and was fabricated on a separate chip. More work is needed to improve the speed, efficiency, and to monolithically integrate driving circuitry with LEDs.

3. ELECTRONIC NEURONAL COMPUTATION

Electronic circuitry capable of performing neuronal dynamical operations will also be necessary. Biological neurons are increasingly recognized as sophisticated computational units (Koch and Segev, 2000; Stuart and Spruston, 2015; Hawkins and Ahmad, 2016; Sardi et al., 2017). Emulating such complicated behavior has been the subject of extensive investigation in both semiconducting (Vogelstein et al., 2007; Indiveri et al., 2011; Brink et al., 2013; Pfeil et al., 2013; Benjamin et al., 2014; Abu-Hassan et al., 2019) and superconducting platforms (Crotty et al., 2010; Shainline, 2019; Toomey et al., 2019). We do not attempt a comprehensive review of circuitry, but rather draw attention to issues specific to optoelectronic networks in both cases.

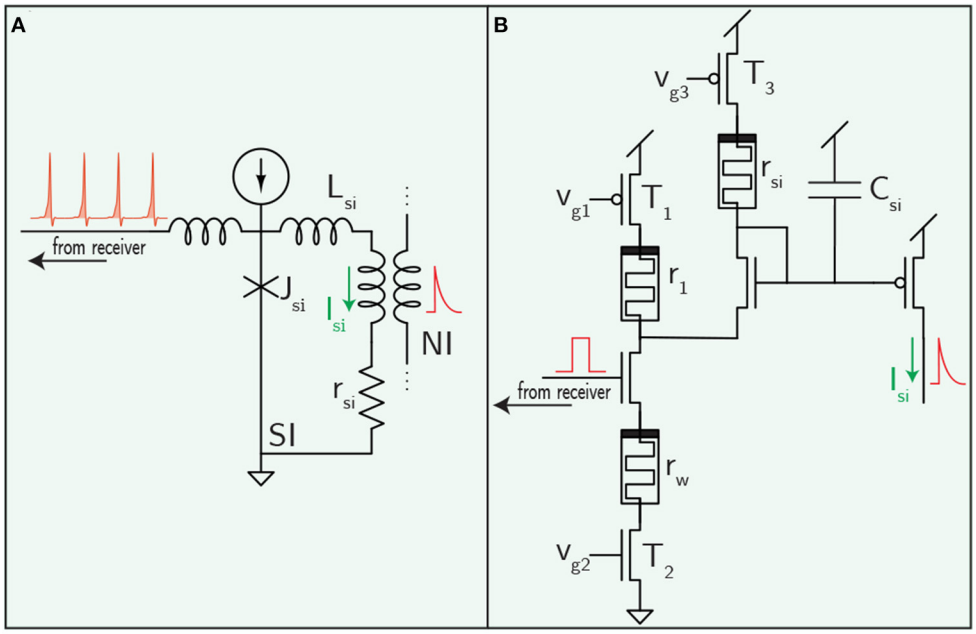


FIGURE 4 | Synaptic filtering circuits for the superconductor **(A)** and semiconductor **(B)** cases. Weighting in the superconducting case was shown in **Figure 2**. The memristor-integrated DPI circuit pictured here is introduced in Ref. (Dalgaty et al., 2019).

3.1. Semiconductor Electronics

The maturity of CMOS processing has allowed great strides in neuromorphic computing. While optical communication would likely also be advantageous in digital approaches, we focus on analog CMOS neurons for their perceived efficiency advantages (Mead, 1990; Rajendran et al., 2012). At a basic level, a neuron must perform three mathematical functions: summation of synaptic inputs, temporal filtering, and threshold detection leading to action potential generation. Summation can be achieved by exploiting Kirchoff's current law. Filtering can be implemented with elementary resistor-capacitor circuits. Thresholding is a natural function of transistors. Building upon this basic mapping, analog neurons have demonstrated a litany of biologically-inspired models (Indiveri et al., 2011; Liu et al., 2015).

It was found in the previous section that optical communication requires a minimum of about 1 fJ of energy to deliver a spike signal to each synapse. For realistic optical link efficiencies, this value will be at least an order of magnitude larger. Synaptic processing circuits would therefore ideally operate with an energy budget of 10–100 fJ to process a single spike. Somatic computation could comfortably consume power larger than that of synaptic processing by a factor of the average fan-out (perhaps 1,000). Many low-energy neuromorphic demonstrations are promising for reaching these targets. By reducing the membrane capacitance and supply voltage, a neuron capable of 25 kHz spike rates was demonstrated to consume only 4 fJ/spike (Sourikopoulos et al., 2017). Many other analog neurons, with energies ranging from femtoJoules to picoJoules per spike, fall comfortably below the power consumption of optical communication (Indiveri and Sandamirskaya, 2019). However, it remains to be seen if more complicated neurons and synapses, implementing a critical subset of behavior necessary for cognition, will be able to maintain such low power operation. In terms of speed, CMOS neurons have demonstrated spike rates in excess of 100 MHz (Schemmel et al., 2017). Optical communication should face few issues achieving such speeds, *if* sufficiently bright light sources can be efficiently integrated with CMOS circuits.

One challenge for the CMOS approach has been to design compact circuits with long time constants. Long time constants are important for systems targeting biological time scales (upwards of 500 ms) (Indiveri and Sandamirskaya, 2019) or power-law distributions of timescales to implement critical behavior (Beggs, 2007). Subthreshold transistor circuits operating with currents in the femtoamp to picoamp range minimize the size of capacitor needed to implement a specific time constant (Indiveri et al., 2011). The area constraints of this scheme are discussed in **Supplementary Information A** and compared to the superconducting approach.

For a concrete example, a circuit diagram for a memristor implementation of the popular differential-pair integrator (DPI) synapse is shown in **Figure 4B** (Dalgaty et al., 2019). The DPI produces a decaying exponential post synaptic signal in response to an input voltage pulse—potentially from an optical receiver. This leaky integrator behavior is characterized by a time constant set by the value of the filtering capacitance and the rate of leakage off the capacitor (Chicca et al., 2014). The time constant could potentially be programmed using memristors—an advantage over superconducting circuits that have been proposed to date.

3.2. Superconducting Electronics

Superconducting neurons have been studied nearly as long as CMOS implementations, with a mapping between neuronal functions and superconducting electronics identified in the early 1990s (Harada and Goto, 1991; Hidaka and Akers, 1991). In this case, Faraday's Law, governing the addition of magnetic flux through mutual inductors to superconducting loops provides the necessary synaptic summation function. Filtering is achieved through resistor-inductor blocks (or RC circuits in some cases Crotty et al., 2010). Josephson junctions (JJs) provide the requisite nonlinear thresholding element.

Like their CMOS counterparts, many superconducting circuits have now been designed to implement sophisticated neuronal dynamics. Superconducting neuromorphic circuits have been designed to implement a variety of bio-inspired neuron models (Crotty et al., 2010; Schneider et al., 2018a; Toomey et al., 2019), dendritic processing (Shainline, 2019), and have performed image classification in simulation (Schneider et al., 2017). The natural spiking behavior of JJs may even require a lower device count than analogous CMOS circuits for various leaky-integrate-and-fire models (Crotty et al., 2010). In short, it does not appear that superconducting circuits are any less capable of complex neuronal computation than CMOS, although experimental demonstrations lag far behind.

Superconducting electronics has long been pursued for gains in energy efficiency and speed. Indeed, superconducting elements dissipate zero static power and spike energies are frequently reported in the sub-femtojoule range, including refrigeration. Optical communication is likely to dominate power consumption for superconducting optoelectronic systems (**Supplementary Information B**). In terms of speed, fully electronic superconducting neurons may be capable of spike rates up to 100 GHz (Schneider et al., 2017, 2018a). However, this is orders of magnitude faster than any SNSPD can respond. This speed disparity is a notable difference between the superconducting and semiconducting architectures. While optical communication could be integrated with CMOS neurons with no degradation in speed, optoelectronic superconducting systems will likely be significantly slower than their fully electronic counterparts. This may be the cost of highly connected systems. That said, the extraordinary switching speed of JJs is still leveraged in optoelectronic networks to perform analog computations within synapses, dendrites, and neurons.

The ability of superconducting electronics to go slow might be just as compelling as their ability to go fast. While it can be challenging to implement long, biologically realistic time constants in CMOS neurons, superconducting loops can create time constants orders of magnitude higher than biology by adjusting the L/R ratio in synaptic and neuronal loops (See **Figure 4A** and **Supplementary Information A**). The ability to generate dynamics across many orders of magnitude in time also dovetails nicely with suggestions that critical behavior is important for cognition (Cocchi et al., 2017).

Fan-in has traditionally been considered a liability of superconducting electronics. If this were the case, it would clearly be an impediment to mature superconducting neuromorphic systems. For superconducting neurons designed to use single fluxons as synaptic signals, fan-in has recently been analyzed (Schneider and Segall, 2020), and it has been found that if a single synapse must be able to drive a neuron above threshold, fan-in may be limited to around 100. However, it is often not necessary for each synapse to be able to trigger a neuronal spike event. It has been analyzed elsewhere that if analog signals containing multitudes of fluxons are communicated from synapses to the neuron cell body, fan-in can likely scale to biological levels through the use of mutual inductors (Shainline et al., 2019). Using more fluxons comes with larger power consumption, but for optoelectronic systems, light production will likely still dominate.

While most diagrams of superconducting circuits (including those here) show many separate biases delivering current to various elements, the ability to construct circuits that can be biased in series will be critical to the scalability of this hardware. A separate bias for every synapse would be untenable in large-scale systems (Tolpygo, 2016). This mimics the evolution that occurred in superconducting digital electronics, in which the field has turned away from parallel biasing schemes and embraced serially biased platforms (Tolpygo, 2016) and current recycling schemes (Kirichenko et al., 2011). SOENs are potenially amenable to serial biasing, but this important point demands further analysis.

A superconducting synaptic filtering circuit is shown in **Figure 4A**. Synaptic weighting is implemented in the receiver circuit (**Figure 2A**), so this circuit block is only responsible for converting a train of fluxons into a decaying exponential post-synaptic potential reminiscent of biological and CMOS synapses. A resistor, r_{si}, converts a superconducting persistent current loop into a leaky-integrator in a similar manner to the DPI synapse. The time constant is set by L_{si}/r_{si}, and the synaptic current can be added to a neuronal circuit through mutual inductors. Unlike the DPI synapse, this circuit does not have a programmable time constant, but does hold the potential to implement a wide range of different time constants by fabricating different values of L_{si} and r_{si}.

4. SYNAPTIC MEMORY

It has been apparent to the neuromorphic community for some time that large-scale neural systems will require innovative approaches to synaptic memory. A local, analog memory element unique to every synapse will provide the most efficient performance by eliminating memory retrieval and digital conversion. Important metrics for analog synaptic memory technologies include weight precision, volatility, area, write energy, write speed, and endurance (the effective number of cycles in a device's lifetime). We attempt to provide desired benchmarks for a few of these metrics in the specific case of optoelectronic networks. For this section, we assume a speedup of about 10^4 over biology, for an average spike rate of 10 kHz and a maximum of 10 MHz. This is commensurate with both the maximum count rates of high-yield SNSPDs and some of the fastest CMOS electronic neuromorphic systems built to-date.

4.1. Memory Benchmarks

4.1.1. Endurance

Large-scale neural systems require significant investments in money and time. Operational lifetimes on the scale of decades (10^9 s), if not longer, are therefore essential. Such systems

will be expected to learn continually during that lifespan, placing significant requirements on the durability of memory technologies. The number of times a synapse is updated in its lifetime is a function of neuron spiking frequency (f) and the number of synapses that are typically updated after each post-synaptic spike. Neuroscientific evidence has been presented that the number of active presynaptic inputs required to trigger a postsynaptic spike goes as $\sqrt{N}$, where N is the fan-in of the neuron—exceeding 1,000 for brain-like systems (van Vreeswijk and Sompolinsky, 1996; Vogels et al., 2005). We assume all synapses that contributed to the spiking of the post-synaptic neuron are updated with each spike. We then estimate the number of weight updates (N_{update}) in the synapses's lifetime (L) will be:

$$N_{\text{update}} = \frac{Lf}{\sqrt{N}} \tag{4}$$

For a decades-long lifetime, and a mean spiking frequency of 10 kHz, the total number of weight updates will be 10^{11}. This is a challenging demand for many emerging non-volatile memory technologies.

4.1.2. Update Energy

One would like the power dedicated to weight updates not to exceed the power used for optical communication. Once again invoking the assumption that $\sqrt{N}$ synapses are updated with each postsynaptic spike, we arrive at the following relation between the energy to produce a single spike (E_{opt}) and that to update a single weight (E_{update}):

$$E_{\text{update}} < \sqrt{N} E_{\text{opt}} \tag{5}$$

Using the analysis in section 2, 1 fJ of energy needs to be delivered to the receiver in either platform. Assuming a transmitter efficiency of 1%, this would mean E_{opt} is 100 fJ. Therefore, for a fan-in of 1,000 synapses, E_{update} would ideally be no more than about 3 pJ. This value includes any energy consumption of peripheral circuitry, both static and that associated with programming. This efficiency appears to have already been met by several emerging memory technologies (Schneider et al., 2018b; Zahoor et al., 2020).

4.1.3. Update Speed

An ideal system would be capable of implementing synaptic updates within the minimum inter-spike interval. While semiconductor optoelectronic systems could potentially produce spike rates in excess of 10 GHz (assuming sufficiently bright, integrated light sources can be achieved), synapses might need to be taken offline during WRITE operations, as it is unlikely that sophisticated plasticity mechanisms can be implemented in under 100 ps. Lower maximum frequencies would allow plasticity to be implemented without ever neglecting a spiking event. For our 10 MHz target, we desire memory updates in under 100 ns. Slower updates may not be completely intolerable, if network dynamics are robust to missed spikes during synaptic updates or to synaptic weights that are in the process of being altered.

TABLE 1 | List of desired performance metrics for synaptic memory in a system with average fan-out of 1,000, maximum spike rate of 10 MHz, average spike rate of 10 kHz, and spike energy of 100 fJ.

Metric	Goal
Endurance	$> 10^{11}$ updates
Update Energy	< 3 pJ
Update Speed	< 100 ns
Weight Precision	4-8 bits

4.1.4. Weight Precision

The necessary weight precision will be determined by the specifics of a chosen learning model and the desired application. Weight precision has been the subject of much discussion. It has been suggested that 4-bit precision is sufficient for state-of-the-art mixed signal neuromorphic systems (Pfeil et al., 2012). Deep learning systems have also demonstrated success with 8-bit precision—a significant reduction from 32-bit floating point numbers (Wang et al., 2018). Hippocampal synapses in rats have been inferred to allow at least 26 different states ($\approx$ 5 bit), which squares nicely with computer science findings (Bartol et al., 2015). It has also been argued that metaplasticity mechanisms are more important for lifelong learning than the bit-depth of the synapse (Fusi et al., 2005; Fusi and Abbott, 2007).

Target values for these key synaptic memory metrics are summarized in **Table 1**.

4.1.5. Programming Signals

One important criterion that eludes quantitative benchmarking is the complexity of programming circuitry for synaptic memory. Significant infrastructure for producing programming signals could limit scalability. For example, floating-gate synapses often require programming signals at significantly higher voltages than are likely to be used in other parts of the network. For large-scale systems, memories with simple programming requirements will be at an advantage. Superconducting loop memory (section 4.2.4) is intriguing from this standpoint, as the plasticity circuits operate with nearly identical signals and circuit blocks as those found in the rest of the network.

4.2. Proposed Technologies

4.2.1. Room-Temperature Analog Memories

Many technologies have been proposed to implement synaptic weighting for room-temperature neuromorphic hardware, each with strengths and weaknesses (Upadhyay et al., 2019). The quest to find a suitable device for local synaptic memory dates back to the origins of the field, when Mead and colleagues investigated floating gate transistors (Diorio et al., 1998). Since then, floating gate synapses have been used to implement STDP (Ramakrishnan et al., 2011), are attractive as a mature alternative to emerging devices, and have been proposed for use in large-scale systems (Hasler and Marr, 2013). However, there are concerns about high programming voltages, speed, and endurance that may limit floating-gate memories to situations with less-frequent updates. More recently, momentum has

shifted to other technologies (Zahoor et al., 2020). Memristive devices (Strukov et al., 2008; Yang et al., 2012; Abraham, 2018), commonly used in resistive random-access memory have emerged as a popular alternative, with recent demonstrations including monolithic integration with CMOS (Yin et al., 2019) and unsupervised pattern recognition with a simple network of synapses (Ielmini, 2018). Questions remain about high variability (both cycle-to-cycle and device-to-device) (Dalgaty et al., 2019), linearity, and endurance (Zahoor et al., 2020). Phase-change memory is another option, with its own demonstration of STDP (Ambrogio et al., 2016). Thermal management and endurance have been raised as issues (Upadhyay et al., 2019; Zahoor et al., 2020). Ferroelectric transistors present another alternative, as they have low variability, good potential for CMOS integration, and linearity (Kim and Lee, 2019). Spin-torque memory, 2D materials, and organic electronics have also been proposed as solutions. Interested readers should consult one of the many review articles on this topic (Kim et al., 2018; Upadhyay et al., 2019; Zhang et al., 2020b). The field is burgeoning with new devices for synaptic memory, but to-date none has been dominant enough to monopolize research. To our knowledge, no technology has been able to simultaneously meet the targets in **Table 1**, but progress in this area is encouraging.

4.2.2. Superconducting Technologies

Many of the aforementioned technologies may also apply to superconducting optoelectronic systems, but their cryogenic operation has been scarcely explored. Two other types of memory, only accessible at low temperatures, have received the most attention for superconducting systems: magnetic Josephson junctions (MJJs) and superconducting loop memories. An important distinction from room-temperature technologies is that for superconducting memory to be truly non-volatile, it must retain its state both in the absence of a power supply and upon warming to room-temperature.

4.2.3. Magnetic Josepson Junctions

MJJs have been proposed as a (nearly) non-volatile memory technology for superconducting neuromorphic computing. A two-terminal device, the critical current of an MJJ can be programmed by changing the magnetic order of a ferromagnetic material placed in the tunneling barrier of a JJ (Schneider et al., 2018b). MJJs are non-volatile with respect to electrical power, and there is optimism they can be made to retain their memory through a warm-up to room-temperature. Additionally, they provide remarkable performance with respect to the metrics given in **Table 1**. The energy per update is on the order of femtojoules (including cooling overhead), switching speeds are commensurate with firing rates exceeding 100 GHz, and devices can be scaled to tens of nanometers. All of these metrics surpass the requirements for optoelectronic networks, and can be exploited in all-electronic superconducting networks as well (Schneider et al., 2018a). More work is needed to analyze the scaling potential of MJJs with respect to yield. The magnetic fields used during programming can be produced with magnetic control lines, but spin-torque mechanisms may provide a more scalable solution. Finding an efficient, scalable solution to programming MJJs in large-scale systems thus remains an area of research that will be critical to their potential for adoption.

4.2.4. Loop Memory

Superconducting loop memories have been in use for decades by the superconducting electronics community (Duzer and Turner, 1998; Kadin, 1999), but are not ideal for dense memory arrays commonly utilized as RAM in digital computing due to area concerns. In the case of optoelectronic spiking neural systems considered here, the objective is not to produce large RAM arrays, and size as well as addressing challenges do not emerge as significant impediments. Therefore, straightforward extensions of binary loop memories are the synaptic memory technology that appears most promising for the SOENs platform (Shainline et al., 2018, 2019). In these memory cells, circulating current persists indefinitely in a loop of superconducting wire. The current in the loop can be controlled by adding/removing magnetic-flux quanta with standard JJ circuitry. This memory loop is then inductively coupled to a wire supplying a bias current to a Josephson junction at the synapse (J_{sf} in **Figure 2A**). When the synaptic SNSPD detects a photon, the biased junction will add an integer number of fluxons to another integrating superconductive loop (analogous to the membrane capacitance of a neuron). The number of fluxons added to the integration loop is a function of the bias supplied to the JJ, which is determined by the magnitude of current circulating in the memory loop. The number of analog memory levels in the memory loop is determined by the inductance of the loop, which is easily set with the length of a wire. High-kinetic-inductance materials (Tolpygo et al., 2018) enable memory storage loops with over a thousand levels (10 bits) to be fabricated in an area of $5\,\mu\text{m} \times 5\,\mu\text{m}$.

The loop-memory approach has several strengths. The memory is nearly analog and updates are nearly linear. Memory is updated by the switching of a JJ, which involves only a change of the phase of the superconducting wave function. This phase can switch 10^{11} times in a second, so the endurance metric defined in the previous section is not an issue. This stands in contrast to room-temperature memories requiring material changes (filament formation, phase changes, etc.) which are often associated with degradation over time. Loop memory is also attractive from a fabrication perspective as it requires no additional materials or devices. The simplicity of the memory lends itself favorably to 3D integration, provided cross-talk from nearby loops can be mitigated. Plasticity circuits based on loop memories will also operate at the energy scale of single photons and flux quanta (10^{-19} J), which is commensurate with the rest of the circuitry in the network. This allows weight updates to be performed with the spikes the network produces in standard operation, reducing peripheral circuitry. There is no need to engineer differently shaped pulses for READ and WRITE operations, and the synapse does not need to be taken offline during programming. Simulations have demonstrated STDP learning with circuits containing four additional Josephson junctions (Shainline et al., 2019).

Two aspects of loop memory are concerning. First, loop memory is not strictly non-volatile. While circulating current

can persist in a superconducting loop without any power supply, superconductivity must be maintained. If the temperature of the system is raised above the critical temperature of the superconducting material, the memory will be lost. Mechanisms for transferring weights stored in current loops to non-volatile solutions will need to be developed if the system's state is to be persevered upon reaching room-temperature (i.e., for maintenance or during a power interruption). The second weakness of loop memory is the size. The employed superconducting loops, as well as the transformers that couple them, will be large compared to all of the other solutions discussed. The consequences of these large-area components must be considered in the context of the entire system, which we discuss next.

5. SYSTEM LEVEL CONSIDERATIONS

Here we consider aspects concerning the integration of the components previously discussed and how systems may reach the scale of the brain. Basic graph theory metrics and the assumption of 300-mm fabrication processes allow us to assess area constraints and the benefits of 3D integration. It is found that at least five planes of photonic routing will be required in either platform to achieve brain-scale systems. Prospects for 3D integration of active elements are addressed. It also must be stressed that an optoelectronic system of the complexity of the human brain will be abjectly impossible on a single 300-mm wafer in either case. A possible vision for connecting many wafers is discussed. Finally, we analyze cooling and power concerns, finding that neither should preclude the development of brain-scale systems in either platform.

5.1. Considerations From Graph Theory

Neurons in brain regions active in cognition, such as the cerebral cortex and hippocampus, are characterized by a high degree of connectivity—often in excess of ten thousand connections per neuron (Braitenberg and Schuz, 1998; Buzsáki, 2006). These connections often extend across appreciable spatial distances. Creating and maintaining these connections comes with high metabolic and spatial costs. The severely constrained biological brain would not support such expenditures if they were not advantageous to cognition (Bullmore and Sporns, 2012).

One reason why such high connectivity is necessary relates to efficient communication across the network. Rapid communication can only be achieved if the average path length across the network is small. In the language of graph theory, a network is a collection of nodes connected by edges. To calculate the shortest average path length across the network, one calculates the number of edges that must be traversed to travel from one node to another node in the network. One takes the mean of this quantity over all pairs of nodes. The shortest average path length ($\bar{L}$) is a global metric that offers a glimpse at the efficiency with which information can be communicated across space.

Equation 6 provides the relationship between $\bar{L}$ and the number of edges connected to a node, or in our case, the number of synapses per neuron ($\bar{k}$) for a random network. In a random network, nearby and distant connections are equally probable. Specifically, the equation holds for Erdös-Rényi random graphs of networks with N_{tot} neurons (Fronczak et al., 2004):

$$\bar{k} = \exp\left[\frac{\ln(N_{\mathrm{tot}}) - \gamma}{\bar{L} - 1/2}\right], \tag{6}$$

where $\gamma \approx 0.5772$ is Euler's constant. For a network with 10^6 neurons, each neuron must make nearly 10,000 connections to support an average path length of two, and 200 synapses must be formed to support a path length of three. For a network with 10^8 neurons, more than 100,000 synapses are required for a path length of two, and more than 1,000 for a path length of three. The human hippocampus is a module with roughly 10^8 neurons, each with 10,000–50,000 nearly spatially random connections. The objective of achieving an average path length between two and three may be an important reason why the hippocampus prioritizes this exceptional degree of connectivity (Buzsáki, 2006). The cerebral cortex in the human brain contains more than 10^{10} neurons, each with roughly 10,000 connections. This analysis indicates that a path length between two and three cannot be achieved across the entire cortex, and accordingly the cortex is constructed with a hierarchical, modular architecture (Simon, 1962; Meunier et al., 2010) with high connectivity and efficient communication within smaller modules, and more sparse connectivity between modules separated by larger distances (Mountcastle, 1997; Meunier et al., 2010; Bota et al., 2015; Betzel and Bassett, 2017).

While more sophisticated graph metrics can further elucidate the network concepts underlying cognition (Bullmore and Sporns, 2009), the simple, global metric of average shortest path length can help inform scaling analysis of artificial cognitive hardware at this early stage of development. We next consider the constraints $\bar{L}$ puts on the size of synaptic circuits.

5.2. Generic Spatial Constraints

Based on the significance of the interplay between the hippocampus and cerebral cortex in cognition (Friston and Buzsáki, 2016), we assume hardware for artificial neural systems will make use of similar architectural principles. Here we assume optoelectronic circuits will be fabricated using the conventional sequential, planar processing techniques of the silicon microelectronics industry. Photonic planes will implement the passive optical interconnects and electronic planes will accommodate all active electronics for neuronal function. We further specify to consideration of 300-mm wafers and seek a relationship between the network path length and the size of components on the wafer.

The area of a neuron occupied by its photonic waveguides can be approximated in a similar manner to the wires for electronic circuits (Keyes, 1982). This gives the following expression for the area of passive photonic circuitry:

$$A_p = \left(\frac{k w_{\mathrm{wg}}}{p_p}\right)^2. \tag{7}$$

p_p is the number of photonic waveguide planes, k is the degree of each neuron (assumed identical), and w_{wg} is the pitch of

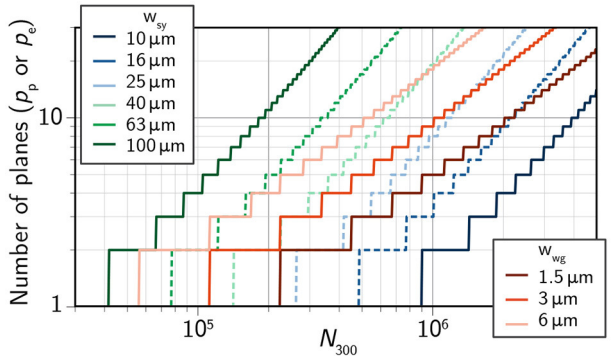


FIGURE 5 | Number of planes of active synaptic circuits (p_e) and passive photonic waveguides (p_e) required to maintain a path length of 2.5 as a function of the number of neurons on a 300-mm wafer (N_{300}).

waveguides. The area of a neuron due to electronic synaptic circuits is given by

$$A_e = \frac{k w_{sy}^2}{p_e}. \tag{8}$$

w_{sy} is the width of a synapse and p_e is the number of planes of electronic circuits. Both $N_{tot}A_p$ and $N_{tot}A_e$ are subject to the area constraint of a 300- mm wafer. We use these relations to calculate the number of planes (electronic and photonic) that will be required to maintain a path length of 2.5 across a network of a given size (**Figure 5**). See Appendix C for analysis of path length dependence on w_{sy} and w_{wg}. A specific routing scheme is analyzed in reference (Shainline et al., 2019). More than 10 million neurons (less than a mouse brain) on a single 300-mm wafer appears out of reach for any platform.

5.3. Fabrication Processes

We assume 300-mm silicon wafer processing. Wafer-scale integration has already been demonstrated for electronic neuromorphic systems (Schemmel et al., 2010). Still, even at this scale, reaching 10^6 optoelectronic neurons per wafer is a tall order for either platform (**Figure 5**). We choose this integration metric somewhat arbitrarily; 10^6 neurons per wafer corresponds to 10^4 wafers for a human-cortex-scale system. This is roughly the same order as the number of processing units in modern supercomputers. If this target is to be reached, 3D integration at some level will be necessary. From **Figure 5**, it is clear that either platform will require a minimum of five photonic planes. Fortunately, photonic planes are quite amenable to 3D integration. Common waveguide materials include amorphous silicon (aSi), silicon nitride (SiN_x) and silicon oxynitride (SiO_xN_y). These dielectric materials can be deposited at low temperature, enabling several multi-planar demonstrations (Sacher et al., 2015; Shang et al., 2015; Chiles et al., 2017; Zhang et al., 2018b). Additionally, low-temperature deposition makes such processes compatible with back-end CMOS fabrication. It should be noted that five photonic planes represents a best-case scenario, as wider waveguides have lower loss and only minimal reduction in average path length (**Supplementary Information C**).

3D integration of active electronics is less straightforward, particularly for the semiconductor approach. 3D CMOS integration has been the subject of decades of research (Rosenberg, 1983; Knickerbocker et al., 2008; Sakuma et al., 2008; Vinet et al., 2011; Lim, 2013; Zhao et al., 2015; Elfadel and Gerhard Fettweis, 2016; Li et al., 2017) and still faces uncertainty. Required high-temperature processing steps for dopant activation and contact anneals typically have a degrading effect on previous layers. Much of 3D integration of silicon microelectronics takes place at the die scale (Elfadel and Gerhard Fettweis, 2016), which is incommensurate with the scale of systems under consideration. For the semiconductor scenario, the best course of action may be to abandon 3D active electronics altogether in favor of simply reducing the footprint (w_{sy}) of synapses. We see again from **Figure 5** that nearly 10^6 neurons can be integrated on a single plane if each synapse is on the order of 10 µm × 10 µm. This may be a challenging benchmark to reach with high-functionality synapses implementing complex plasticity and dynamics. Subthreshold circuits that have embraced larger CMOS nodes for decreased variability may need to adjust to more modern nodes, of which there is some precedent (Rubino et al., 2019). Additionally, photodetectors will be on the micron scale and long time-constant capacitors can require significant area (Appendix A) (Indiveri and Sandamirskaya, 2019). Both of these elements would however be fabricated on separate planes from MOSFETs.

Superconducting platforms would likely take the opposite approach, embracing 3D integration in the face of necessarily large device areas. Superconducting electronics, including active JJs, are routinely deposited at low temperatures (< 180 °C). Integrated circuits with two stacked planes of JJs have been demonstrated by two research laboratories (Ando et al., 2017; Tolpygo et al., 2019), along with multiple of planes of SNSPDs (Verma et al., 2012). This is particularly important, as superconducting systems will not be able to reach 10^6 neurons per wafer without 3D integration. A reasonable estimate for a superconducting synapse may be 30 µm on a side (**Supplementary Information B**). Such a size would require eight electronic planes.

We note that even if $p_p = p_e = 1$, it is still possible to fabricate wafers with 10^6 neurons, provided $\bar{k} = 100$, giving $\bar{L} = 3.5$ (Figures 9, 10 in **Supplementary Information C**). While this does not match the short path lengths of cognitive circuits in the brain, such a network is likely to have significant technological and scientific utility while offering an intermediate-term practical objective.

5.4. Constructing Multi-Wafer Systems

Given that neither system will scale to billions of neurons on a single wafer, many wafers (∼10,000) will need to be connected together to support human-brain-scale computing. A vision for a multi-wafer system is discussed in reference (Shainline, 2021) for the SOENs platform. Briefly, wafers are stacked and free-space optical communication is used to form highly inter-connected columns mimicking the modular structure of biological circuits

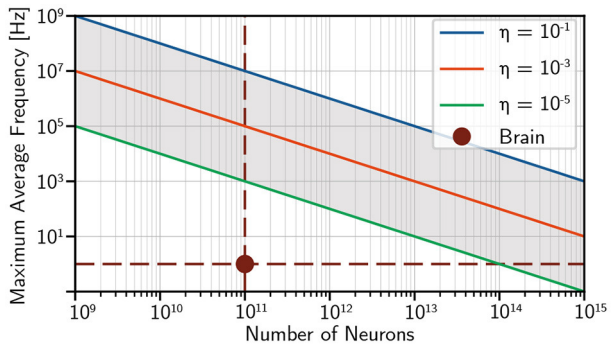


FIGURE 6 | Tradeoff between size and average spiking frequency for a population of optoelectronic neurons with a power budget of 10 MW ($f = \eta^* \frac{10\,\mathrm{MW}}{(N_{neurons} N\, E_{opt})}$. Fan-out ($N$) is 10^3 and the optical energy needed at each synapse is assumed to be 1 fJ (accounting for cooling in superconductor case). This likely would correspond to the limits of either superconductor or semiconductor neurons.

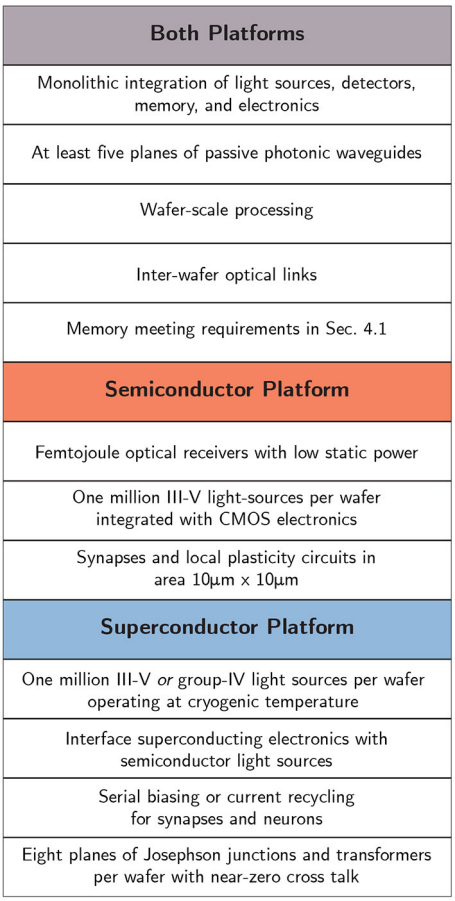


FIGURE 7 | Summary of necessary hardware demonstrations for each platform if human-brain-scale artificial cognition is to be achieved.

(Mountcastle, 1978, 1997; Meunier et al., 2010; Bota et al., 2015; Betzel and Bassett, 2017). Columns are coupled to each other with lateral inter-wafer connections, but such connectivity is more sparse than that within a column. Optical fibers provide low-loss communication over long distances.

Achieving systems of this scale requires advances, particularly in wafer-scale circuit integration and system-level construction. A phenomenon akin to Moore's law, with ever-decreasing feature sizes enabling ever-higher integration density is unlikely to carry this concept forward, as many device sizes are limited by other physical considerations. Metrics related to number of planes of integrated circuits and number of wafers in a system may be more relevant to chart progress in neuromorphic supercomputing. Gradual progress may be possible by consistently scaling up, but it is difficult to envision this sustained trend without a powerful economic drive.

5.5. Power Consumption and Cooling

5.5.1. Cooling Systems

Cooling systems will be a key component to either platform. For superconducting electronics, the system will fail completely if the temperature rises above the critical temperature (T_c). Superconducting neuromorphic systems will rely on niobium ($T_c = 9.3$ K) or a material with a similarly low T_c. Liquid helium (4.2 K) is the cryogen of choice for such temperatures. Cooling systems will add significantly to the power consumption of superconducting electronics. The power efficiency of a refrigeration system is measured by its specific power (Alekseev, 2015). The specific power gives the number of watts consumed by the refrigeration system for every watt of heat removed. The theoretical limit for specific power, given by the Carnot limit, is $\frac{T_H - T_C}{T_C}$. For liquid helium temperature (4.2 K), the Carnot limit demands that at least 74 watts of refrigeration power are required to remove every watt of heat produced on-chip if the system is operated in a 300 K ambient. State-of-the-art systems have reached specific powers below 400 W/W. Auspiciously, the most efficient refrigeration systems also tend to have the highest heat loads. The ability to cool heat loads as high as 10 kW at 4.2 K have already been demonstrated by commercially available systems (Holmes et al., 2013). Throughout this paper we assume a more conservative specific power of 1,000 W/W, representative of the smaller scale cryogenic systems used in most laboratories today. It does not appear that cryogenic capability will be an insurmountable obstacle toward large-scale superconducting neural systems.

5.5.2. Power Limitations

Modern supercomputers typically consume megawatts of power. Tianhe 2, for instance, requires 17.8 MW for operation (and another 6.4 MW for cooling) (Tolpygo, 2016). If we thus assume a total power budget of 10 MW, we can analyze the trade-off between average firing rate and number of neurons. We assume 1 fJ of optical energy is required to initiate a firing event at each synapse and plot the maximum average frequency spiking frequency for several different optical link efficiencies in **Figure 6**.

Power does not appear to be a limiting factor in achieving brain-scale and brain-speed optoelectronic networks. If the power resources of modern supercomputers were dedicated to a

brain-scale optoelectronic neuromorphic system, average spiking rates on the order of 10 kHz (10^4 speedup over biology) appear feasible even with relatively inefficient optical links. Such a system may enable brain-scale computation with time accelerated by four orders of magnitude.

Another factor to consider is power density. There is a maximum power density that can be handled by heat removal systems for both the semiconducting and superconducting case. In the semiconductor case, high-performance computing routinely generates power densities of hundreds of watts per square centimeter (Tolpygo, 2016). A theoretical limit of around 1 kW/cm^2 is postulated in Zhirnov et al. (2003). In contrast, superconducting systems will be required to operate at significantly lower power densities. Roughly 1 W/cm^2 is a conservative limit for on-chip power density that can be cooled with liquid helium (Tolpygo, 2016). Superconducting optical links appear to be capable of dissipating about three orders of magnitude less energy per bit, approximately canceling out the limited power density requirements of superconducting systems in comparison with semiconductors. In practice, it might well be the case that mature, sophisticated synapses and neurons will occupy so much area that these power density limitations will be of no consequence. For instance, even with link efficiency of $\eta = 10^{-4}$, a synapse would require a lateral dimension of less than 30 μm for power density considerations to limit spiking to less than 1 GHz. Section 5 argued that superconducting synapses are not likely to be smaller than this. 10 μm semiconducting synapses could reach 1 GHz with 1×10^{-3} efficiency. However, optoelectronic systems will have nonuniform power dissipation across the chip/wafer, with most of the power being dissipated at the light sources. A more in-depth analysis is required to see if heat removal will be an issue near the light sources in particular, but for the superconducting case it is convenient that the light sources themselves are not superconducting, and can afford to be raised to higher temperatures without failure. Concerns about local heating may be assuaged with layouts that sufficiently shield and/or separate thermally sensitive devices from the light sources.

6. CONCLUSION

The prospects of neuromorphic systems at the scale of the brain and beyond are tantalizing. The fan-out capability of optical communication coupled with the computational power of electronic circuitry makes optoelectronic systems a promising framework for realizing these high ambitions. However, there is no technology platform that is ready to support optoelectronic spiking networks of the scale and sophistication of the human brain. Making this vision a reality will require breakthroughs at the device level, no matter which path is chosen, particularly with regard to integrated light sources. Beyond that, several different classes of devices must be integrated alongside each other, which further reduces the likelihood for success. Efficient, densely integrated light sources, waveguide-coupled detectors, local memory devices, and capable neuronal circuitry all must be consolidated onto a single platform. Candidates for all requisite devices can be proposed for either semiconducting or superconducting platforms, and the two systems may be capable of similar performance. However, the technological paths toward achieving brain-scale systems with the two platforms diverge in important respects (**Figure 7**).

Semiconductor platforms hold advantages in technological maturity, room-temperature operation, and perhaps speed. Spike rates in excess of 10 GHz may be feasible, but only for systems significantly smaller than the human brain due to power constraints. Semiconductor receivers can potentially operate with extremely low energies per spiking event (sub femto-joule), making them a worthy competitor of superconducting single photon detectors. However, these low energy receivers require significant optical power from integrated light sources. To achieve biological-scale fan-out, either very bright light sources, repeatering schemes (costing area and yield), or additional gain stages (costing power) will need to be included. In terms of neuronal computation, semiconductor neurons have already demonstrated impressive functionality and low-power operation that should be capable of integration with optical communication infrastructure, provided the long-standing challenges with CMOS-integrated III-V light sources can be overcome. Synaptic memory is a major open question, but a variety of non-volatile memory solutions have seen extensive investigation, and time will tell if one technology can meet the requirements we have laid out for brain-scale optoelectronic systems. 3D integration of transistors, photodetectors, and memory may not be a feasible solution, meaning aggressive scaling of synaptic circuits while maintaining complex functionality is perhaps a better strategy. The fabrication processes for mature semiconductor neural systems may prove to be prohibitively complicated and heterogeneous, perhaps requiring different processing strategies for sources, detectors, and memories. If wafer-scale monolithic integration of these components cannot be achieved, and chip-scale die-stacking techniques are required, the outlook for achieving brain-scale systems is limited.

Superconducting optoelectronic neural systems suffer from a comparatively primitive fabrication ecosystem, but the incorporation of superconducting devices provides several intriguing properties. SNSPD receivers place nearly the theoretical minimum burden on integrated light sources. This attribute compounds positively with the improvements in efficiency for light sources operating at cryogenic temperatures. Integration of light sources with superconducting electronics does not appear to have the same material integration challenges as integration with CMOS, but this state of affairs may be due to the lack of attention the effort has received. These factors make the large-scale integration of light sources appear more tractable than in the semiconductor case—perhaps even opening the door to silicon as an active optical material. Driving these light sources with superconducting electronics, however, has yet to demonstrate the performance required for this application. The implementation of a high-impedance pulse-and-reset circuit remains an open challenge. For computation, superconducting neuronal circuits appear just as capable of implementing complex neuronal and synaptic behaviors as their CMOS counterparts, but will need to be designed with serial biasing in order to scale.

Additionally, some speed advantages present in superconducting electronics will be negated by the response time of SNSPDs (<1 GHz). Of course, even if maximum spike rates are limited to 20 MHz, this would still represent a speed-up of four orders of magnitude over biological systems. Memory seems to be a strength for the superconducting platform, as superconductivity provides new avenues of storing synaptic weights. Loop memory in particular may be capable of implementing plasticity mechanisms that operate with only the signals produced through normal network activity. Caution is in order here, however, as superconducting synaptic plasticity mechanisms have scarcely been explored. 3D integration may yield more readily in the superconductor platform. The inconvenience of cryogenic cooling is a serious consideration, but power and heat removal estimations indicate this is unlikely to be a limiting factor for brain-scale systems. *If* all these issues can be resolved, superconducting optoelectronic systems may require simpler manufacturing processes than the semiconductor approach, as the material ecosystem could potentially be parsimonious. Of course, superconducting foundries are far less developed than their semiconductor counterparts, which may negate these advantages in the near-term.

We would be remiss to paint the quest for neuromorphic supercomputing as only a question of hardware. The inner workings of the brain are the subject of intense investigation, and the emergent phenomena of cognition and consciousness remain taunting, increasingly lonely enigmas entrenched in the netherworld between philosophy and science. Watershed breakthroughs in neuroscience and algorithmic development will be required for the discussed hardware platforms to have practical applications, although the hardware platforms themselves may be of use in helping to unravel some of these mysteries. The question of whether it is prudent to develop hardware before algorithms has pestered the field of neuromorphic computing since its inception. In this case, we believe that the length of development, rich opportunities for spin-off technologies, and inestimable potential make such hardware development well-worth pursuing even at this incipient stage.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

This work was supported by the National Institute of Standards and Technology. BP was supported under the financial assistance award 70NANB18H006 from the U.S. Department of Commerce, National Institute of Standards and Technology.

ACKNOWLEDGMENTS

We thank Drs. Brian Hoskins, Advait Madhavan, and Alexander Tait for helpful insights and conversation.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.732368/full#supplementary-material

REFERENCES

Abraham, I. (2018). The case for rejecting the memristor as a fundamental circuit element. *Nature* 8:10972. doi: 10.1038/s41598-018-29394-7

Abu-Hassan, K., Taylor, J., Morris, P., Donati, E., Bortolotto, Z., Indiveri, G., et al. (2019). Optimal solid state neurons. *Nat. Comm.* 10:5309. doi: 10.1038/s41467-019-13177-3

Akhlaghi, M. K., Schelew, E., and Young, J. F. (2015). Waveguide integrated superconducting single-photon detectors implemented as near-perfect absorbers of coherent radiation. *Nat. Commun.* 6, 1–8. doi: 10.1038/ncomms9233

Alekseev, A. (2015). Basics of low-temperature refrigeration. *arXiv [Preprint] arXiv*:1501.07392. doi: 10.5170/CERN-2014-005.111

Allen, C., and Stevens, C. F. (1994). An evaluation of causes for unreliability of synaptic transmission. *Proc. Natl. Acad. Sci. U.S.A.* 91, 10380–10383. doi: 10.1073/pnas.91.22.10380

Ambrogio, S., Ciocchini, N., Laudato, M., Milo, V., Pirovano, A., Fantini, P., et al. (2016). Unsupervised learning by spike timing dependent plasticity in phase change memory (pcm) synapses. *Front. Neurosci.* 10:56. doi: 10.3389/fnins.2016.00056

Ando, T., Nagasawa, S., Takeuchi, N., Tsuji, N., China, F., Hidaka, M., et al. (2017). Three-dimensional adiabatic quantum-flux-parametron fabricated using a double-active-layered niobium process. *Supercond. Sci. Technol.* 30:075003. doi: 10.1088/1361-6668/aa6ef4

Assefa, S., Xia, F., and Vlasov, Y. A. (2010). Reinventing germanium avalanche photodetector for nanophotonic on-chip optical interconnects. *Nature* 464, 80–84. doi: 10.1038/nature08813

Bao, J., Tabbal, M., Kim, T., Charnvanichborikarn, S., Williams, J. S., Aziz, M. J., et al. (2007b). Point defect engineered si sub-bandgap light-emitting diode. *Opt. Express* 15, 6727–6733. doi: 10.1364/OE.15.006727

Bartol T. M. Jr, Bromer, C., Kinney, J., Chirillo, M. A., Bourne, J. N., Harris, K. M., et al. (2015). Nanoconnectomic upper bound on the variability of synaptic plasticity. *Elife* 4:e10778. doi: 10.7554/eLife.10778

Beaufils, C., Redjem, W., Rousseau, E., Jacques, V., Kuznetsov, A., Raynaud, C., et al. (2018). Optical properties of an ensembe of G-centers in silicon. *Phys. Rev. B* 97:035303. doi: 10.1103/PhysRevB.97.035303

Beggs, J. (2007). The criticality hypothesis: how local cortical networks might optimize information processing. *Philos. Trans. R. Soc. A* 366:329. doi: 10.1098/rsta.2007.2092

Benjamin, B., Gao, P., McQuinn, E., Choudhary, S., Chandresekaran, A., Bussat, J.-M., et al. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102:699. doi: 10.1109/JPROC.2014.2313565

Bergeron, L., Chartrand, C., Kurkjian, A., Morse, K., Riemann, H., Abrosimov, N., et al. (2020). Silicon-Integrated Telecommunications Photon-Spin Interface. *PRX Quantum* 1:020301. doi: 10.1103/PRXQuantum.1.020301

Betzel, R., and Bassett, D. (2017). Multi-scale brain networks. *Neuroimage* 160:73. doi: 10.1016/j.neuroimage.2016.11.006

Bota, M., Sporns, O., and Swanson, L. (2015). Architecture of the cerebral cortical association connectome underlying cognition. *Proc. Natl. Acad. Sci. U.S.A*. 112, E2093–E2101. doi: 10.1073/pnas.1504394112

Bowers, J. E., Komljenovic, T., Davenport, M., Hulme, J., Liu, A. Y., Santis, C. T., et al. (2016). "Recent advances in silicon photonic integrated circuits," in *Next-Generation Optical Communication: Components, Sub-Systems, and Systems V, Vol. 9774* (San Francisco, CA: International Society for Optics and Photonics), 977402.

Bradfield, P., Brown, T., and Hall, D. (1989). Electroluminescence from sulfur impurities in an p-n junction formed in epitaxial silicon. *Appl. Phys. Lett*. 55:100. doi: 10.1063/1.102115

Braitenberg, V., and Schuz, A. (1998). *Cortex: Statistics and Geometry of Neuronal Connectivity*. Berlin: Springer.

Brink, S., Nease, S., and Hasler, P. (2013). Computing with networks of spiking neurons on a biophysically motivated floating-gate based neuromorphic integrated circuit. *Neural Netw*. 45:39. doi: 10.1016/j.neunet.2013.02.011

Brown, T., and Hall, D. (1986). Observation of electroluminescence from excitons bound to isoelectronic impurities in crystalline silicon. *J. Appl. Phys*. 59:1399. doi: 10.1063/1.336489

Buckley, S., Chiles, J., McCaughan, A. N., Moody, G., Silverman, K. L., Stevens, M. J., et al. (2017). All-silicon light-emitting diodes waveguide-integrated with superconducting single-photon detectors. *Appl. Phys. Lett*. 111, 141101. doi: 10.1063/1.4994692

Buckley, S., Tait, A., Chiles, J., McCaughan, A., Khan, S., Mirin, R., et al. (2020a). Integrated-photonic characterization of single-photon detectors for use in neuromorphic synapses. *Phys. Rev. Appl*. 14:054008. doi: 10.1103/PhysRevApplied.14.054008

Buckley, S. M., Tait, A. N., Moody, G., Primavera, B., Olson, S., Herman, J., et al. (2020b). Optimization of photoluminescence from w centers in a silicon-on-insulator. *Opt Express* 28, 16057–16072. doi: 10.1364/OE.386450

Bullmore, E., and Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and functional systems. *Nat. Rev. Neurosci*. 10:186. doi: 10.1038/nrn2575

Bullmore, E., and Sporns, O. (2012). The economy of brain network organization. *Nat. Rev. Neurosci*. 13:336. doi: 10.1038/nrn3214

Buzsáki, G. (2006). *Rhythms of the Brain*. New York, NY: Oxford University Press.

Camacho-Aguilera, R., Han, Z., Cai, Y., Kimerling, L. C., and Michel, J. (2013). Direct band gap narrowing in highly doped ge. *Appl. Phys. Lett*. New York, NY: 102, 152106. doi: 10.1063/1.4802199

Chartrand, C., Bergeron, L., Morse, K., Riemann, H., Abrosimov, N., Becker, P., et al. (2018). Highly enriched ^{2}8Si reveals remarkable optical linewidths and fine structure for well-known damage centers. *Phys. Rev. B* 98:195201. doi: 10.1103/PhysRevB.98.195201

Chen, S., Li, W., Wu, J., Jiang, Q., Tang, M., Shutts, S., et al. (2016). Electrically pumped continuous-wave III-V quantum dot lasers on silicon. *Nat. Photonics* 10:307. doi: 10.1038/nphoton.2016.21

Chicca, E., Stefanini, F., Bartolozzi, C., and Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proc. IEEE* 102, 1367–1388. doi: 10.1109/JPROC.2014.2313954

Chiles, J., Buckley, S., Nader, N., Nam, S., Mirin, R., and Shainline, J. (2017). Multi-planar amorphous silicon photonics with compact interplanar couplers, cross talk mitigation, and low crossing loss. *APL Photonics* 2:116101. doi: 10.1063/1.5000384

Cocchi, L., Gollo, L. L., Zalesky, A., and Breakspear, M. (2017). Criticality in the brain: A synthesis of neurobiology, models and cognition. *Progr. Neurobiol*. 158, 132–152. doi: 10.1016/j.pneurobio.2017.07.002

Crosnier, G., Sanchez, D., Bouchoule, S., Monnier, P., Beaudoin, G., Sagnes, I., et al. (2017). Hybrid indium phosphide-on-silicon nanolaser diode. *Nat. Photon*. 11:297. doi: 10.1038/nphoton.2017.56

Crotty, P., Schult, D., and Segall, K. (2010). Josephson junction simulation of neurons. *Phys. Rev. E* 82, 011914. doi: 10.1103/PhysRevE.82.011914

Dalgaty, T., Payvand, M., Moro, F., Ly, D. R., Pebay-Peyroula, F., Casas, J., et al. (2019). Hybrid neuromorphic circuits exploiting non-conventional properties of rram for massively parallel local plasticity mechanisms. *APL Mater*. 7, 081125. doi: 10.1063/1.5108663

Davies, G. (1989). The optical properties of luminescence centres in silicon. *Phys. Rep*. 176, 83–188. doi: 10.1016/0370-1573(89)90064-1

Debaes, C., Bhatnagar, A., Agarwal, D., Chen, R., Keeler, G. A., Helman, N. C., et al. (2003). Receiver-less optical clock injection for clock distribution networks. *IEEE J. Select. Top. Quantum Electron*. 9, 400–409. doi: 10.1109/JSTQE.2003.813319

DeRose, C. T., Trotter, D. C., Zortman, W. A., Starbuck, A. L., Fisher, M., Watts, M. R., et al. (2011). Ultra compact 45 ghz cmos compatible germanium waveguide photodiode with low dark current. *Opt Express* 19, 24897–24904. doi: 10.1364/OE.19.024897

Diorio, C., Hasler, P., Minch, B. A., and Mead, C. (1998). "Floating-gate mos synapse transistors," in *Neuromorphic Systems Engineering* (Springer), 315–337. doi: 10.1007/978-0-585-28001-1_14

Dolores-Calzadilla, V., Romeira, B., Pagliano, F., Birindelli, S., Higuera-Rodriguez, A., Van Veldhoven, P., et al. (2017). Waveguide-coupled nanopillar metal-cavity light-emitting diodes on silicon. *Nat. Commun*. 8, 1–8. doi: 10.1038/ncomms14323

Duzer, T. V., and Turner, C. (1998). *Principles of Superconductive Devices and Circuits, 2nd Edn*. Hoboken, NJ: Prentice Hall.

El Kurdi, M., Kociniewski, T., Ngo, T.-P., Boulmer, J., Debarre, D., Boucaud, P., et al. (2009). Enhanced photoluminescence of heavily n-doped germanium. *Appl. Phys. Lett*. 94, 191107. doi: 10.1063/1.3138155

Elfadel, A., and Gerhard Fettweis (2016). *3D stacked chips*. New York, NY: Springer.

Ennen, H., Pomrenke, G., Axmann, A., Eisele, K., Haydl, W., and Schneider, J. (1985). 1.54 μ m electroluminescence of erbium-doped silicon grown by molecular beam epitaxy. *Appl. Phys. Lett*. 46:381.

Euler, T., and Denk, W. (2001). Dendritic processing. *Curr. Opin. Neurobiol*. 11, 415–422. doi: 10.1016/S0959-4388(00)00228-2

Fadaly, E. M., Dijkstra, A., Suckert, J. R., Ziss, D., van Tilburg, M. A., Mao, C., et al. (2020). Direct-bandgap emission from hexagonal Ge and SiGe alloys. *Nature* 580, 205–209. doi: 10.1038/s41586-020-2150-y

Ferrari, S., Kahl, O., Kovalyuk, V., Goltsman, G., Korneev, A., and Pernice, W. (2015). Waveguide-integrated single- and multi-photon detecton at telecom wavelengths using superconducting nanowires. *Appl. Phys. Lett*. 106:151101. doi: 10.1063/1.4917166

Ferrari, S., Schuck, C., and Pernice, W. (2018). Waveguide-integrated superconducting nanowire single-photon detectors. *Nanophotonics* 7, 1725–1758. doi: 10.1515/nanoph-2018-0059

Friston, K., and Buzsáki, G. (2016). The functional anatomy of time: what and when in the brain. *Trends Cogn. Sci*. 20, 500. doi: 10.1016/j.tics.2016.05.001

Fronczak, A., Fronczak, P., and Holyst, J. (2004). Average path length in random networks. *Phys. Rev. E* 70:056110. doi: 10.1103/PhysRevE.70.056110

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng*. 13, 051001. doi: 10.1088/1741-2560/13/5/051001

Fusi, S., and Abbott, L. (2007). Limits on the memory storage capacity of bounded synapses. *Nat. Neurosci*. 10:485. doi: 10.1038/nn1859

Fusi, S., Drew, P., and Abbott, L. (2005). Casdcade models of synaptically stored memories. *Neuron* 45:599. doi: 10.1016/j.neuron.2005.02.001

Ghrib, A., De Kersauson, M., El Kurdi, M., Jakomin, R., Beaudoin, G., Sauvage, S., et al. (2012). Control of tensile strain in germanium waveguides through silicon nitride layers. *Appl. Phys. Lett*. 100, 201104. doi: 10.1063/1.4718525

Green, M. A., Zaho, J., Wang, A., Reese, P. J., and Gal, M. (2001). Efficient silicon light-emitting diodes. *Nature* 412:805. doi: 10.1038/35090539

Gurioli, M., Vinattieri, A., Colocci, M., Deparis, C., Massies, J., Neu, G., et al. (1991). Temperature dependence of the radiative and nonradiative recombination time in gaas/al x ga 1- x as quantum-well structures. *Phys. Rev. B* 44, 3115.

Halbritter, H., Jäger, C., Weber, R., Schwind, M., and Möllmer, F. (2014). High-speed led driver for ns-pulse switching of high-current leds. *IEEE Photonics Technol. Lett*. 26, 1871–1873. doi: 10.1109/LPT.2014.2336732

Han, Y., Xue, Y., Yan, Z., and Lau, K. (2021). Selectively Grown III-V Lasers for Integrated Si-Photonics. *J. Lightwave Technol*. 39:940. doi: 10.1109/JLT.2020.3041348

Harada, Y., and Goto, E. (1991). Artificial neural network circuits with josephson devices. IEEE Trans. *Magnetics* 27:2863. doi: 10.1109/20.133806

Hasler, J., and Marr, H. B. (2013). Finding a roadmap to achieve large neuromorphic hardware systems. *Front. Neurosci*. 7:118. doi: 10.3389/fnins.2013.00118

Hawkins, J., and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Front. Neural Circ.* 10:23. doi: 10.3389/fncir.2016.00023

Hennessy, J. L., and Patterson, D. A. (2011). *Computer Architecture: A Quantitative Approach*. Cambridge: MA, Elsevier.

Hidaka, M., and Akers, L. (1991). An artificial neural cell implemented with superconducting circuits. *Supercond. Sci. Technol.* 4:654. doi: 10.1088/0953-2048/4/11/027

Hollenback, M., Berencén, Y., Kentsch, U., Helm, M., and Astakhov, G. (2020). Engineering telecom single-photon emitters in silicon for scalable quantum photonics. *Opt. Express* 28:26111. doi: 10.1364/OE.397377

Holmes, D. S., Ripple, A. L., and Manheimer, M. A. (2013). Energy-efficient superconducting computing–power budgets and requirements. *IEEE Trans. Appl. Supercond.* 23, 1701610–1701610. doi: 10.1109/TASC.2013.2244634

Hu, Y., Liang, D., Mikherjee, K., Li, Y., Zhang, C., Kurcveil, G., et al. (2019). III/V-on-Si MQW lasers by using a novel photonic integration method of regrowth on a bonding template. *Light Sci. Appl.* 8:93. doi: 10.1038/s41377-019-0202-6

Ielmini, D. (2018). Brain-inspired computing with resistive switching memory (rram): devices, synapses and neural networks. *Microelectron. Eng.* 190:44–53. doi: 10.1016/j.mee.2018.01.009

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., and Sandamirskaya, Y. (2019). The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment. *IEEE Signal Process Mag.* 36, 16–28. doi: 10.1109/MSP.2019.2928376

Ishikawa, Y., Wada, K., Cannon, D. D., Liu, J., Luan, H.-C., and Kimerling, L. C. (2003). Strain-induced band gap shrinkage in ge grown on si substrate. *Appl. Phy.s Lett*. 82, 2044–2046. doi: 10.1063/1.1564868

Iyer, S. S., and Xie, Y.-H. (1993). Light emission from silicon. *Science* 260:40. doi: 10.1126/science.260.5104.40

Jiao, Y., van der Tol, J., Pogoretskii, V., van Engelen, J., Kashi, A., Reniers, S., et al. (2020). Indium phosphide membrane nanophotonic integrated circuits on silicon. *Physica Status Solidi* 217:1900606. doi: 10.1002/pssa.201900606

Jung, D., Norman, J., Kennedy, M., Shang, C., Shin, B., Wan, Y., et al. (2017). High efficiency low threshold current 1.3 μ m inas quantum dot lasers on on-axis (001) gap/si. *Appl. Phys. Lett*. 111, 122107. doi: 10.1063/1.4993226

Justice, J., Bower, C., Meitl, M., Mooney, M., Gubbins, M., and Corbett, B. (2012). Wafer-scale integration of group III-V lasers on silicon using transfer printing of epitaxial layers. *Nat. Photonics* 6:610. doi: 10.1038/nphoton.2012.204

Kadin, A. M. (1999). *Introduction to Superconducting Circuits, 1st Edn*. New York, NY: John Wiley and Sons. first edition.

Keyes, R. (1982). The wire-limited logic chip. *IEEE J. Sol. State Circ*. 17:1232. doi: 10.1109/JSSC.1982.1051887

Kim, M.-K., and Lee, J.-S. (2019). Ferroelectric analog synaptic transistors. *Nano Lett*. 19, 2044–2050. doi: 10.1021/acs.nanolett.9b00180

Kim, S. G., Han, J. S., Kim, H., Kim, S. Y., and Jang, H. W. (2018). Recent advances in memristive materials for artificial synapses. *Adv. Mater. Technol.* 3, 1800457. doi: 10.1002/admt.201800457

Kirichenko, D., Sarwana, S., and Kirichenko, A. (2011). Zero static power dissipation biasing of rsfq circuits. *IEEE Trans. Appl. Supercond.* 21:776. doi: 10.1109/TASC.2010.2098432

Knickerbocker, J. U., Andry, P. S., Dang, B., Horton, R. R., Interrante, M. J., Patel, C. S., et al. (2008). Three- dimensional silicon integration. *IBM J. Res. Dev*. 52, 553–569. doi: 10.1147/JRD.2008.5388564

Koch, C., and Segev, I. (2000). The role of single neurons in information processing. *Nat. Neurosci*. 3:1171. doi: 10.1038/81444

Kveder, V., Badylevich, M., Steinman, E., Izotov, A., Seibt, M., and Schröter, W. (2004). Room-temperature silicon light-emitting diodes based on dislocation luminescence. *Appl. Phys. Lett*. 84:2106. doi: 10.1063/1.1689402

Li, M., Shi, J., Rahman, M., Khasanvis, S., Bhat, S., and Moritz, C. (2017). Skybridge-3D-CMOS: a fine-grained 3D CMOS integrated circuit technology. *IEEE Trans. Nanotech.* 16, 639. doi: 10.1109/TNANO.2017.2700626

Lim, S. (2013). *Design for High Performance, Low Power, and Reliable 3D Integrated Circuits*. New York, NY: Springer. doi: 10.1007/978-1-4419-9542-1

Lisman, J. (1997). Bursts as a unit of neural information: making unreliable synapses reliable. *Trends Neurosci*. 20:38. doi: 10.1016/S0166-2236(96)10070-9

Liu, J., Sun, X., Camacho-Aguilera, R., Kimerling, L. C., and Michel, J. (2010). Ge-on-si laser operating at room temperature. *Opt. Lett*. 35, 679–681. doi: 10.1364/OL.35.000679

Liu, J., Sun, X., Pan, D., Wang, X., Kimerling, L. C., Koch, T. L., et al. (2007). Tensile-strained, n-type ge as a gain medium for monolithic laser integration on si. *Opt Express* 15, 11272–11277. doi: 10.1364/OE.15.011272

Liu, S.-C., Delbruck, T., Indiveri, G., Whatley, A., and Douglas, R., (eds.) (2015). *Event-Based Neuromorphic Systems*. Chichester: John Wiley and Sons.

Marder, E. (1987). "Neurotransmitters and neuromodulators," in *The Crustacean Stomatogastric System* (Berlin: Springer), 263–306. doi: 10.1007/978-3-642-71516-7_10

Marsili, F., Verma, V., Stern, J., Harrington, S., Lita, A., Gerrits, T., et al. (2013). Detecting single infrared photons with 93% system efficiency. *Nat. Photon.* 7:210. doi: 10.1038/nphoton.2013.13

McCaughan, A. N., Verma, V. B., Buckley, S. M., Allmaras, J., Kozorezov, A., Tait, A., et al. (2019). A superconducting thermal switch with ultrahigh impedance for interfacing superconductors to semiconductors. *Nat. Electron.* 2, 451–456. doi: 10.1038/s41928-019-0300-8

McDonnell, M. D., and Ward, L. M. (2011). The benefits of noise in neural systems: bridging theory and experiment. *Nat. Rev. Neurosci.* 12, 415–425. doi: 10.1038/nrn3061

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Mehta, K., Orcutt, J., Shainline, J., Tehar-Zahav, O., Sternberg, Z., Meade, R., et al. (2014). Polycrystalline silicon ring resonator photodiodes in a bulk complementary metal-oxide-semiconductor process. *Opt. Lett*. 39:1061. doi: 10.1364/OL.39.001061

Meunier, D., Lambiotte, R., and Bullmore, E. (2010). Modular and hierarchically modular organization of brain networks. *Front. Neurosci.* 4, 1. doi: 10.3389/fnins.2010.00200

Miller, D. A. (2017). Attojoule optoelectronics for low-energy information processing and communications. *J. Lightwave Technol.* 35, 346–396. doi: 10.1109/JLT.2017.2647779

Mountcastle, V. (1978). *An Organizing Principle for Cerebral Function: The Unit Module and the Distributed System*. Cambridge, MA: The MIT Press.

Mountcastle, V. (1997). The columnar organization of the neocortex. *Brain* 120:701. doi: 10.1093/brain/120.4.701

Ng, W. L., Lourenco, M., Gwilliam, R., Ledain, S., Shao, G., and Homewood, K. (2001). An efficient room-temperature silicon-based light-emitting diode. *Nature* 410, 192–194. doi: 10.1038/35065571

Norman, J. C., Jung, D., Wan, Y., and Bowers, J. E. (2018). Perspective: The future of quantum dot photonic integrated circuits. *APL Photonics* 3, 030901. doi: 10.1063/1.5021345

Nozaki, K., Matsuo, S., Fujii, T., Takeda, K., Shinya, A., Kuramochi, E., et al. (2018). Forward-biased nanophotonic detector for ultralow-energy dissipation receiver. *APL Photonics* 3, 046101. doi: 10.1063/1.5022074

Palm, J., Gan, F., Zheng, B., Michel, J., and Kimerling, L. (1996). Electroluminescence of erbium-doped silicon. *Phys. Rev. B* 54:17603. doi: 10.1103/PhysRevB.54.17603

Pernice, W., Schuck, C., Minaeva, O., Li, M., Goltsman, G., Sergienko, A., et al. (2012). High speed travelling wave single-photon detectors with near-unity quantum efficiency. *Nat. Comm.* 3:1325. doi: 10.1038/ncomms2307

Pfeil, T., Grubl, A., Jeltsch, S., Müller, E., Metrovici, M., Schmuker, M., et al. (2013). Six networks on a universal neuromorphic computing substrate. *Front. Neurosci.* 7:1. doi: 10.3389/fnins.2013.00011

Pfeil, T., Potjans, T. C., Schrader, S., Potjans, W., Schemmel, J., Diesmann, M., et al. (2012). Is a 4-bit synaptic weight resolution enough–constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Front. Neurosci.* 6:90. doi: 10.3389/fnins.2012.00090

Pizzone, A., Srinivasan, S. A., Verheyen, P., Lepage, G., Balakrishnan, S., and Van Campenhout, J. (2020). "Analysis of dark current in ge-on-si photodiodes at cryogenic temperatures," in *2020 IEEE Photonics Conference (IPC)* (Vancouver, BC: IEEE), 1-2. doi: 10.1109/IPC47351.2020.9252362

Rajendran, B., Liu, Y., Seo, J.-,s., Gopalakrishnan, K., Chang, L., Friedman, D. J., et al. (2012). Specifications of nanoscale devices and circuits for neuromorphic computational systems. *IEEE Trans. Electron. Devices* 60, 246–253. doi: 10.1109/TED.2012.2227969

Ramakrishnan, S., Hasler, P. E., and Gordon, C. (2011). Floating gate synapses with spike-time-dependent plasticity. *IEEE Trans. Biomed. Circ. Syst.* 5, 244–252. doi: 10.1109/TBCAS.2011.2109000

Razavi, B. (2012). *Design of Integrated Circuits for Optical Communications*. Hoboken, NJ: John Wiley Sons.

Reddy, D., Nerem, R., Nam, S., Mirin, R., and Verma, V. (2020). Superconducting nanowire single-photon detectors with 98% system detection efficiency at 1550 nm. *Optica* 7:1649.

Redjem, W., Durand, A., Herzig, T., Benali, A., Pezzagna, S., Meijer, J., et al. (2020). Single artificial atoms in silicon emitting at telecom wavelengths. *Nat. Electron.* 3:738. doi: 10.1038/s41928-020-00499-0

Romeira, B., and Fiore, A. (2018). Purcell effect in the stimulated and spontaneous emission rates of nanoscale semiconductor lasers. *IEEE J. Quantum. Electron.* 54, 1–12. doi: 10.1109/JQE.2018.2802464

Romeira, B., and Fiore, A. (2019). Physical limits of nanoleds and nanolasers for optical communications. *Proc. IEEE* 108, 735–748. doi: 10.1109/JPROC.2019.2912293

Rosenberg, A. (1983). Three-dimensional VLSI: a case study. *J. Assoc. Computing Machinery* 30, 397. doi: 10.1145/2402.322384

Rosenberg, D., Kerman, A., Molnar, R., and Dauler, E. (2013). High-speed and high-efficiency superconducting nanowire single photon detector array. *Opt. Express* 21, 1440–1447. doi: 10.1364/OE.21.001440

Rotem, E., Shainline, J., and Xu, J. (2007). Electroluminescence of nanopatterned silicon with carbon implantation and solid phase epitaxial regrowth. *Opt. Express* 15:14099. doi: 10.1364/OE.15.014099

Rubino, A., Payvand, M., and Indiveri, G. (2019). "2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019* (Genova: IEEE), 458–461.

Sacher, W., Huang, Y., Lo, G.-Q., and Poon, J. (2015). Multilayer silicon nitride-on-silicon integrated photonic platforms and devices. *J. Lightwave Tech.* 33:901. doi: 10.1109/JLT.2015.2392784

Sahin, D., Gaggero, A., Weber, J.-W., Agafonov, I., Verheijen, M., Mattioli, F., et al. (2015). Waveguide nanowire superconducting single-photon detectors fabricated on gaas and the study of their optical properties. *IEEE J. Sel. Top. Quant. Electron.* 21:3800210. doi: 10.1109/JSTQE.2014.2359539

Sakuma, K., Andry, P., Tsang, C., Wright, S., Dang, B., Patel, C., et al. (2008). 3D chip-stacking technology with through-silicon vias and low-volume lead-free interconnections. *IBM J. Res. Dev.* 52, 611. doi: 10.1147/JRD.2008.5388567

Sandiford, D. (1958). Temperature dependence of carrier lifetime in silicon. *Proc. Phys. Soc.* 71:1002. doi: 10.1088/0370-1328/71/6/313

Sardi, S., Vardi, R., Sheinin, A., Goldental, A., and Kanter, I. (2017). New types of experiments reveal that a neuron functions as multiple independent threshold units. *Sci. Rep.* 7:18036. doi: 10.1038/s41598-017-18363-1

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Schemmel, J., Kriener, L., Müller, P., and Meier, K. (2017). "An accelerated analog neuromorphic hardware system emulating nmda-and calcium-based non-linear dendrites," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 2217–2226.

Schneider, M., and Segall, K. (2020). Fan-out and fan-in properties of superconducting neuromorphic circuits. *J. Appl. Phys.* 128, 214903. doi: 10.1063/5.0025168

Schneider, M. L., Donnelly, C. A., and Russek, S. E. (2018a). Tutorial: high-speed low-power neuromorphic systems based on magnetic josephson junctions. *J. Appl. Phys.* 124, 161102. doi: 10.1063/1.5042425

Schneider, M. L., Donnelly, C. A., Russek, S. E., Baek, B., Pufall, M. R., Hopkins, P. F., et al. (2017). "Energy-efficient single-flux-quantum based neuromorphic computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Washington, DC, IEEE), 1–4. doi: 10.1109/ICRC.2017.8123634

Schneider, M. L., Donnelly, C. A., Russek, S. E., Baek, B., Pufall, M. R., Hopkins, P. F., et al. (2018b). Ultralow power artificial synapses using nanotextured magnetic josephson junctions. *Sci. Adv.* 4:e1701329.

Segal, C., Dalakoti, A., Miller, M., and Brewer, F. (2016). "Connectivity effects on energy and area for neuromorphic system with high speed asynchronous pulse mode links," in *2016 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)* (Austin, TX: IEEE), 16.

Shainline, J., Buckley, S., McCaughan, A., Chiles, J., Jafari-Salim, A., Mirin, R., et al. (2018). Circuit designs for superconducting optoelectronic loop neurons. *J. Appl. Phys.* 124:152130. doi: 10.1063/1.5038031

Shainline, J., Buckley, S., Nader, N., Gentry, C., Cossel, K., Cleary, J., et al. (2017a). Room-temperature-deposited dielectrics and superconductors for integrated photonics. *Opt. Express* 25:10322.

Shainline, J., and Xu, J. (2007). Silicon as an emissive optical medium. *Laser Photonics Rev.* 1:334. doi: 10.1002/lpor.200710021

Shainline, J. M. (2019). Fluxonic processing of photonic synapse events. *IEEE J. Select. Top. Quantum Electron.* 26, 1–15. doi: 10.1109/JSTQE.2019.2927473

Shainline, J. M. (2021). Optoelectronic intelligence. *arXiv [Preprint] arXiv*:2010.08690. doi: 10.1063/5.0040567

Shainline, J. M., Buckley, S. M., McCaughan, A. N., Chiles, J. T., Jafari Salim, A., Castellanos-Beltran, M., et al. (2019). Superconducting optoelectronic loop neurons. *J. Appl. Phys.* 126, 044902. doi: 10.1063/1.5096403

Shainline, J. M., Buckley, S. M., Mirin, R. P., and Nam, S. W. (2017b). Superconducting optoelectronic circuits for neuromorphic computing. *Phys. Rev. Appl.* 7, 034013. doi: 10.1103/PhysRevApplied.7.034013

Shang, K., Pathak, S., Guan, B., Liu, G., and Yoo, S. (2015). Low-loss compact multilayer silicon nitride platform for 3D photonic integrated circuits. *Opt. Express* 23:21334. doi: 10.1364/OE.23.021334

Simon, H. (1962). The architecture of complexity. *Proc. Amer. Phil. Soc.* 106:467.

Song, B., Stagarescu, C., Ristic, S., Behfar, A., and Klamkin, J. (2016). 3D integrated hybrid silicon laser. *Opt. Express* 24:10435. doi: 10.1364/OE.24.010435

Sourikopoulos, I., Hedayat, S., Loyez, C., Danneville, F., Hoel, V., Mercier, E., et al. (2017). A 4-fj/spike artificial neuron in 65 nm cmos technology. *Front. Neurosci.* 11:123. doi: 10.3389/fnins.2017.00123

Sprengers, J., Gaggero, A., Sahin, D., Jahanmirinejad, S., Frucci, G., Mattioli, F., et al. (2011). Waveguide superconducting single-photon detectors for integrated quantum photonic circuits. *Appl. Phys. Lett.* 99:181110. doi: 10.1063/1.3657518

Stein, R. B., Gossen, E. R., and Jones, K. E. (2005). Neuronal variability: noise or part of the signal? *Nat. Rev. Neurosci.* 6, 389–397. doi: 10.1038/nrn1668

Strukov, D., Snider, G., Stewart, D., and Williams, R. (2008). The missing memristor found. *Nature*. 453:80. doi: 10.1038/nature06932

Stuart, G., and Spruston, N. (2015). Dendritic integration: 60 years of progress. *Nat. Neurosci.* 18:1713. doi: 10.1038/nn.4157

Sumikura, H., Kuramochi, E., Taniyama, H., and Notomi, M. (2014). Ultrafast spontaneous emission of copper-doped silicon enhanced by an optical nanocavity. *Sci. Rep.* 4:5040. doi: 10.1038/srep05040

Sun, X., Liu, J., Kimerling, L. C., and Michel, J. (2009a). Direct gap photoluminescence of n-type tensile-strained ge-on-si. *Appl. Phys. Lett.* 95, 011911. doi: 10.1063/1.3170870

Sun, X., Liu, J., Kimerling, L. C., and Michel, J. (2009b). Room-temperature direct bandgap electroluminesence from ge-on-si light-emitting diodes. *Opt. Lett.* 34, 1198–1200. doi: 10.1364/OL.34.001198

Sun, X., Liu, J., Kimerling, L. C., and Michel, J. (2009c). Toward a germanium laser for integrated silicon photonics. *IEEE. Select. Top. Quantum Electron.* 16, 124–131. doi: 10.1109/JSTQE.2009.2027445

Szelag, B., Hassan, K., Adelmini, L., Ghegin, E., Rodriguez, P., Nemouchi, F., et al. (2019). Hybrid iii-V/Silicon Technology for Laser Integration on a 200-mm Fully CMOS-Compatible Silicon Photonics Platform. *IEEE J. Sel. Top. Quant. Electron.* 25:8201210. doi: 10.1109/JSTQE.2019.2904445

Tang, M., Park, J.-S., Wang, Z., Chen, S., Jurczak, P., Seeds, A., et al. (2019). Integration of iii-v lasers on si for si photonics. *Progr. Quantum Electron.* 66:1–18. doi: 10.1016/j.pquantelec.2019.05.002

Tani, K., Oda, K., Deura, M., and Ido, T. (2021). Enhanced room-temperature electroluminescence from a germanium waveguide on a silicon-on-insulator diode with a silicon nitride stressor. *Opt. Express* 29, 3584–3595. doi: 10.1364/OE.415230

Tolpygo, S., Bolkhovsky, V., Oates, D., Rastogi, R., Zarr, S., Day, A., et al. (2018). Superconductor Electronics Fabrication Process with MoN_x Kinetic Inductors and Self-Shunted Josephson Junctions. *IEEE Trans. Appl. Supercond.* 28, 1100212. doi: 10.1109/TASC.2018.2809442

Tolpygo, S., Bolkhovsky, V., Rastogi, R., Zarr, S., Day, A., Golden, E., et al. (2019). Planarized Fabrication Process With Two Layers of SIS Josephson Junctions

and Integration of SIS and SFS π-Junctions. *IEEE Trans. Appl. Supercond*. 29, 1101208. doi: 10.1109/TASC.2019.2901709

Tolpygo, S. K. (2016). Superconductor digital electronics: Scalability and energy efficiency issues. *Low Temperature Phys*. 42, 361–379. doi: 10.1063/1.4948618

Toomey, E., Segall, K., and Berggren, K. K. (2019). Design of a power efficient artificial neuron using superconducting nanowires. *Front. Neurosci*. 13:933. doi: 10.3389/fnins.2019.00933

Upadhyay, N. K., Jiang, H., Wang, Z., Asapu, S., Xia, Q., and Joshua Yang, J. (2019). Emerging memory devices for neuromorphic computing. *Adv. Mater. Technol*. 4, 1800589. doi: 10.1002/admt.201800589

van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274:1724. doi: 10.1126/science.274.5293.1724

Verma, V., Marsili, F., Harrington, S., Lita, A., Mirin, R., and Nam, S. (2012). A three-dimensional, polarization-insensitive superconducting nanowire avalanche photodetector. *Appl. Phys. Lett*. 101:251114. doi: 10.1063/1.4768788

Vetter, A., Ferrari, S., Rath, P., Alaee, R., Kahl, O., Kovalyuk, V., et al. (2016). Cavity-enhanced and ultrafast superconducting single-photon detectors. *Nano Lett*. 16, 7085–7092. doi: 10.1021/acs.nanolett.6b03344

Vinet, M., Batude, P., Tabone, C., Previtali, B., LeRoyer, C., Pouydebasque, A., et al. (2011). 3D monolithic integration: Technological challenges and electrical results. *Microelectron. Eng*. 88:331. doi: 10.1016/j.mee.2010.10.022

Virgilio, M., Manganelli, C., Grosso, G., Pizzi, G., and Capellini, G. (2013). Radiative recombination and optical gain spectra in biaxially strained n-type germanium. *Phys. Rev. B* 87, 235313. doi: 10.1103/PhysRevB.87.235313

Virot, L., Crozat, P., Fédéli, J.-M., Hartmann, J.-M., Marris-Morini, D., Cassan, E., et al. (2014). Germanium avalanche receiver for low power interconnects. *Nat. Commun*. 5, 1–6. doi: 10.1038/ncomms5957

Vogels, T., Rajan, K., and Abbott, L. (2005). Neural network dynamics. *Annu. Rev. Neurosci*. 28:357. doi: 10.1146/annurev.neuro.28.061604.135637

Vogelstein, R., Mallik, U., Vogelstein, J., and Cauwenberghs, G. (2007). Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses. *IEEE Trans. Neural Netw*. 18:253. doi: 10.1109/TNN.2006.883007

Walters, R. J., Bourianoff, G. I., and Atwater, H. A. (2005). Field-effect electroluminescence in silicon nanocrystals. *Nat. Mater*. 4:143–146. doi: 10.1038/nmat1307

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. *arXiv [Preprint] arXiv*:1812.08011.

Warga, J., Li, R., Basu, S., and Dal Negro, L. (2008). Electroluminescence from silicon-rich nitride/silicon superlattice structures. *Appl. Phys. Lett*. 93, 151116. doi: 10.1063/1.3003867

Wollman, E., Verma, V., Lita, A., Farr, W., Shaw, M., Mirin, R., et al. (2019). Kilopixel array of superconducting nanowire single-photon detectors. *Opt. Express* 27:35279. doi: 10.1364/OE.27.035279

Yang, J., Strukov, D., and Stewart, D. (2012). Memristive devices for computing. *Nat. Nanotech*. 8:13. doi: 10.1038/nnano.2012.240

Yin, S., Kim, Y., Han, X., Barnaby, H., Yu, S., Luo, Y., et al. (2019). Monolithically integrated rram-and cmos-based in-memory computing optimizations for efficient deep learning. *IEEE Micro* 39, 54–63. doi: 10.1109/MM.2019.2943047

Young, A. R., Dean, M. E., Plank, J. S., and Rose, G. S. (2019). A review of spiking neuromorphic hardware communication systems. *IEEE Access* 7, 135606–135620. doi: 10.1109/ACCESS.2019.2941772

Zahoor, F., Azni Zulkifli, T. Z., and Khanday, F. A. (2020). Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications. *Nanoscale Res. Lett*. 15, 1–26. doi: 10.1186/s11671-020-03299-9

Zhang, J., Haq, B., O'Callaghan, J., Gocalinska, A., Pelucchi, E., Trindade, A., et al. (2018a). Transfer-printing-based integration of a III-V-on-silicon distributed feedback laser. *Opt. Express* 26:8821. doi: 10.1364/OE.26.008821

Zhang, J., Muliuk, G., Juvert, J., Kumari, S., Goyvaerts, J., Haq, B., et al. (2019). Iii-v-on-si photonic integrated circuits realized using micro-transfer-printing. *APL Photonics* 4, 110803. doi: 10.1063/1.5120004

Zhang, Y., Ling, Y., Zhang, Y., Shang, K., and Yoo, S. (2018b). High-density wafer-scale 3-d silicon-photonic integrated circuits. *IEEE J. Select. Top. Quantum Electron*. 24:8200510. doi: 10.1109/JSTQE.2018.2827784

Zhang, Y., Samanta, A., Shang, K., and Yoo, S. B. (2020a). Scalable 3d silicon photonic electronic integrated circuits and their applications. *IEEE J. Select. Top. Quantum Electron*. 26, 1–10. doi: 10.1109/JSTQE.2020.2975656

Zhang, Y., Wang, Z., Zhu, J., Yang, Y., Rao, M., Song, W., et al. (2020b). Brain-inspired computing with memristors: Challenges in devices, circuits, and systems. *Appl. Phys. Rev*. 7, 011308. doi: 10.1063/1.5124027

Zhao, J., Xie, Y., and Zou, Q. (2015). Overview of 3-D architecture design opportunities and techniques. *IEEE Design Test* 34, 60. doi: 10.1109/MDAT.2015.2463282

Zhirnov, V. V., Cavin, R. K., Hutchby, J. A., and Bourianoff, G. I. (2003). Limits to binary logic switch scaling-a gedanken model. *Proc. IEEE* 91, 1934–1939. doi: 10.1109/JPROC.2003.818324

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers in Artificial Intelligence

ORIGINAL RESEARCH
published: 09 September 2021
doi: 10.3389/frai.2021.699148

Enabling Training of Neural Networks on Noisy Hardware

Tayfun Gokmen *

IBM Research AI, Yorktown Heights, NY, United States

Deep neural networks (DNNs) are typically trained using the conventional stochastic gradient descent (SGD) algorithm. However, SGD performs poorly when applied to train networks on non-ideal analog hardware composed of resistive device arrays with non-symmetric conductance modulation characteristics. Recently we proposed a new algorithm, the Tiki-Taka algorithm, that overcomes this stringent symmetry requirement. Here we build on top of Tiki-Taka and describe a more robust algorithm that further relaxes other stringent hardware requirements. This more robust second version of the Tiki-Taka algorithm (referred to as TTv2) 1. decreases the number of device conductance states requirement from 1000s of states to only 10s of states, 2. increases the noise tolerance to the device conductance modulations by about 100x, and 3. increases the noise tolerance to the matrix-vector multiplication performed by the analog arrays by about 10x. Empirical simulation results show that TTv2 can train various neural networks close to their ideal accuracy even at extremely noisy hardware settings. TTv2 achieves these capabilities by complementing the original Tiki-Taka algorithm with lightweight and low computational complexity digital filtering operations performed outside the analog arrays. Therefore, the implementation cost of TTv2 compared to SGD and Tiki-Taka is minimal, and it maintains the usual power and speed benefits of using analog hardware for training workloads. Here we also show how to extract the neural network from the analog hardware once the training is complete for further model deployment. Similar to Bayesian model averaging, we form analog hardware compatible averages over the neural network weights derived from TTv2 iterates. This model average then can be transferred to another analog or digital hardware with notable improvements in test accuracy, transcending the trained model itself. In short, we describe an end-to-end training and model extraction technique for extremely noisy crossbar-based analog hardware that can be used to accelerate DNN training workloads and match the performance of full-precision SGD.

Keywords: learning algorithms, training algorithms, neural network acceleration, Bayesian neural network, in-memory computing, on-chip learning, crossbar arrays, memristor

OPEN ACCESS

Edited by:
Oliver Rhodes,
The University of Manchester,
United Kingdom

Reviewed by:
Shimeng Yu,
Georgia Institute of Technology,
United States
Emre O. Neftci,
University of California, Irvine,
United States

***Correspondence:**
Tayfun Gokmen
tgokmen@us.ibm.com

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: *22 April 2021*
Accepted: *16 August 2021*
Published: *09 September 2021*

Citation:
Gokmen T (2021) Enabling Training of Neural Networks on Noisy Hardware. Front. Artif. Intell. 4:699148. doi: 10.3389/frai.2021.699148

INTRODUCTION

Deep neural networks (DNNs) (LeCun et al., 2015) have achieved tremendous success in multiple domains outperforming other approaches and even humans (He et al., 2015) at many problems: object recognition, video analysis, and natural language processing are only a few to mention. However, this success was enabled mainly by scaling the DNNs and datasets to extreme sizes, and therefore, it came at the expense of needing immense computation power and time. For instance, the amount of compute required to train a single GPT-3 model composed of 175B parameters is

tremendous: 3,600 Petaflops/s-days (Brown et al., 2020), equivalent to running 1,000 state-of-the-art NVIDIA A100 GPUs, each delivering 150 Teraflops/s performance for about 24 days. Hence, today's and tomorrow's large models are costly to train both financially and environmentally on currently available hardware (Strubell et al., 2019), begging for faster and more energy-efficient solutions.

DNNs are typically trained using the conventional stochastic gradient descent (SGD) and backpropagation (BP) algorithm (Rumelhart et al., 1986). During DNN training, matrix-matrix multiplications; hence repeated multiply and add operations dominate the total workload. Therefore, regardless of the underlying technology, realizing highly optimized multiply and add units and sustaining many of these units with appropriate data paths is practically the only game everybody plays while proposing new hardware for DNN training workloads (Sze et al., 2017).

One approach that has been quite successful in the past few years is to design highly optimized digital circuits using the conventional CMOS technology that leverages reduced-precision arithmetic for the multiply and add operations. These techniques are already employed to some extent by current GPUs (Nvidia, 2021) and other application-specific-integrated-circuits (ASIC) designs, such as TPUs (Cloud TPU, 2007) and IPUs (Graphcore, 2021). There are also many research efforts extending the boundaries of the reduced precision training, using hybrid 8-bit (Sun et al., 2019) and 4-bit (Sun et al., 2020) floating-point and 5-bit logarithmically scaled (Miyashita et al., 2016) number formats.

Alternative to digital CMOS, hardware architectures composed of novel resistive cross-point device arrays have been proposed that can deliver significant power and speed benefits for DNN training (Gokmen and Vlasov, 2016; Haensch et al., 2019; Burr et al., 2017; Burr et al., 2015; Yu, 2018). We refer to these cross-point devices as resistive processing unit [RPU (Gokmen and Vlasov, 2016)] devices as they can perform all the multiply and add operations needed for training by relying on physics. Out of all multiply and add operations during training, 1/3 are performed during forward propagation, 1/3 are performed during error backpropagation, and finally, 1/3 are performed during gradient computation. RPU devices use Ohm's law and Kirchhoff's law (Steinbuch, 1961) to perform the multiply and add needed for the forward propagation and error backpropagation. However, more importantly, *RPUs use the device conductance modulation and memory characteristics to perform the multiply and add needed during the gradient computation* (Gokmen and Vlasov, 2016).

Unfortunately, RPU based crossbar architectures have had only minimal success so far. That is mainly because the training accuracy on this imminent analog hardware strongly depends on the cross-point elements' conductance modulation characteristics when the conventional SGD algorithm is used. One of the key requirements is that these devices must symmetrically change conductance when subjected to positive or negative pulse stimuli (Gokmen and Vlasov, 2016; Agarwal et al., 2016). Theoretically, it is shown that only symmetric devices provide an unbiased gradient calculation and accumulation needed for the SGD algorithm. Whereas any non-symmetric device characteristic modifies the optimization objective and hampers the convergence of SGD based training (Gokmen and Haensch, 2020; Onen et al., 2021).

Many different solutions are proposed to tackle the SGD's converge problem on crossbar arrays. First, widespread efforts to engineer resistive devices with symmetric modulation characteristics have been made (Fuller et al., 2019; Woo and Yu, 2018; Grollier et al., 2020), but a mature device technology with the desired behavior remains to be seen. Second, many high-level mitigation techniques have been proposed to overcome the device asymmetry problem. One critical issue with these techniques is the serial access to cross-point elements either one-by-one or row-by-row (Ambrogio et al., 2018; Agarwal et al., 2017; Yu et al., 2015). Serial operations such as reading conductance values individually, engineering update pulses to force symmetric modulation artificially, and carrying or resetting weights periodically come with a tremendous overhead for large networks. Alternatively, there are approaches that perform the gradient computation outside the arrays using digital processing (Nandakumar et al., 2020). Note that irrespective of the DNN architecture, 1/3 of the whole training workload is in the gradient computation. For instance, for the GPT-3 network, 1,200 Petaflops/s-days are required solely for gradient computation throughout the training. Consequently, these approaches cannot deliver much more performance than the fully digital reduced-precision alternatives mentioned above. In short, there exist solutions possibly addressing the convergence issue of SGD on non-symmetric device arrays. However, they all defeat the purpose of performing the multiply and add operations on the RPU device and lose the performance benefits.

In contrast to all previous approaches, we recently proposed a new training algorithm, the so-called Tiki-Taka algorithm (Gokmen and Haensch, 2020), that performs all three cycles (forward propagation, error backpropagation, and gradient computation) on the RPU arrays using the physics and converges with non-symmetric device arrays. Tiki-Taka works very differently from SGD, and we showed in another study that non-symmetric device behavior plays a useful role in the convergence of Tiki-Taka (Onen et al., 2021).

Here, we build on top of Tiki-Taka and present a more robust second version that relaxes other stringent hardware issues by orders of magnitude, namely the limited number of states of RPU devices and noise. We refer to this more robust second version of the Tiki-Taka algorithm as TTv2 for the rest of the paper. In the first part of the paper, we focus on training and present TTv2 algorithm details and provide simulation results at various hardware settings. We tested TTv2 on various network architectures, including fully connected, convolutional, and LSTMs, although the presented results focus on the more challenging LSTM network. TTv2 shows significant improvements in the training accuracy compared to Tiki-Taka, even at much more challenging hardware settings. In the second part of the paper, we show an analog-hardware-friendly technique to extract the trained model from the noisy hardware. We also generalize this technique and apply it over TTv2 iterates and extract each weight's time average from a

particular training period. These weight averages provide a model that approximates the Bayesian model average, and it outperforms the trained model itself. *With this new training algorithm and accurate model extraction technique, we show that the noisy analog hardware composed of RPU device arrays can provide scalable training solutions that match the performance of full-precision SGD.*

PART I: Training

In this section, we first give an overview of the device arrays and device update characteristics used for training. Then we present a brief background on Tiki-Taka. Finally, we detail TTv2 and provide comprehensive simulation results on an LSTM network at various hardware settings.

Device Arrays and Conductance Modulation Characteristics

Resistive crossbar array of devices performs efficient matrix-vector multiply ($\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$) using Ohm's law and Kirchhoff's law. The device array's stored conductance values form a matrix ($\boldsymbol{W}$), whereas the input vector ($\boldsymbol{x}$) is transmitted as voltage pulses through the columns, and the resulting vector ($\boldsymbol{y}$) is read as current signals from the rows. However, only positive conductance values are allowed physically. Therefore, to encode both positive and negative matrix elements, a pair of devices is operated in differential mode. With the help of the peripheral circuits supplying the voltage inputs and reading out the differential current signals, logical matrix elements (w_{ij}) are mapped to physical conductance pairs as

$$w_{ij} = \mathrm{K}\left(g_{ij} - g_{ij,ref}\right) \tag{1}$$

where K is a global gain factor controlled by the periphery, and g_{ij} and $g_{ij,ref}$ are the conductance values stored at each pair corresponding to the ith row and jth column. Moreover, crossbar arrays can be easily operated in the transpose mode by changing the periphery's input and output directions. As a result, a pair of arrays with the supporting peripheral circuits provide a logical matrix (also referred to as a single tile) that any algorithm can utilize to perform a series of matrix-vector multiplications (mat-vec) using $\boldsymbol{W}$ and $\boldsymbol{W}^{\boldsymbol{T}}$.

For training algorithms, the efficient update of the stored matrix elements is also an essential component. Therefore, device conductance modulation and memory characteristics are utilized to implement a local and parallel update on RPU arrays. During the update cycle, input signals are encoded as a series of voltage pulses and simultaneously supplied to the array's rows and columns. Note that the voltage pulses are applied only to the first set of RPU devices, and the reference devices are kept constant. As a result of voltage pulse coincidence, the corresponding RPU device changes its conductance by a small amount bi-directionally, depending on the voltage polarity. This incremental change in device conductance results in an incremental change in the stored weight value, and the RPU response is governed by **Eq. 2**.

$$w_{ij} \leftarrow w_{ij} \mp \Delta w_{min,ij} F_{ij}\left(w_{ij}\right) - \left|\Delta w_{min,ij}\right| G_{ij}\left(w_{ij}\right) \tag{2}$$

In **Eq. 2**, $\mp$ sign is decided by the external voltage pulse polarity, whereas $\Delta w_{min,ij}$ is the incremental weight change due to single pulse coincidence, and $F_{ij}(w_{ij})$ and $G_{ij}(w_{ij})$ are the symmetric (additive) and antisymmetric (subtractive) combinations of the positive and negative conductance modulation characteristics (Gokmen and Haensch, 2020), all of which are the properties of the updated device corresponding to the ith row and jth column. **Eq. 2** is very general and governs the computation (hardware-induced update) performed by the tile for all sorts of RPU device behaviors, assuming the device conductance modulation characteristics are some function of the device conductance state. If the conductance modulations are much smaller than the whole conductance range of operation, **Eq. 3** can be derived from **Eq. 2**.

$$w_{ij} \leftarrow w_{ij} + \eta\left[\delta_i \times x_j\right] F_{ij}\left(w_{ij}\right) - \eta\left|\left[\delta_i \times x_j\right]\right| G_{ij}\left(w_{ij}\right) \tag{3}$$

In **Eq. 3**, x_j and δ_i represent the input values used for updates for each column and row, respectively corresponding to activations and errors calculated in the forward and backward cycles, and η is a scalar controlling the strength of the update, all of which are inputs to pulse generation circuitry at the periphery. Here, we use the stochastic pulsing scheme proposed in Ref Gokmen and Vlasov (2016), and during the parallel update, the number of pulses generated by the periphery is bounded by $n_{pulse} = \left\lceil \eta \max(|\delta_i|)\max(|x_j|)/\mu_{\Delta w} \right\rceil$, where $\mu_{\Delta w}$ is the mean of $\Delta w_{min,ij}$ for the whole tile. Using n_{pulse} stochastic translators generate pulses with the correct probability; therefore, **Eq. 3** is valid in expectation. Whereas in the limit of a single pulse coincidence, the RPU response is governed by **Eq. 2**.

Figure 1A illustrates a pulse response of a linear and symmetric device, where $F(w) = 1$ and $G(w) = 0$, and the hardware-induced update rule simplifies to the SGD update rule of $w_{ij} \leftarrow w_{ij} + \eta[\delta_i \times x_j]$. In the literature, this kind of device behavior is usually referred to as the "ideal" device required for SGD. For a non-linear but symmetric device, $F(w)$ deviates from unity and becomes a function of w, but $G(w)$ remains zero. For non-symmetric devices, $G(w)$ also deviates from zero and becomes a function of w, hence differing from the form required by SGD. **Figure 1B** illustrates an exponentially saturating non-symmetric device where $w_{ij} \leftarrow w_{ij} + \eta[\delta_i \times x_j] - \eta\left|[\delta_i \times x_j]\right| w$ provides the computation performed by this device. Although this form of update behavior causes convergence issues for SGD, Tiki-Taka trains DNNs successfully with all sorts of non-symmetric devices (Gokmen and Haensch, 2020). Therefore, in contrast to SGD, all sorts of non-symmetric device behaviors can be considered "ideal" for Tiki-Taka.

Tiki-Taka's training performance depends on the successful application of the symmetry point shifting technique (Kim et al., 2019), which guarantees $G(w = 0) = 0$ for all elements in the tile. This behavior is illustrated for the device in **Figure 1B**, where the strengths of the positive and negative weight increments are equal

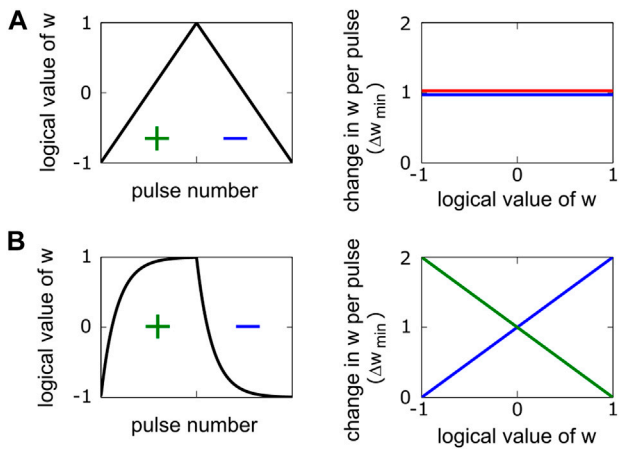


FIGURE 1 | Pulse responses and weight modulation characteristics are illustrated for two different devices. **(A)** Symmetric and linear device: Weight increments (red) and decrements (blue) are equal in size and do not depend on the weight. **(B)** Exponentially saturating device: Weight increments and decrements both have linear dependencies on the weight. However, there exists a single weight value at which the strengths of the weight increment and decrement are equal. This point is called the symmetry point, and it is at $w = 0$ for the illustrated example.

in size at $w = 0$. The symmetry point shifting is achieved by programming the reference device conductance to a value corresponding to the updated device's symmetry point. For the rest of the paper, we assume the symmetry point shifting is also applied in the context of TTv2. Although we developed techniques to eliminate this requirement, it is beyond the scope of this paper and will be published elsewhere.

Algorithms

SGD, Tiki-Taka, and TTv2 all use the error backpropagation, but they process the gradient information differently and hence are fundamentally distinct algorithms. **Figures 2A,B** show schematics of SGD and Tiki-Taka dynamics (iterations), respectively. Tiki-Taka replaces each weight matrix W of SGD with two matrices (referred to as matrix A and C) and creates a coupled dynamical system by exchanging information between the two. As shown in Ref Onen et al. (2021), the non-symmetric behavior is a valuable and required property of the device in the Tiki-Taka dynamics. During the information exchange between the two systems, device asymmetry creates a dissipation mechanism, resulting in minimization of the system's total energy (Hamiltonian); hence Tiki-Taka is also called Stochastic Hamiltonian Descent (Onen et al., 2021). However, the noise introduced during the transfer of the information (processed gradients) from A to C caused additional test error for Tiki-Taka and needed to be addressed (Gokmen and Haensch, 2020).

The schematic in **Figure 2C** illustrates the TTv2 dynamics, highlighting our main contribution. TTv2 introduces an additional stage (H), between the transfer from A to C, which performs integration in the digital domain, providing a low pass filtering function. Furthermore, the model's parameters are stored solely on C and only updated if H reaches a threshold value. Because of these modifications in TTv2, the model's parameters are updated more slowly but with higher confidence, bringing significant benefits against various hardware noise issues. Details of the algorithm are provided below.

Tiki-Taka Algorithm

Algorithm 1 outlines the details of the Tiki-Taka algorithm. Tiki-Taka uses two matrices, A and C, and the neural network parameters are defined by $\boldsymbol{W} = \gamma \boldsymbol{A} + \boldsymbol{C}$, where γ is a scalar hyperparameter set between [0,1]. Using W, Tiki-Taka computes the activations ($\boldsymbol{x}$) and the error signals ($\boldsymbol{\delta}$) by utilizing the conventional backpropagation algorithm. The activation and error computations are identical to SGD and therefore omitted from the algorithm description. Also, there are multiple layers, but **Algorithm 1** only illustrates the operations performed on a single layer for simplicity. After performing the forward propagation and the error backpropagation on A and C (lines 8 and 9), Tiki-Taka updates only A by employing the hardware-induced parallel update (line 10) using $\boldsymbol{x}$ and $\boldsymbol{\delta}$. η_a is the learning rate used for updating A. These operations are repeated for ns times, a hyperparameter of Tiki-Taka. After every ns update on A, an analog mat-vec is performed on A with an input vector $\boldsymbol{u}$, resulting in a vector $\boldsymbol{v}$ (line 14). The vector $\boldsymbol{u}$ is generated each time locally, and it is either a one-hot encoded vector or a column vector of a Hadamard matrix used in a cyclic fashion.

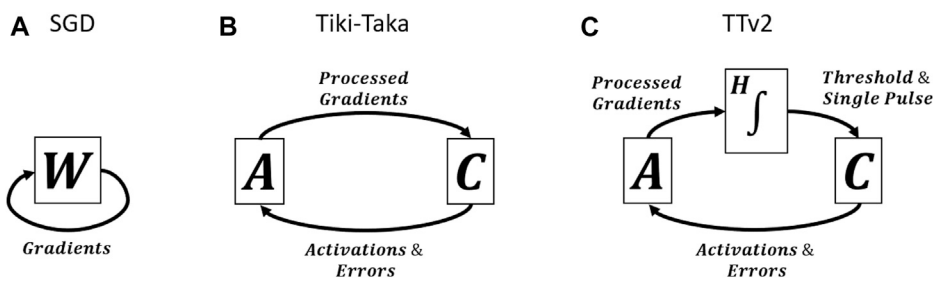


FIGURE 2 | Schematics of SGD, Tiki-Taka, and TTv2 dynamics.

Algorithm 1: Tiki-Taka	
1 : initialize A = 0 after symmetry point shifting	// analog matrix A initialization
2 : initialize C to random values	// analog matrix C initialization
3 : k = 0	
4 : t = 0	
5 : nr = Number of rows in A/C	
6 : ns = Hyperparameter - Number of update cycles	
7 : **For** each data in the training dataset	
8 : $\boldsymbol{y} = (\gamma \boldsymbol{A} + \boldsymbol{C})\boldsymbol{x}$	// parallel analog mat-vec using A and C
9 : $\boldsymbol{z} = (\gamma \boldsymbol{A} + \boldsymbol{C})^T \boldsymbol{\delta}$	// parallel analog mat-vec using A and C transpose
10: $a_{ij} \leftarrow a_{ij} + \eta_a [\delta_i \times x_j] F_{ij}(a_{ij}) - \eta_a \lvert[\delta_i \times x_j]\rvert G_{ij}(a_{ij})$	// parallel hardware-induced array update on A
11: k = mod(k+1, ns)	
12: **If** (k = 0)	
13: $\boldsymbol{u}$ = prepare_vector(t)	// prepare new vector $\boldsymbol{u}$ using t
14: $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$	// parallel analog mat-vec using A
15: $c_{ij} \leftarrow c_{ij} + \eta_c [f(v_i) \times u_j] F_{ij}(c_{ij}) - \eta_c \lvert[f(v_i) \times u_j]\rvert G_{ij}(c_{ij})$	// parallel hardware-induced array update on C
16: t = mod(t+1, nr)	
17: **end**	
18: **end**	

Using the generated $\boldsymbol{u}$ vector and the result of $f(\boldsymbol{v})$, C is updated by employing the hardware-induced parallel update (line 15). $f(\boldsymbol{v})$ is a pointwise function: $f(v_i) = \begin{cases} v_i, & \text{if } |v_i| \geq T \\ 0, & \text{otherwise} \end{cases}$ where T is set to the mat-vec noise. η_c is the learning rate used for updating C. These operations are repeated for the data examples in the training dataset for multiple epochs until a converge criteria is met. Following the same practices described in Ref Gokmen and Haensch (2020), here we also use the one-hot encoded $\boldsymbol{u}$ vectors and the thresholding $f(\boldsymbol{v})$ for the LSTM simulations.

TTv2 Algorithm

Algorithm 2 outlines the details of the TTv2 algorithm. In addition to A and C matrices allocated on analog arrays, TTv2 also allocates another matrix H in the digital domain. This matrix H is used to implement a low pass filter while transferring the gradient information processed by A to C. In contrast to Tiki-Taka, TTv2 uses only the C matrix to encode the neural network's parameters, corresponding to $\gamma = 0$. Therefore, the activation ($\boldsymbol{x}$) and error ($\boldsymbol{\delta}$) computations are performed using C (lines 10 and 11). TTv2 does not change the updates performed on A. After ns updates, a mat-vec is performed on A. Unlike Tiki-Taka, TTv2 only uses a one-hot encoded $\boldsymbol{u}$ vector while performing a mat-vec on A. This provides a noisy estimate of a single row of A, and the results are stored in $\boldsymbol{v}$. After this step, the significant distinction between Tiki-Taka and TTv2 appears. Instead of using $\boldsymbol{u}$ and $\boldsymbol{v}$ to update C, TTv2 first accumulates $\boldsymbol{v}$ (after scaling with η_c) on H's corresponding row, referred to as H(row = t). During this digital vector-vector addition, the magnitude of any element in H(row = t) may exceed unity. In that case, the corresponding elements are reset back to zero, and a single pulse parallel update on C is performed. The C update of TTv2 uses the sign information of the elements that grew in amplitude beyond one and the row information t. After these steps, TTv2 loops back and repeats these operations for other data examples until it reaches convergence.

Array Model

We use a device model like the one presented in **Figure 1B** but with significant array level variability and noise for the training simulations. We simulate stochastic translators at the periphery during the update, and each coincidence event triggers an incremental weight change on the corresponding RPU as described below. We also introduce noise and signal bounds during the matrix-vector multiplications performed on the arrays.

During the update, the weight increments (Δw_{ij}^+) and decrements (Δw_{ij}^-) are assumed to be functions of the current weight value. For the positive branch $\Delta w_{ij}^+ = \Delta w_{min,ij}(1 - slope_{ij}^+ \times w_{ij})$ and for the negative branch $\Delta w_{ij}^- = \Delta w_{min,ij}(1 + slope_{ij}^- \times w_{ij})$, where $slope_{ij}^+$ and $slope_{ij}^-$ are the slopes that control the dependence of the weight changes on the current weight values, and $\Delta w_{min,ij}$ is the weight change due to a single coincidence event at the symmetry point. This model results in three unique parameters for each RPU element. All these parameters are sampled independently using a unit Gaussian random variable and then used throughout the training, providing device-to-device variability. The slopes are obtained using $slope_{ij}^+ = \mu_s(1 + \sigma_s \xi_{ij}^+)$ and $slope_{ij}^- = \mu_s(1 + \sigma_s \xi_{ij}^-)$, where $\mu_s = 1.66$, σ_s is set to 0.1, 0.2, or 0.3 for different experiments, and ξ are the independent random samples. The simulation results were insensitive to σ_s; therefore, we only show results corresponding to $\sigma_s = 0.2$. The weight increments at the symmetry point are obtained using $\Delta w_{min,ij} = \mu_{\Delta w}(1 + \sigma_{\Delta w} \xi_{ij})$, where $\sigma_{\Delta w} = 0.3$ and $\mu_{\Delta w}$ is the array average varied from 0.6 $\times 10^{-4}$ up to 0.15 for different experiments to study the effects number of states on training accuracy. We define the number of states as the ratio of the nominal weight range to the nominal weight increment at the symmetry point; therefore, $2/(\mu_s \mu_{\Delta w})$ provides the average number of states. Note that this definition of the number of states is very different from the definition used for devices developed for memory applications, and it should not be compared against multi-bit storage elements. Besides, additional Gaussian noise is introduced to each weight increment and decrement to capture the cycle-to-cycle noise: For the

Algorithm 2: TTv2	
1 : initialize H = 0	// digital matrix H initialization, $\boldsymbol{H}(row = t)$ refers t^{th} row in H.
2 : initialize A = 0 after symmetry point shifting	// analog matrix A initialization
3 : initialize C to random values	// analog matrix C initialization
4 : k = 0	
5 : t = 0	
6 : nr = Number of rows in A/C/H	
7 : ns = Hyperparameter – Number of update cycles	
9 : **For** each data in the training dataset	
10: $\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$	// parallel analog mat-vec using C
11: $\boldsymbol{z} = \boldsymbol{C}^T\boldsymbol{\delta}$	// parallel analog mat-vec using C transpose
12: $a_{ij} \leftarrow a_{ij} + \eta_a[\delta_i \times x_j]F_{ij}(a_{ij}) - \eta_a\lvert[\delta_i \times x_j]\rvert G_{ij}(a_{ij})$	// parallel hardware-induced array update on A
13: k = mod(k+1, ns)	
14: **If** (k = 0)	
15: $\boldsymbol{u}$ = prepare_one_hot_vector(t)	// prepare new one hot encoded vector $\boldsymbol{u}$ using t
16: $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$	// parallel analog mat-vec using A
17: $\boldsymbol{H}(row = t) = \boldsymbol{H}(row = t) + \eta_c\boldsymbol{v}$	// digital vector-vector summation
18: **If** ($\lvert h_{it}\rvert > 1$) **for any element in** $\boldsymbol{H}(row = t)$	// check if an element in $\boldsymbol{H}(row = t)$ is beyond [-1,1]
19: $c_{it} \leftarrow c_{it} + sign(h_{it})c_{min,it}F_{it}(c_{it}) - \lvert\Delta c_{min,it}\rvert G_{it}(c_{it})$	// parallel hardware-induced array update on C, only elements corresponding to row t and columns that are beyond [-1,1] are updated with a single pulse
20: $h_{it} = 0$	// reset elements in $\boldsymbol{H}(row = t)$ that are beyond [-1,1]
21: **end**	
22: t = mod(t+1, nr)	
23: **end**	
24: **end**	

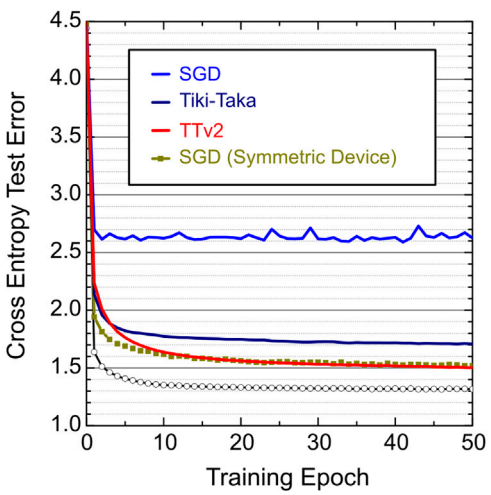


FIGURE 3 | LSTM training simulations for SGD, Tiki-Taka, and TTv2 algorithms. Different color curves use an array model with non-symmetric devices, $\mu_{\Delta w}$ = 0.001 (corresponding to 1,200 states), and the multiplicative cycle-to-cycle update noise at σ_{cycle} = 0.3. The square symbols show the SGD training using linear and symmetric devices where all devices' slope parameters are set to zero while all other array parameters remain unchanged. The open circles are the floating-point baseline.

multiplicative noise model $\Delta w_{ij}^{\mp} \rightarrow \Delta w_{ij}^{\mp}(1 + \sigma_{cycle}\xi)$, whereas for the additive noise model $\Delta w_{ij}^{\mp} \rightarrow \Delta w_{ij}^{\mp} + \Delta w_{min,ij}\sigma_{cycle}\xi$, where σ_{cycle} is set to 0.3 or 1 for different experiments, and ξ is sampled from a unit Gaussian for each coincidence event.

During the matrix-vector multiplications, we inject additive Gaussian noise into each output line to account for analog noise. Therefore, the model becomes $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \sigma_{MV}\boldsymbol{\xi}$, where σ_{MV} = 0.06, corresponding to 10% of the nominal weight maximum ($1/\mu_s$). Moreover, the matrix-vector multiplications are bounded to 20 times the nominal weight maximum to account for signal saturation at the output lines. The input signals are assumed to be between [−1, 1] with a 7-bit input resolution, whereas the outputs are quantized assuming a 9-bit ADC. To mitigate the shortcomings of the signal bounds, we use the noise, bound, and update management techniques described in Ref Gokmen et al. (2017).

Training Simulations

We performed training simulations for fully connected, convolutional, and LSTM networks: the same three networks and datasets studied in Ref Gokmen and Haensch (2020). However, the presented results focus on the most challenging LSTM network referred to as LSTM2-64-WP in Ref Gokmen et al. (2018). This network is composed of two stacked LSTM blocks, each with a hidden state number of 64. Leo Tolstoy's War and Peace (WP) novel is used as a dataset, and it is split into training and test sets as 2,933,246 and 325,000 characters with a total vocabulary of 87 characters. This task performs a character-based language model where the input to the network is a sequence of characters from the WP novel, and the network is trained with the cross-entropy loss function to predict the next character in the sequence. LSTM2-64-WP has three different weight matrices for SGD, and including the

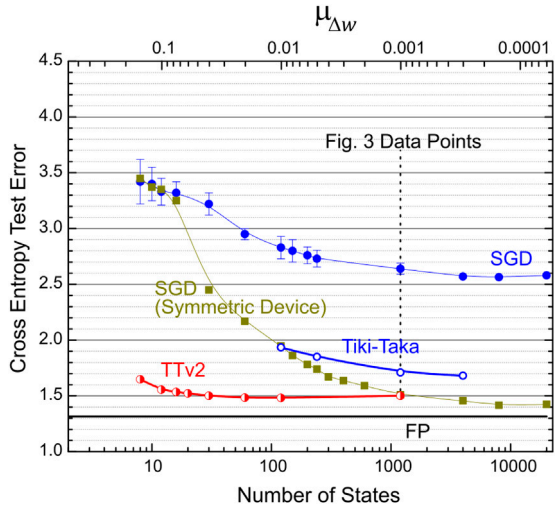


FIGURE 4 | LSTM training simulations for SGD, Tiki-Taka, and TTv2 algorithms as a function of $\mu_{\Delta w}$. 10x increase in $\mu_{\Delta w}$ results in a simultaneous 10x reduction in the number of states and a 10x increase in the cycle-to-cycle update noise. Circles correspond to an array model with non-symmetric devices, whereas squares are for symmetric and linear devices. All symbols report the test error at the end of the 50th epoch, and the error bars capture the test error fluctuations for the last five epochs. Lines are guides to the eye. The floating-point baseline is shown with the black horizontal line at 1.32 test error. After random weight initialization, an untrained network gives ~4.46 test error corresponding to a random guess.

biases, they have sizes $256 \times (64 + 87 + 1)$ and $256 \times (64 + 64 + 1)$ for the two LSTM blocks and $87 \times (64 + 1)$ for the fully connected layer before the softmax activation. Each matrix of SGD maps to two seperate A and C matrices for Tiki-Taka and TTv2.

Figure 3 shows simulation results for SGD, Tiki-Taka, and TTv2 for non-symmetric device arrays with $\mu_{\Delta w} = 0.001$ (corresponding to 1,200 average number of states) and the multiplicative cycle-to-cycle noise $\sigma_{cycle} = 0.3$. Additionally, we simulate the SGD training using symmetric device arrays where all devices' slope parameters are set to zero while all other array parameters remain unchanged. We also note that without changing the analog hardware settings, we virtually remap the nominal weight range from [−0.6, 0.6] to [−2, 2] using the digital scaling trick shown in Ref Rasch et al. (2020) for all LSTM simulations. This remapping slightly increases SGD and Tiki-Taka's training performance compared to the results published in Ref Gokmen and Haensch (2020). We also optimized Tiki-Taka's hyper-parameters to achieve the best possible training performance at this modified weight range.

In **Figure 3**, Tiki-Taka performs significantly better than SGD for non-symmetric devices, but a clear gap exists between the symmetric device SGD and the Tiki-Taka results. This gap is due to the noise during the analog mat-vec performed on A (line 14 of Tiki-Taka). Ref Gokmen and Haensch (2020) showed that the remaining gap closes if the noise during the mat-vec on A is reduced by 10x to $\sigma_{MV} = 0.006$; however, this low noise setting is unrealistic for analog hardware. In contrast, TTv2 shows indistinguishable results compared to the symmetric device SGD, even when the mat-vec noise on A is at $\sigma_{MV} = 0.06$. Therefore, these simulation results prove the benefits of introducing the filtering stage while transferring information from A to C, and TTv2 increases the algorithm's noise tolerance to the mat-vec performed by the analog arrays at least by 10x compared to Tiki-Taka.

To further examine the resilience of TTv2 to other analog hardware issues, namely the number of states and the cycle-to-cycle update noise, we performed training simulations by varying $\mu_{\Delta w}$ many decades from 0.6×10^{-4} to 0.15. This 2,500x increase in $\mu_{\Delta w}$ causes a 2,500x reduction in states' number on RPU devices from 20,000 down to 8. Furthermore, as $\mu_{\Delta w}$ increases, the amount of noise existing during the pulsed updates increases by 2,500x since cycle-to-cycle noise is defined relative to the state definition on each device as described above. **Figure 4** summarizes these simulation results, where the test error at the end of the 50th epoch is reported. For each data point in **Figure 4**, we finetuned each algorithm's hyper-parameters independently and reported the best training results. Both SGD and Tiki-Taka are very sensitive to the number of states and the update noise as the test error increases quickly with an increase in $\mu_{\Delta w}$. Whereas the error for TTv2 remains unchanged for many decades and highlights the orders of magnitude increased tolerance of TTv2 to the limited number of states and the update noise. Compared to SGD and Tiki-Taka, TTv2 is at least 100x more resilient to these two common hardware issues that appear during the update cycle on analog arrays.

Finally, in **Figure 5**, we additionally tested the success of TTv2 at an extremely noisy hardware setting. These simulations assume $\mu_{\Delta w} = 0.08$ corresponding to an average of 15 states, but with an even higher cycle-to-cycle update noise setting with the additive noise model at $\sigma_{cycle} = 1$. **Figures 5A–C** illustrate (for three different devices) the amount of update noise and the array level variability used for TTv2. The blue curves show the evolution of the weights after each pulse during training. The red curves show the sign of the updates and the expected average saturation value for the corresponding device for positive and negative pulses. The saturation values are very different due to array level variability, and the response to each pulse is very noisy due to the additive cycle-to-cycle update noise. As a comparison, we also show the response of a linear and symmetric device with $\sigma_{cycle} = 0.3$ and more than 1,000 states in **Figure 5D**. The noise is not even visible for this device used only for the SGD simulations, further emphasizing the burden imposed on the TTv2 algorithm.

The training simulations in **Figure 5E** show that TTv2 achieves acceptable training results even at these extremely noisy hardware settings. **Figure 5E** also shows a slightly modified TTv2 implementation with a hysteretic threshold that achieves a better result than TTv2. In this modified TTv2 implementation, we only changed line 20 of TTv2 from $h_{it} = 0$ to $h_{it} = sign(h_{it})0.6$. This change makes the thresholding event asymmetric and hysteretic: Back to back same sign updates on C happens with a 0.4 threshold, whereas back to back different sign updates must overcome a threshold of 1.6. These hysteretic updates allow the system to correct itself quickly if the previous update caused an undesired modulation on the weight. Note that the update noise is so large that it may even cause a change in the

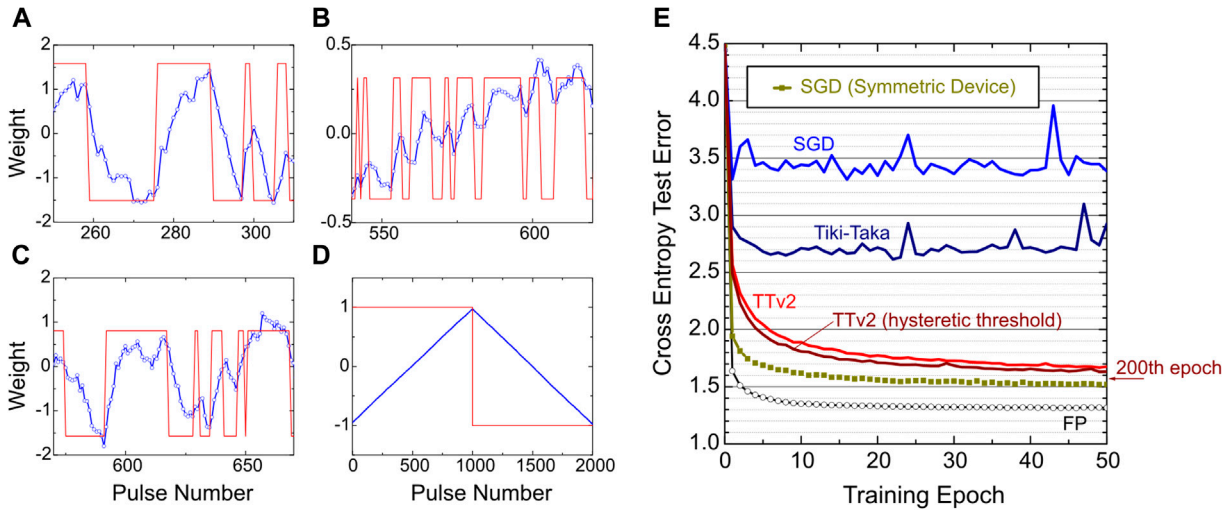


FIGURE 5 | (A, B, C) Blue curves: The evolution of three different weights (corresponding to three different devices with non-symmetric behavior, σ_{cycle} = 1 and about 15 states) during TTv2 training. Red curves show the sign of the updates and the expected average saturation value for the corresponding device. **(D)** The evaluation of a linear and symmetric device with σ_{cycle} = 0.3 and more than 1,000 states. **(E)** LSTM training simulations for SGD, Tiki-Taka, and TTv2 algorithms. Different color curves use an extremely noisy array model with non-symmetric devices, $\mu_{\Delta w}$ = 0.08 (corresponding to 15 states), and the additive cycle-to-cycle update noise with σ_{cycle} = 1. The square symbols show the SGD training baseline from **Figure 3** with symmetric device arrays with 1,200 states and cycle-to-cycle noise at σ_{cycle} = 0.3. The open circles are the floating-point baseline.

weight opposite to the intended direction, as illustrated in **Figures 5A–C**. Furthermore, same sign updates are encouraged to accelerate the learning along the dimensions that have higher confidence.

Finally, we emphasize that, in contrast to SGD and Tiki-Taka, TTv2 only fails gracefully at these extremely challenging hardware settings. We note that the continued training further improves the performance of TTv2 until 200 epochs, and a test error of 1.57 is achieved for the modified TTv2. This test error is almost identical to one achieved by the symmetric device SGD baseline with 1,200 states and many orders of magnitude less noise. All these results show that TTv2 is superior to Tiki-Taka and SGD, especially when the analog hardware becomes noisy and provides a very limited number of states on RPU devices.

Implementation Cost of TTv2

The true benefit of using device arrays for training workloads emerges when the required gradient computation (and processing) step is performed in the array using the RPU device properties. As mentioned in the introduction, the gradient computation is 1/3 of the training operations performed on the weights that the hardware must handle efficiently. Irrespective of the layer type, such as convolution, fully connected, or LSTM, for an $n \times n$ weight matrix in a neural network, each gradient processing step per weight reuse has a computational complexity of $O(n^2)$. RPU arrays perform the required gradient processing step efficiently at $O(1)$ constant time using array parallelism. Specifically, analog arrays deliver $O(1)$ time complexity simply because the array has $O(n^2)$ compute resources (RPU devices). In this scheme, each computation is mapped to a resource, and consequently, RPU arrays trade space complexity for time complexity, whereas computational complexity remains unchanged. As a result of this spatial mapping, crossbar-based analog accelerators require a multi-tile architecture design irrespective of the training algorithm so that each neural network layer and the corresponding weights can be allocated on separate tiles. Nevertheless, RPU arrays provide a scalable solution for a spatially mapped weight stationary architecture for training workloads thanks to the nano-scale device concepts.

As highlighted in **Algorithm 2**, TTv2 uses the same tile operations and therefore running TTv2 on array architectures requires no change in the tile design compared to SGD or Tiki-Taka. Assuming the tile design remains unchanged, a pair of device arrays operated differentially with the supporting peripheral circuits, TTv2 (like Tiki-Taka) requires twice more tiles to allocate A and C separately. However, alternatively, the logical A and C values can be realized using only three devices by sharing a common reference, as described in Ref Onen et al. (2021). In that case, logical A and C matrices can be absorbed into a single tile design composed of three device arrays and operated in a time multiplex fashion. This tile design minimizes or even possibly eliminates the area cost of TTv2 and Tiki-Taka compared to SGD.

In contrast to A and C matrices allocated on analog arrays, H does not require any spatial mapping as it is allocated digitally, and it can reside on an off-chip memory. Furthermore, we emphasize that the digital H processing of TTv2 must not be confused with the gradient computation step. For an $n \times n$ weight matrix in a neural network, the computational complexity of the operations performed on H is only $O(n)$, even for the most aggressive setting of ns = 1. As detailed in **Algorithm 2**, only a single row of H is accessed and processed digitally for ns parallel

array update operations on A. Therefore, H processing has reduced computational complexity compared to gradient computation: $O(n)$ vs. $O(n^2)$. This property differentiates TTv2 from other approaches performing the gradient computation in the digital domain with $O(n^2)$ complexity (Nandakumar et al., 2020). Regardless, the digital H processing in TTv2 brings additional digital computation and memory bandwidth requirements compared to SGD or Tiki-Taka. To understand the extra burden introduced by H in TTv2, we must compare it to the burden already handled by the digital components for the SGD algorithm. We argue that the extra burden introduced in TTv2 is usually only on the order of 1/ns, and the digital components required by the SGD algorithm can also handle the H processing of TTv2.

A weight reuse factor (ws) for each layer in a neural network is determined by various factors, such as time unrolling steps in an LSTM, reuse of filters for different image portions in a convolution, or simply using mini-batches during training. For an $n \times n$ weight matrix with a weight reuse factor of ws, the compute performed on the analog array is $O(n^2.ws)$. In contrast, the storage and processing performed digitally for the activations and error backpropagation are usually $O(n.ws)$. We emphasize that these $O(n.ws)$ compute and storage requirements are common to TTv2, Tiki-Taka, and SGD and are already addressed by digital components.

The digital filter of TTv2 computes straightforward vector-vector additions and thresholds, which require $O(n)$ operations performed only after ns weight reuses. As mentioned above, SGD (likewise Tiki-Taka and TTv2) uses digital units to compute the activations and the error signals, both of which are usually $O(n.ws)$. Therefore, the digital compute needed for the H processing of TTv2 increases the total digital compute by $O(n.ws/ns)$.

Additionally, the filter requires the H matrix to be stored digitally. H is as large as the neural network model and requires off-chip memory storage and access. One may argue that this defeats the purpose of using analog crossbar arrays. However, note that even though the storage requirements for H are $O(n^2)$, the access to H happens one row at a time, which is $O(n)$. Therefore, as long as the memory bandwidth can sustain access to H, the storage requirement is a secondary concern that can easily be addressed by allocating space on external off-chip memory. This increases the required storage capacity from $O(n.ws)$ (only for activations) to $O(n.ws) + O(n^2)$ (activations + H).

Finally, assuming H resides on an off-chip memory, the hardware architecture must provide enough memory bandwidth to access H. As noted in **Algorithm 2**, access to H is very regular, and only a single row of H is needed after ns weight reuses. For SGD (and hence for Tiki-Taka and TTv2), the activations computed in the forward pass are first stored in off-chip memory and then fetched from it to compute the error signals during the backpropagation. The activation store and loads are also usually $O(n.ws)$, and therefore the additional access to H in TTv2 similarly increases required memory bandwidth by about $O(n.ws/ns)$.

In summary, compared to SGD, TTv2 introduces extra digital costs that are only on the order of 1/ns, whereas it brings orders of magnitude relaxation to many stringent analog hardware specs. For instance, ns = 5 provided the best training results for the LSTM network, and for that network, the additional burden introduced to digital compute and memory bandwidth remains less than 20%. For the first convolutional layer of the MNIST problem, ns = 576 is used, making the additional cost negligible (Gokmen and Haensch, 2020). However, we note that the neural networks come in many different flavors, beyond those studied in this manuscript, with different stress points on various hardware architectures. Our complexity arguments should only be used to compare the relative overhead of TTv2 compared to SGD, assuming a fixed analog crossbar-based architecture and particular neural network layers. Detailed power/performance analysis of TTv2 with optimized architecture for a broad class of neural network models requires additional studies.

PART II: Model Extraction

Machine learning experts try various neural network architectures and hyper-parameter settings to obtain the best performing model during model development. Therefore, accelerating the DNN training process is extremely important. However, once the desired model is obtained, it is equally important to deploy the model in the field successfully. Even though training may use one set of hardware, numerous users likely run the deployed model on several hardware architectures, separate from the one the machine learning experts trained the model with. Therefore, to close the development and deployment lifecycle, the desired model must be extracted from the analog hardware for its deployment on another hardware.

In contrast to digital solutions, the weights of the model are not directly accessible on analog hardware. Analog arrays encode the model's weights, and the tile's noisy mat-vec limits access to these weight matrices. Therefore, the extraction of the model from analog hardware is a non-trivial task. Furthermore, the model extraction must produce a good representation of the trained model to be deployed without loss of accuracy on another analog or a completely different digital hardware for inference workloads.

In Part II, we first provide how an accurate weight extraction can be performed from noisy analog hardware. Then we further generalize this method to obtain an accurate model average over the TTv2 iterates. Ref Izmailov et al. (2019a) showed that the Stochastic Weight Averaging (SWA) procedure that performs a simple averaging of multiple points along the trajectory of SGD leads to better generalization than conventional training. Our analog-hardware-friendly SWA on TTv2 iterates shows that these techniques inspired by the Bayesian treatment of neural networks can also be applied to analog training hardware successfully. We show that the model averaging further boosts the extracted model's generalization performance and provides a model that is even better than the trained model itself, enabling the deployment of the extracted model virtually on any other hardware.

Accurate Weight Extraction

Analog tiles perform mat-vec on the stored matrices. Therefore, naively one can perform a series of mat-vecs using one-hot encoded inputs to extract the stored values one column (or one row) at a time. However, this scheme results in a very crude estimation of the weights due to the mat-vec noise and limited ADC resolution. Instead, we perform a series of mat-vecs using random inputs and then use the conventional linear regression formula, **Eq. 4**, to estimate the weights.

$$\widehat{C} = \left(\left(XX^T \right)^{-1} XY^T \right)^T \tag{4}$$

In **Eq. 4**, $\widehat{C}$ is an estimate of the ground truth matrix C stored on the tile, X has the inputs used during weight extraction, and Y has the resulting outputs read from the tile. Both X and Y are written in matrix form, capturing all the mat-vecs performed on the tile.

Figure 6 shows the quality of different weight estimations for a simulated tile of size 512 × 512 with the same analog array assumptions described in Part I. When one-hot encoded input vectors are used only once, corresponding to 512 mat-vecs, the correlation of the extracted values to the ground truth is very poor due to analog mat-vec noise (σ_{MV}) and ADC quantization, as seen in **Figure 6A**. Repeating the same measurements 20 times, corresponding to a total of 10,240 mat-vecs, improves the quality of the estimate (**Figure 6B**). However, the best estimate is obtained when completely random inputs with uniform distribution are used, as illustrated in **Figure 6C**. We note that the total number of mat-vecs is the same for **Figures 6B,C**, and yet **Figure 6C** provides a much better estimate. This is because the completely random inputs have the highest entropy (information content), and therefore they provide the best estimate of the ground truth for the same number of mat-vecs.

Note, in this linear regression formalism, the tile noise and quantization error correspond to aleatoric uncertainty and cannot be improved. However, the weight estimates are not limited by the aleatoric uncertainty; and instead, the epistemic uncertainty limits these estimates. For the data shown in **Figure 6C**, the standard deviation in weight estimation (corresponding to the epistemic uncertainty) is 0.002, only 0.1% of the nominal weight range of [−1, 1] used for these experiments. The uncertainty in weight estimates scales with $1/\sqrt{\text{number\_of\_mat\_vecs}}$, and if needed, this uncertainty can be further reduced by performing more measurements.

Accurate Model Average

As shown in Ref Izmailov et al. (2019a), SWA performs a simple averaging of multiple points along the trajectory of SGD and leads to better generalization than conventional training. This SWA procedure approximates the Fast Geometric Ensemble (FGE) approach with a single model. Furthermore, Ref Yang et al. (2019) showed that SWA brings benefits to low precision training. Here, we propose that weight averaging over TTv2 iterates would also bring similar gains and possibly overcome noisy updates unique to the RPU devices. However, obtaining the weight averages from analog hardware may become prohibitively expensive. Naïvely, the weights can be first extracted from analog hardware after each iteration and then accumulated in the digital domain to compute averages. However, this requires thousands of mat-vecs per iteration and therefore is not feasible.

Instead, to estimate the weight averages, we perform a series of mat-vecs that are very sparse in time but performed while the training progresses and then use the same linear regression formula to extract the weights. Since the mat-vecs are performed while weights are still evolving, the extracted values

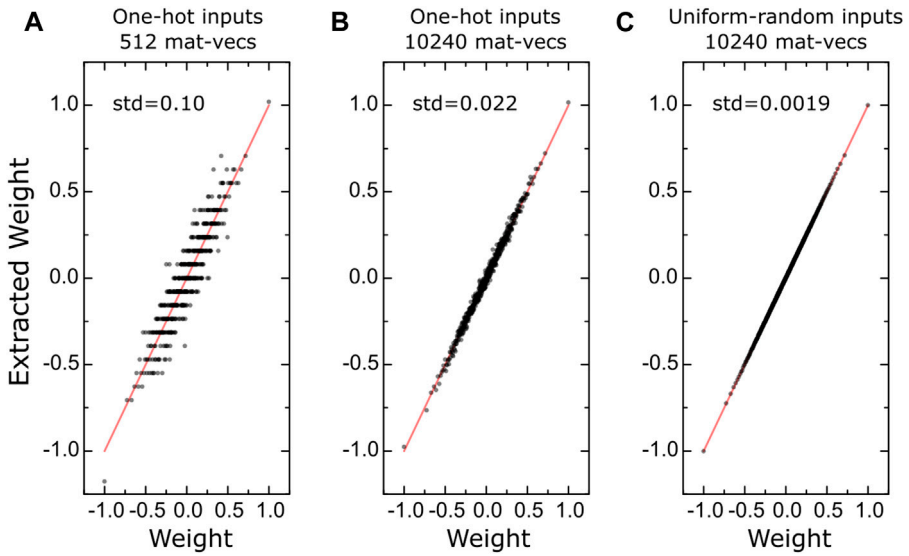


FIGURE 6 | (**A–C**) Correlation between the ground truth weights and the extracted values using different input forms and number of mat-vecs for a simulated 512 × 512 tile. Red lines are guides to the eye showing perfect correlation.

TABLE 1 | Inference results of various models on different hardware.

	Model–I, I_x, I_{avg} #States = 15, σ_{cycle} = 1	**Model–II, II_x, II_{avg} #States = 60, σ_{cycle} = 0.3**	**Model–III,III_x, III_{avg} #States = 120, σ_{cycle} = 0.3**
Inference on Analog Hardware			
50th Epoch-Trained Model	**1.633**	**1.454**	**1.430**
50th Epoch-Extracted Model	1.633	1.455	1.430
40th–50th Epochs-Extracted Model Avg	1.560	1.425	1.407
200th Epoch-Trained Model	**1.570**	**1.410**	**1.403**
200th Epoch-Extracted Model	1.571	1.410	1.403
180th–200th Epochs-Extracted Model Avg	1.487	1.377	1.372
Inference on Digital Hardware			
50th Epoch-Extracted Model	1.583	1.403	1.378
40th–50th Epochs-Extracted Model Avg	1.520	1.379	1.359
200th Epoch-Extracted Model	1.524	1.360	1.350
180th–200th Epochs-Extracted Model Avg	1.454	*1.334*	*1.326*

FP Baseline Model: 1.315–1.332.

Repeating the same FP training results in about 0.01 variability in the test error due to the randomness in weight initialization. Bold values provide the baseline training results without model extraction. Italic values correspond to models that are indistinguishable from the FP model.

closely approximate the weight averages for that training period. For instance, during the last 10 epochs of the TTv2 iterates, we performed 100 K mat-vecs with uniform-random inputs and showed that it is sufficient to estimate the actual weight averages with less than 0.1% uncertainty.

We note that about 60 M mat-vecs on C and 30 M updates on A are performed during 10 epochs of training. Therefore, the additional 100 K mat-vecs on C needed for weight averaging increases the compute on the analog tiles by only 0.1%. Furthermore, the input and output vectors (x, y) for each mat-vec can be processed on the fly by accumulating the results of xx^T and xy^T on two separate matrices in the digital domain: $M_{xx} \leftarrow M_{xx} + xx^T$ and $M_{xy} \leftarrow M_{xy} + xy^T$. Then at the end of the training, one matrix inversion and a final matrix-matrix multiply need to be performed to complete all the steps needed to estimate the weight averages: $\widehat{C}_{avg} = ((M_{xx})^{-1}M_{xy})^T$.

In practical applications, a separate conventional digital processor (like CPU) can perform the computations needed for weight averages by only receiving the results of the mat-vecs from the analog accelerator. Note that the CPU can generate the same input vectors by using the same random seed. Therefore, M_{xx} and its inverse can be computed and stored well ahead of time, even before training starts. Furthermore, the same input vectors and a common $(M_{xx})^{-1}$ can extract the weight averages from multiple analog tiles. After all these optimizations, even a conventional digital processor can sustain the computation needed for M_{xy} from multiple tiles and provide the weight averages at the end of training.

Inference Results

To test the validity of the proposed weight extraction and averaging techniques, we study the same model trained on extremely noisy analog hardware using TTv2 with the hysteretic threshold. We refer to this model as Model-I. As shown in **Figure 5E**, the test error of Model-I at the end of the 50th and 200th epochs are 1.633 and 1.570, respectively. These test errors assume Model-I runs inferences on the same analog hardware it trained on and form our baseline.

We apply our model extraction technique in the first experiment and obtain the weights using only 10 K mat-vecs with random inputs. We refer to this extracted model as Model-I_x, and it is an estimate of Model-I. We evaluate the test error of Model-I_x when it runs either on another analog hardware (with the same analog array properties) or digital hardware. As summarized in **Table 1**, Model-I_x's test error remains unchanged on the new analog hardware compared to Model-I, showing our model extraction technique's success. Interestingly, the inference results of Model-I_x are better on the digital hardware, and the test errors drop to 1.583 and 1.524 respective for the 50th and 200th epochs. These improvements are due to the absence of the mat-vec noise introduced by the forward propagation on analog hardware. However, these results also highlight that the analog training yields a better model than the test error on the same analog hardware indicates. Therefore, such benefits ease analog hardware's adoption for training only purposes, and the improved test results on digital hardware are the relevant metrics for such a use case.

We implement our model averaging technique using 100 K mat-vecs with random inputs applied between 40–50 or 180–200 epochs in the following experiment. We refer to the extracted model average as Model-I_{avg}, and the test error for Model-I_{avg} is also evaluated on analog or digital hardware. In all cases, as illustrated in **Table 1**, Model-I_{avg} gives non-trivial improvements compared to Model-I_x (and Model-I). These improvements on the averaged models' generalization performance show the success of our model averaging technique. We emphasize that the model training is performed on extremely noisy analog hardware using TTv2. Nevertheless, the test error achieved by Model-I_{avg} on digital hardware is 1.454, just shy of the FP model's performance at about 1.325.

Finally, to further illustrate the success of the proposed model extraction and averaging techniques, we performed simulations for another two models, Model II and III, which are also

TABLE 2 | Inference results of pruned networks for Model-III on digital hardware.

Signal-to-noise used for pruning	Weight proportion removed (%)	Carefully pruned network	Randomly pruned network
(No pruning)	0	1.326	1.326
$\lvert\mu_i\rvert/\sigma_i < 1$	16.7	1.331	3.42 ± (0.26)
$\lvert\mu_i\rvert/\sigma_i < 2$	30.2	1.371	4.09 ± (0.32)
$\lvert\mu_i\rvert/\sigma_i < 3$	40.8	1.466	4.40 ± (0.29)

Random pruning experiments are performed 10 times. The table reports the mean and standard deviation of these 10 experiments for the randomly pruned networks. An untrained network gives ~4.46 test error corresponding to a random guess.

TABLE 3 | Inference results of disturbed networks for Model-III on analog hardware.

Signal-to-noise used for disturbance	Carefully disturbed network	Randomly disturbed network
(No disturbance)	1.370	1.370
$\mu_i \pm \sigma_i$	1.493	3.54 ± (0.15)

Random disturb experiments are performed 10 times. The table reports the mean and standard deviation of these 10 experiments.

summarized in **Table 1**. Like Model-I, these models are also trained on noisy analog hardware but with slightly relaxed array assumptions. The only two differences compared to Model-I are 1) Model-II and III both used analog arrays with the additive cycle-to-cycle update noise at $\sigma_{cycle} = 0.3$, 2) Model-II and III respectively had 60 and 120 states on RPU devices. For these slightly relaxed but still significantly noisy analog hardware settings, both Model-II and III provide test results on the digital hardware that are virtually indistinguishable from the FP model when the model averages between 180–200 epochs are used.

We note that the inference simulations performed on analog hardware did not include any weight programming errors that may otherwise exist in real hardware. Depending on its strength, these weight programming errors cause an accuracy drop on the analog hardware used solely for inference purposes. Additionally, after the initial programming, the accuracy may further decline over time due to device instability, such as the conductance drift (Mackin et al., 2020; Joshi et al., 2020). Therefore, any analog hardware targeting inference workloads must address these non-idealities. However, we emphasize that these problems are unique to inference workloads. Instead, if analog hardware is targeting training workloads only, these problems become obsolete. Furthermore, the unique challenges of the analog training hardware, namely the limited number of states on RPU devices and the update noise, are successfully handled by our proposed TTv2 training algorithm and the model averaging technique. As illustrated above, even very noisy analog hardware can deliver models on par in their accuracy compared to FP models. In addition, after the training process is performed on analog hardware using TTv2, the extracted model average can be deployed on various digital hardware and perform inference without any accuracy loss. Therefore, these results provide a clear path for analog hardware to be employed to accelerate DNN training workloads.

DISCUSSION AND FUTURE DIRECTIONS

DNN training using SGD is simply an optimization algorithm that provides a point estimate of the DNN parameters at the end of the training. In this frequentist view, a hypothesis is tested without assigning any probability distribution to the DNN parameters and lacks the representation of uncertainty. More recently, however, the Bayesian treatment of DNNs has gained more traction with new approximate Bayesian approaches (Wilson, 2020). Bayesian approaches treat the DNN parameters as random variables with probabilities. We believe many exciting directions for future research may connect these approximate Bayesian approaches and neural networks running on noisy analog hardware.

For instance, Ref Maddox et al. (2019) showed that a simple baseline for Bayesian uncertainty could be formed by determining the weight uncertainties from the SGD iterates, referred to as SWA-Gaussian. It is empirically shown that SWA-Gaussian approximates the shape of the true posterior distribution of the weights, described by the stationary distribution of SGD iterates. We can intuitively generalize these results to the TTv2 algorithm running on analog hardware. For instance, the proposed TTv2 algorithm updates a tiny fraction of the neural network weights when enough evidence is accumulated by A and H's gradient processing steps. Nevertheless, the updates on weights are still noisy due to stochasticity in analog hardware. Therefore, TTv2 iterates resemble the Gibbs sampling algorithm used to approximate a posterior multivariate probability distribution governed by the loss surface of the DNN. Assuming this intuition is correct, analyzing the uncertainty in weights over TTv2 iterates may provide a simple Bayesian treatment of a DNN, similar to SWA-Gaussian.

To test the feasibility of the above arguments, we performed the following experiments that are motivated by the results of SWA-Gaussian (Maddox et al., 2019) and Bayes-by-Backprop (Blundell et al., 2015): First, we extract the mean (μ_i) and the standard deviation (σ_i) of each weight from the TTv2 iterates

and define a signal-to-noise ratio as $|\mu_i|/\sigma_i$. Then we remove the weights with the lowest signal-to-noise ratio below a certain value and compare the inference performance of this *carefully* pruned network to the unpruned one. We also look at the performance degradation of a randomly pruned network with the same amount of weight pruning. **Table 2** summarizes the results of these experiments performed for Model-III from 180 to 200 epochs.

As illustrated in **Table 2**, the *carefully* pruned network's performance (1.331) is almost identical to the unpruned one (1.326) when $|\mu_i|/\sigma_i < 1$, corresponding to 16.7% pruning. However, the same amount of pruning causes significant performance degradation for a randomly pruned network (~3.42). When the signal-to-noise threshold is raised to 3, corresponding to 40.8% pruning, the *carefully* pruned network still performs reasonably well (1.466). Whereas at this level of pruning, a randomly pruned network is not any better than an untrained network producing random predictions.

In the second set of experiments, as summarized in **Table 3**, we use the extracted means (μ_i) and standard deviations (σ_i) and disturb each weight randomly proportional to its standard deviation: $w_i = \mu_i + \xi\sigma_i$, where ξ is sampled from a unit Gaussian for each weight. Then, we compare the inference performance of this *carefully* disturbed network to a randomly disturbed network with the same amount of total weight disturbance. Although the *carefully* disturbed network performs reasonably well at 1.493, the randomly disturbed networks' performance significantly degrades to about 3.54.

These experiments empirically suggest that the weight uncertainty of TTv2 iterates on analog hardware provides additional valuable information about the posterior probability distribution of the weights governed by the loss surface of the DNN. The results illustrated in **Tables 2**, **3** do not address how the weight uncertainty can be extracted from analog hardware in practical settings; however, suppose this information can be extracted. In that case, the weight uncertainty can be used to sparsify the DNN during the model deployment on digital hardware (Blundell et al., 2015). Alternatively, the weight uncertainties can be leveraged to devise better programming routines while transferring the model to another noisy analog hardware. In addition, a low dimensional subspace can be constructed over TTv2 iterates so that the model can be deployed as a Bayesian neural network, similar to the results presented in Ref Izmailov et al. (2019b). The Bayesian model averaging performed even in low dimensional subspaces produces accurate predictions and well-calibrated predictive uncertainty (Izmailov et al., 2019b). We believe that noisy analog hardware with modified learning algorithms can also accelerate Bayesian approaches while simultaneously providing many known benefits, such as improved generalization and uncertainty calibration. However, these ideas require further investigation, and new techniques that can also extract the weight uncertainty from analog hardware are needed. Furthermore, extending this work to larger and more extensive networks is a general task for the feasibility of analog crossbar arrays, not only restricted to the work presented here.

SUMMARY

In summary, we presented a new DNN training algorithm, TTv2, that provides successful training on extremely noise analog hardware composed of resistive crossbar arrays. Compared to previous solutions, TTv2 addresses all sorts of hardware non-idealities coming from resistive devices and peripheral circuits and provides orders of magnitude relaxation to many hardware specs. Device arrays with non-symmetric and noisy conductance modulation characteristics and a limited number of states are enough for TTv2 to train neural networks close to their ideal accuracy. In addition, the model averaging technique applied over TTv2 iterates provides further enhancements during the model extraction. In short, we describe an end-to-end training algorithm and model extraction technique from extremely noisy crossbar-based analog hardware that matches the performance of full-precision SGD training. Our techniques can be immediately realized and applied to many readily available device technologies that can be utilized for analog deep learning accelerators.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

TG conceived the original idea, developed the methodology, wrote the simulation code, analyzed and interpreted the results, and drafted the manuscript.

ACKNOWLEDGMENTS

Author thanks to Wilfried Haensch for illuminating discussions and Paul Solomon for careful reading of the manuscript.

REFERENCES

Agarwal, S., Gedrim, R. B. J., Hsia, A. H., Hughart, D. R., Fuller, E. J., Talin, A. A., James, C. D., Plimpton, S. J., and Marinella, M. J. (2017). "Achieving Ideal Accuracies in Analog Neuromorphic Computing Using Periodic Carry," in Symposium on VLSI Technology, Kyoto, Japan. doi:10.23919/vlsit.2017.7998164

Agarwal, S., Plimpton, S. J., Hughart, D. R., Hsia, A. H., Richter, I., Cox, J. A., et al. (2016). *Resistive Memory Device Requirements for a Neural Network Accelerator*. Vancouver, BC, Canada: IJCNN. doi:10.1109/IJCNN.2016.7727298

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., di Nolfo, C., et al. (2018). Equivalent-accuracy Accelerated Neural-Network Training Using Analogue Memory. *Nature* 558, 60–67. doi:10.1038/s41586-018-0180-5

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). "Weight Uncertainty in Neural Networks," in Proceedings of the 32nd International Conference on International Conference on Machine Learning, PMLR 37, 1613–1622.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). "Language Models Are Few-Shot Learners," arXiv:2005.14165 [cs.CL].

Burr, G. W., Narayanan, P., Shelby, R. M., Sidler, S., Boybat, I., di Nolfo, C., and Leblebici, Y. (2015). "Large-scale Neural Networks Implemented with Non-volatile Memory as the Synaptic Weight Element: Comparative Performance Analysis (Accuracy, Speed, and Power)," in IEDM (International Electron Devices Meeting). doi:10.1109/iedm.2015.7409625

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic Computing Using Non-volatile Memory. *Adv. Phys. X* 2, 89–124. doi:10.1080/23746149.2016.1259585

Cloud Tpu. (2007). Available: https://cloud.google.com/tpu/docs/bfloat16.

Fuller, E. J., Keene, S. T., Melianas, A., Wang, Z., Agarwal, S., Li, Y., et al. (2019). Parallel Programming of an Ionic Floating-Gate Memory Array for Scalable Neuromorphic Computing. *Science* 364 (6440), 570–574. doi:10.1126/science.aaw5581

Gokmen, T., and Haensch, W. (2020). Algorithm for Training Neural Networks on Resistive Device Arrays. *Front. Neurosci.* 14, 103. doi:10.3389/fnins.2020.00103

Gokmen, T., Onen, M., and Haensch, W. (2017). Training Deep Convolutional Neural Networks with Resistive Cross-Point Devices. *Front. Neurosci.* 11, 538. doi:10.3389/fnins.2017.00538

Gokmen, T., Rasch, M. J., and Haensch, W. (2018). Training LSTM Networks with Resistive Cross-Point Devices. *Front. Neurosci.* 12, 745. doi:10.3389/fnins.2018.00745

Gokmen, T., and Vlasov, Y. (2016). Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations. *Front. Neurosci.* 10, 333. doi:10.3389/fnins.2016.00333

Graphcore. (2021). Available: https://www.graphcore.ai/.

Grollier, J., Querlioz, D., Camsari, K. Y., Everschor-Sitte, K., Fukami, S., and Stiles, M. D. (2020). Neuromorphic Spintronics. *Nat. Electron.* 3, 360–370. doi:10.1038/s41928-019-0360-9

Yang, G., Zhang, T., Kirichenko, P., Bai, J., Wilson, A. G., and Sa, C. D. (2019). "SWALP: Stochastic Weight Averaging in Low-Precision Training," arXiv:1904.11943 [cs.LG].

Haensch, W., Gokmen, T., and Puri, R. (2019). The Next Generation of Deep Learning Hardware: Analog Computing. *Proc. IEEE*, 107, 108–122. doi:10.1109/jproc.2018.2871057

He, K., Zhang, X., Ren, S., and Sun, J. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in IEEE International Conference on Computer Vision (ICCV). doi:10.1109/iccv.2015.123

Kim, H., Rasch, M., Gokmen, T., Ando, T., Miyazoe, H., Kim, J.-J., et al. (2019). "Zero-shifting Technique for Deep Neural Network Training on Resistive Cross-point Arrays," arXiv:1907.10228 [cs.ET].

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2019). "Averaging Weights Leads to Wider Optima and Better Generalization," arXiv: 1803.05407 [cs.LG].

Izmailov, P., Maddox, W., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. (2019b). "Subspace Inference for Bayesian Deep Learning," in *Uncertainty in Artificial Intelligence* (UAI) 115, 1169–1179.

Joshi, V., Le Gallo, M., Haefeli, S., Boybat, I., Nandakumar, S. R., Piveteau, C., et al. (2020). Accurate Deep Neural Network Inference Using Computational Phase-Change Memory. *Nat. Commun.* 11, 2473. doi:10.1038/s41467-020-16108-9

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep Learning. *Nature* 521, 436–444. doi:10.1038/nature14539

Mackin, C., Narayanan, P., Ambrogio, S., Tsai, H., Spoon, K., Fasoli, A., Chen, A., Friz, A., Shelby, R. M., and Burr, G. W. (2020). "Neuromorphic Computing with Phase Change, Device Reliability, and Variability Challenges," in IEEE International Reliability Physics Symposium, Dallas, TX, USA (IRPS). doi:10.1109/irps45951.2020.9128315

Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. (2019). "A Simple Baseline for Bayesian Uncertainty in Deep Learning," in Advances in Neural Information Processing Systems (Vancouver, BC, Canada: NeurIPS), 32, 13153–13164.

Miyashita, D., Lee, E. H., and Murmann, B. (2016). "Convolutional Neural Networks Using Logarithmic Data Representation," arXiv:1603.01025 [cs.NE].

Nandakumar, S. R., Le Gallo, M., Piveteau, C., Joshi, V., Mariani, G., Boybat, I., et al. (2020). Mixed-Precision Deep Learning Based on Computational Memory. *Front. Neurosci.* 14, 406. doi:10.3389/fnins.2020.00406

Nvidia. (2021). Available: https://www.nvidia.com/en-us/data-center/a100/.

Onen, M., Gokmen, T., Todorov, T. K., Nowicki, T., Alamo, J. A. D., Rozen, J., et al. (2021). *Neural Network Training with Asymmetric Crosspoint Elements*. submitted for publication.

Rasch, M. J., Gokmen, T., and Haensch, W. (2020). Training Large-Scale Artificial Neural Networks on Simulated Resistive Crossbar Arrays. *IEEE Des. Test.* 37 (2), 19–29. doi:10.1109/mdat.2019.2952341

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature* 323, 533–536. doi:10.1038/323533a0

Steinbuch, K. (1961). Die Lernmatrix. *Kybernetik* 1, 36–45.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP," *ACL 2019 - 57th. Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.*, 3645–3650.

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., et al. (2019). Hybrid 8-bit Floating point (HFP8) Training and Inference for Deep Neural Networks. *Adv. Neural Inf. Process. Syst.* 32, 4901–4910.

Sun, X., Wang, N., Chen, C.-Y., Ni, J.-M., Agrawal, A., Cui, X., et al. (2020). Ultra-Low Precision 4-bit Training of Deep Neural Networks. *Adv. Neural Inf. Process. Syst.* 33, 1796–1807.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE*, 105, 2295–2329. doi:10.1109/jproc.2017.2761740

Wilson, A. G. (2020). *Bayesian Deep Learning and a Probabilistic Perspective of Model Construction*. International Conference on Machine Learning Tutorial.

Woo, J., and Yu, S. (2018). Resistive Memory-Based Analog Synapse: The Pursuit for Linear and Symmetric Weight Update. *IEEE Nanotechnology Mag.* 12, 36–44. doi:10.1109/mnano.2018.2844902

Yu, S., Chen, P., Cao, Y., Xia, L., Wang, Y., and Wu, H. (2015). "Scaling-up Resistive Synaptic Arrays for Neuro-Inspired Architecture: Challenges and prospect," in International Electron Devices Meeting (IEDM), Washington, DC, USA (IEEE). doi:10.1109/iedm.2015.7409718

Yu, S. (2018). Neuro-inspired Computing with Emerging Nonvolatile Memorys. *Proc. IEEE*, 106, 260–285. doi:10.1109/jproc.2018.2790840

Conflict of Interest: TG was employed by the company IBM. The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 22 November 2021
doi: 10.3389/fnins.2021.749811

Gradient Decomposition Methods for Training Neural Networks With Non-ideal Synaptic Devices

Junyun Zhao[1†], Siyuan Huang[1†], Osama Yousuf[2], Yutong Gao[1], Brian D. Hoskins[3] and Gina C. Adam[2]*

[1] Department of Computer Science, George Washington University, Washington, DC, United States, [2] Department of Electrical and Computer Engineering, George Washington University, Washington, DC, United States, [3] Physical Measurement Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, United States

While promising for high-capacity machine learning accelerators, memristor devices have non-idealities that prevent software-equivalent accuracies when used for online training. This work uses a combination of Mini-Batch Gradient Descent (MBGD) to average gradients, stochastic rounding to avoid vanishing weight updates, and decomposition methods to keep the memory overhead low during mini-batch training. Since the weight update has to be transferred to the memristor matrices efficiently, we also investigate the impact of reconstructing the gradient matrixes both internally (*rank-seq*) and externally (*rank-sum*) to the memristor array. Our results show that streaming batch principal component analysis (streaming batch PCA) and non-negative matrix factorization (NMF) decomposition algorithms can achieve near MBGD accuracy in a memristor-based multi-layer perceptron trained on the MNIST (Modified National Institute of Standards and Technology) database with only 3 to 10 ranks at significant memory savings. Moreover, NMF *rank-seq* outperforms streaming batch PCA *rank-seq* at low-ranks making it more suitable for hardware implementation in future memristor-based accelerators.

Keywords: non-negative matrix factorization, gradient data decomposition, principal component analysis, memristor, non-idealities, ReRAM

OPEN ACCESS

Edited by:
Alexantrou Serb, University of Southampton, United Kingdom

Reviewed by:
Wei Wang, Technion – Israel Institute of Technology, Israel
Seyoung Kim, Pohang University of Science and Technology, South Korea

****Correspondence:***
Gina C. Adam
ginaadam@gwu.edu

† These authors have contributed equally to this work and share first authorship

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *30 July 2021*
Accepted: *20 October 2021*
Published: *22 November 2021*

Citation:
Zhao J, Huang S, Yousuf O, Gao Y, Hoskins BD and Adam GC (2021) Gradient Decomposition Methods for Training Neural Networks With Non-ideal Synaptic Devices. Front. Neurosci. 15:749811. doi: 10.3389/fnins.2021.749811

INTRODUCTION

As artificial intelligence (AI) applications become ubiquitous in medical care, autonomous driving, robotics, and other fields, accuracy requirements and neural network complexity increase in tandem, requiring extensive hardware support for training. For example, GPT-3 is made up of ≈175 billion parameters and requires 285,000 central processing unit (CPU) cores and 10,000 graphics processing units (GPUs) to be trained on tens of billions of web pages and book texts (Langston, 2020). Moreover, the use of such significant computing resources has major financial and environmental impacts (Nugent and Molter, 2014; Strubell et al., 2020). New neuroinspired hardware alternatives are necessary for keeping up with increasing demands on complexity and energy efficiency.

Emerging non-volatile memory (NVM) technologies, such as oxygen vacancy-driven resistive switches, also known as ReRAM or memristors (Chang et al., 2011; Wong et al., 2012; Chen, 2020), can combine data processing and storage. Memristor matrices (crossbar arrays) use physical

principles to enable efficient parallel multiply-accumulate (MAC) operations (Hu et al., 2018). This in-memory computing paradigm can achieve a substantial increase in speed and energy efficiency (Ceze et al., 2016) without the bottleneck caused by traditional complementary metal-oxide-semiconductor (CMOS) transistor-based von Neumann architectures. However, due to the inherent operational stochasticity of memristors in addition to manufacturing yield and reproducibility challenges, this emerging technology suffers from non-idealities. Thus, the accuracy of a neural network implemented with non-ideal memristor synaptic weights is not software-equivalent. To alleviate the undesirable effects of these devices, it is necessary to engineer better devices and improve the existing training algorithms.

This work investigates the use of Mini-Batch Gradient Descent (MBGD) for high accuracy training of neural networks with non-ideal memristor-based weights together with the use of gradient decomposition methods to alleviate the memory overhead due to the storage of gradient information between batch updates. An initial investigation (Gao et al., 2020) showed that the MBGD of moderate batch sizes (e.g., 128) can overcome the low accuracy of SGD for a one-hidden-layer perceptron network implemented with non-ideal synaptic weights trained on MNIST dataset. Accuracies of up to 86.5% were obtained for the batch sizes of 128 compared with only 50.9% for SGD. Although these results are promising, they are still far from the software equivalency of 96.5% at our studied network size.

Moreover, MBGD is memory intensive—particularly at higher batch sizes—since the gradient information needs to be stored before the batch update. We propose using a hardware co-processor to compress MBGD gradient data and work in tandem with the resistive array to support efficient array-level updates (**Figure 1A**). The first step toward this goal and the key question addressed by this paper is what decomposition algorithm should be mapped to a hardware co-processor to best support the training, particularly in neural networks implemented with non-ideal devices. Different common low-rank decomposition methods are available and have been extensively used in computer science literature to pre-process the dataset, remove noise and reduce the number of the network parameters (Garipov et al., 2016; Schein et al., 2016). Our prior work (Huang et al., 2020a,b) proposed streaming batch Principal Component Analysis (PCA) and showed that an accurate gradient matrix can be recomposed with as few as 3 to 10 ranks depending on the dataset complexity. Tests on CIFAR-10, CIFAR-100, and ImageNet showed near equivalent accuracy to MBGD at significant memory savings. However, in that work, non-ideal neural networks were not investigated.

In this study, we investigate the device-algorithm interaction which highlights the importance of hyperparameter optimization and stochastic rounding for overcoming the low-bit precision coding of the memristor weights. We propose an expansion of MBGD for larger batch sizes in conjunction with two gradient decomposition methods - Streaming Batch PCA and non-negative matrix factorization (NMF) - and recomposition methods based on rank summation (rank-sum) vs. rank-by-rank update (rank-seq) applied to a network with realistic memristor hardware models. For a $m \times n$ gradient matrix with batch size B, the MBGD cost is approximated at 2Bmn. By comparison, Streaming Batch PCA and NMF have asymptotic complexities of $k(m+n)$ and $k^2(m+n)^2$, respectively (see **Figure 1B**). The issue of gradient recomposition in order to support weight updating is also investigated, considering that rank-sum would require additional overhead on the training co-processor, while for rank-seq it is possible to envision a series of rank-1 array level updates that support recomposition on the array itself. However, it is important to point out that these decomposition algorithms have high complexity requiring QR decompositions or iterative calculations when implemented at the algorithmic level and executed on a CPU. Dedicated hardware decomposers can be envisioned that support streaming operation on data flows.

The remainder of the paper is organized as follows. Section 2 has background information related to memristors and their applicability to neural networks, as well as an overview of decomposition algorithms. Section 3 describes the methodological details, the simulation environment, and the algorithms used. Section 4 introduces the evaluation of the proposed methodology on MNIST and its comparison with SGD and MBGD. Section 5 concludes with a discussion of the results.

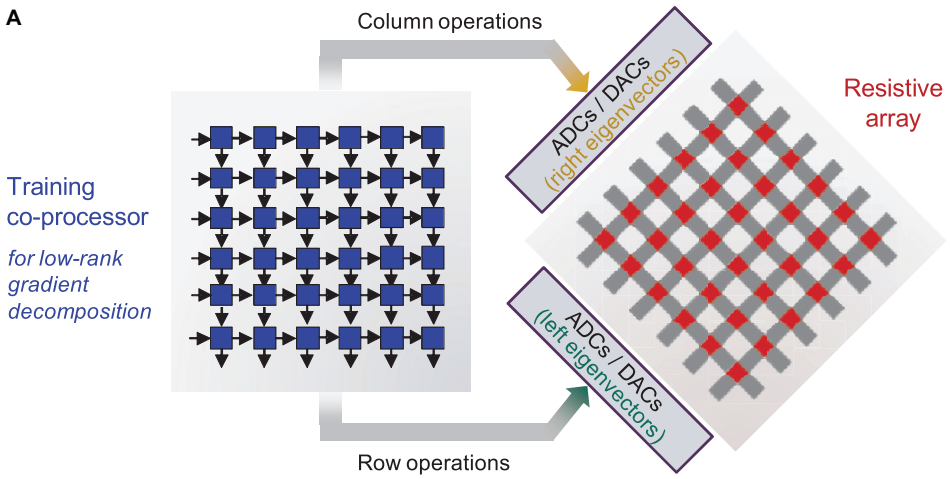
RELATED WORK

Resistive Switching Phenomena and Memristor Technology

The resistive switching phenomena was discovered in aluminum oxide in the early 1960s (Hickmott, 1962) and in other materials in the following decades (Argall, 1968; Dearnaley et al., 1970; Oxley, 1977; Pagnia and Sotnik, 1988). Due to the focus on silicon integrated circuits of the time, the technological potential of this phenomenon was not explored until the early 2000s, sparked by industry's interest in the one transistor and one memristor (1T1R) cell for digital memories (Baek et al., 2004; Seo et al., 2004; Rohde et al., 2005).

These devices have a simple structure: the upper and lower layers are metal electrodes, and the middle layer is a dielectric layer, typically a transition metal oxide. The device behavior is driven by complex multi-physics phenomena, and it is not yet fully understood. However, the main model is based on the formation and reshaping of conductive filaments. When a voltage pulse is applied, the electronic and ionic conduction driven by local Joule heating causes the filament to reshape, thus changing the device resistance and programming the weight. When the voltage is removed, and the local Joule heating stops, the ions in the structure "freeze" in place, thus retaining the filament shape and its associated resistance/weight state providing memory to the system.

Due to the inherent stochastic nature of the ionic movement under Joule heating, the devices exhibit non-ideal characteristics, such as programming variability from cycle to cycle and from device to device, the asymmetry between the resistance increase (turn OFF – long term depression) and resistance decrease (turn ON – long term potentiation), read noise, and limited ON/OFF ratio or accessible resistance states. Other device



B

	Decomposition asymptotics
MBGD	2Bmn
Streaming Batch PCA	~k(m+n)
NMF	~$k^2(m+n)^2$

	Recomposition	
	Location	**Complexity**
rank-sum	co-processor	2kmn
rank-seq	analog array	-

B – batch size
m x n – gradient size
k – rank
k – rank
b – block size

FIGURE 1 | Training co-processor for decomposition. **(A)** Sketch showing an integrated system – a digital training co-processor will implement the best identified algorithm in an efficient way to support neural network training on non-ideal analog arrays. **(B)** Computational complexity for Mini-Batch Gradient Descent (MBGD), streaming batch principal component analysis (PCA), and non-negative matrix factorization (NMF), as well recomposition methods – rank summation (rank-sum) vs. rank-by-rank update (rank-seq). The respective eigenvectors are color coded.

indicators, such as device yield, read noise, and retention also impact their practical applicability (Gokmen and Vlasov, 2016; Lin et al., 2019).

Memristor-Based Neural Network Training

A memristor crossbar can efficiently implement vector matrix multiplication using Ohm's law for the input voltage to synaptic weight conductance multiplication and Kirchhoff's law for the addition of the resulting currents. Together, these principles give rise to a vector dot product, which is the fundamental operation needed for fully-connected neural network layers (Prezioso et al., 2015). However, memristor non-idealities make the training process difficult (Adam et al., 2018). Therefore, the classification accuracies of *in-situ* training using non-volatile-memory hardware have generally been less than those of software-based training.

Several approaches have been used to mitigate these memristor device non-idealities. At the software level, binary neural networks (Chen et al., 2018) can use the devices as ON/OFF switches to reduce the impact of variability and conductance quantization. Alternatively, stochastic networks can exploit inherent cycle-to-cycle variability (Payvand et al., 2019; She et al., 2019). At the hardware level, more complex multi-memristor cells can be used (Boybat et al., 2018) to overcome asymmetry, limited bit precision and device variability at the expense of increased hardware overhead. Feedback circuitry can also be used to set the device to a well-defined value and mitigate the cycle-to-cycle variability of the devices (Serb et al., 2015). These solutions can be similarly applied to other types of emerging non-volatile memory technologies such as phase-change memory (Kim et al., 2019), magnetoresistive memory (Hirtzlin et al., 2019), ferroelectric-based memories (Berdan et al., 2020), among others.

A recent solution proposed by Ambrogio et al. (2018) has shown that batch analog systems can achieve equivalent training performance to that of the software but only at the costs of doubling the memory and exerting additional efforts in closed-loop training. Their proposed accelerator uses an analog short-term memory based on capacitors and transistors for fast and highly linear programming during training with only infrequent transfer to an analog long-term memory based on phase changes. The capacitive short-term memory is used to correct problems due to the imperfections in programming long-term phase change memories (Haensch et al., 2018). This approach, which combines the advantages of two device technologies, is feasible. However, it relies on duplicate short-term and long-term memories. Additionally, any imperfections of the short-term memory also need to be managed in hardware. A working prototype has not yet been demonstrated. Nevertheless, understanding how to leverage

alternative algorithms and architectures is critical since evidence suggests that certain algorithms, like batch update, are more resilient to the non-idealities of various devices (Kataeva et al., 2015; Gao et al., 2020; Gokmen and Haensch, 2020).

Matrix Decomposition Algorithms

Rather than using a duplicative short-term memory, linear algebra techniques can be used to compress gradient data and support efficient array-level updates. Principal component analysis (PCA), a commonly used decomposition method, projects high-dimensional data into low-dimensional subspaces. Through computing and analyzing the underlying eigenspectrum, the variance in the data is maximized. Streaming PCA (Oja, 1982), streaming history PCA (Burrello et al., 2019; Hoskins et al., 2019), and streaming batch PCA (Huang et al., 2020b) were all developed based on the core PCA algorithm. Streaming batch PCA can extract an approximation of a full matrix from samples of its contributed parts by combining bi-iterative stochastic power iterations (Vogels et al., 2019) with QR factorization to produce low rank approximations of stochastic rectangular matrices. This method reduces gradient storage and processing requirements brought by MBGD and is composed of a batch of randomly generated rank-1 matrices of forward propagated activations and backpropagated errors.

However, streaming batch PCA has no restriction on the sign of the data element, so negative values can appear in the matrix factorization. Even if all the values are strictly positive, such as in an image, the decomposition may include negative terms. This oscillatory behavior, while usually harmless, causes challenges when computation is done at the physical level: for instance, summation on memristor devices which are not inherently reversible in their programming behavior. By contrast, the Non-Negative Matrix Factorization (NMF) algorithm (Paatero and Tapper, 1994; Wang et al., 2015) calculates the decomposition by adding the non-negative constraints which results in additive features.

The NMF decomposition is particularly meaningful when the gradient information is mapped on a memristor matrix for physical recomposition. NMF can decrease the overlap between ranks, eliminating the oscillatory behavior during summation that exists in a standard PCA decomposition. This is crucial for devices that do not have a linear and symmetric weight update.

The streaming batch PCA algorithm and NMF decomposition algorithms will be used in the following sections to approximate the MBGD gradient and train a fully connected network to classify MNIST handwritten digits with high accuracy, despite device non-idealities.

METHOD DETAILS

Streaming Batch Principal Component Analysis

Streaming batch PCA or SBPCA (Huang et al., 2020b) is used to decompose the gradient information from MBGD. It compresses batch data in the neural network training period through rank-k outer product updates. The streaming batch PCA can expedite gradient descent training and decrease the memory cost by generating a stochastic low-rank approximation of the gradient. Gradient descent reduces the error between the predicted value of the neural network and the actual value by updating the parameters to minimize the result of the loss function,

$$\Theta_p = \Theta_p - \alpha * \nabla_\Theta l,$$

where Θ_p is the weight matrix of layer p, α is learning rate, $l(\Theta)$ is the loss function, and $\nabla_\Theta l = \frac{\partial l(\Theta_p)}{\partial \Theta_p}$ is the gradient.

To extract significant batch gradient data, average out the noise due to non-ideal memristor weights, and improve the network accuracy, a streaming low-rank approximation of $\widehat{\nabla}_\Theta^{(k,B)} l$ is obtained by the Streaming Batch PCA. The gradient is approximated for a batch of size B and the top-k most important k ranks as follows:

$$\widehat{\nabla}_\Theta^{(k,B)} l = \hat{X} \cdot \hat{\Sigma} \cdot \widehat{\triangle}^T \ ,$$

where $\hat{X} \in \mathbb{R}^{n\times k}$and $\widehat{\triangle} \in \mathbb{R}^{n\times k}$ denote the left singular matrix and right singular matrix, respectively. $\hat{\Sigma} = diag(\overrightarrow{\sigma}) \in \mathbb{R}^{k\times k}$ is a diagonal matrix, which has on its diagonal the corresponding singular values $\overrightarrow{\sigma}$ for the top k ranks. In the Streaming Batch PCA algorithm, the input $\overrightarrow{x} \in \mathbb{R}^{1\times m}$ and the error $\overrightarrow{\delta} \in \mathbb{R}^{1\times n}$ help to update $\hat{X}$ and $\widehat{\triangle}$. Based on Oja's rule and stochastic power iterations (Oja, 1992; Huang et al., 2020a), $\hat{X}$ and $\widehat{\triangle}$ are updated separately and bi-iteratively in a streaming fashion with an averaged block size $b < B$, followed by re-orthogonalization via QR factorization. Our QR factorization is defined to have non-increasing values on the diagonal of the R matrix. For updating $\hat{X}$, we use

$$\hat{X} \leftarrow QR\left[\frac{i}{i+1} \cdot \hat{X} + \frac{1}{i+1} \cdot \frac{\hat{x}^T \widehat{\delta\triangle} \hat{\Sigma}^{-1}}{b}\right],$$

where $\frac{i}{i+1}$ represents the convergence coefficient and $\hat{X}$ decays with each QR factorization, running from $i = 1$ until reaching $i = B/b$. The update of $\hat{\Sigma}$ is similar,

$$\hat{\Sigma} \leftarrow \frac{i}{i+1} \cdot \hat{\Sigma} + \frac{1}{(i+1)} \sum_{\text{rows}} \frac{(\widehat{\text{xX}}) \odot \left(\hat{\delta}\widehat{\triangle}\right)}{b},$$

where $\odot$ is the Hadamard (elementwise) matrix product.

From the standpoint of computational complexity, Streaming Batch PCA with k-ranks requires $4Bk(m+n)+\frac{B}{b}\cdot 4k(m+n)$ floating point operations (FLOPs) where $\frac{B}{b}\cdot$4k(m+n) is for the batch size (B) / block size (b) times QR factorizations. Overall, the complexity tends to scale as k(m+n), leading to an overall reduced computational load as compared to MBGD. However, the recomposition complexity scales as kmn, What this means is that recreating the approximation of the gradient is more computationally expensive than getting the most important eigenvectors making the recomposition calculation the most expensive part the algorithm.

Non-negative Matrix Factorization

The Non-Negative Matrix Factorization (NMF) (Lee and Seung, 1999) algorithm decomposes a non-negative matrix into

two non-negative left and right matrices $\hat{X} \in \mathbb{R}_+^{m\times k}$and $\widehat{\triangle} \in \mathbb{R}_+^{k\times n}$, respectively.

However, the gradient $\nabla_\Theta l$ is not non-negative. This is why in our NMF algorithm, we first start with a batch size B approximation of $\nabla_\Theta l$, and then use the rectified linear unit (ReLU) activation function to restrict the sign of gradient $\nabla_\Theta l$ by its unilateral inhibition feature, whereby ReLU(v) = $max(v, 0)$. The goal is to approximate the positive and negative parts separately with two sets of k-rank matrices such that $\widehat{\nabla} l_P = \hat{X}_P \cdot \widehat{\triangle}_P$ and $\widehat{\nabla} l_N = \hat{X}_N \cdot \widehat{\triangle}_N$. Four random matrices $\hat{X}_P, \hat{X}_N \in \mathbb{R}_+^{m\times k}$ and $\widehat{\triangle}_P, \widehat{\triangle}_N \in \mathbb{R}_+^{n\times k}$ are randomly initialized from a Gaussian distribution at the beginning of training with a standard deviation calculated from the root of the mean values of the gradient over the rank k, $\sqrt{\frac{\nabla_\Theta \bar{l}_P}{k}}$ and $\sqrt{\frac{\nabla_\Theta \bar{l}_N}{k}}$. Then, we use a modified version of the Fast HALS (Hierarchical Alternating Least Squares) (Cichocki and Phan, 2009) algorithm to alternately update the left and right matrices. To do the minimization, we assume a pair of loss functions of the form $\frac{1}{2}||\nabla_\Theta l_P^{(k)} - \hat{X}_{Pk}\widehat{\triangle}_{Pk}^T||_{\mathrm{F}}^2$, where k is the rank and F is the Frobenius norm, with one loss function for the positive matrix and a similar one for the negative matrix. This product of the left ($\hat{X}_{Pk}$) and right ($\widehat{\triangle}_{Pk}^T$) matrices best approximates the non-negative gradient when these loss functions are minimized. During the non-negative part iteration update, the quantities $\hat{X}_P^T\hat{X}_P$ and $\widehat{\triangle}_P\widehat{\triangle}_P^T$ are calculated. The diagonal matrices $D_X \leftarrow \mathrm{Diag}(\hat{X}_P^T\hat{X}_P)^{-1}$ and $D_\triangle \leftarrow \mathrm{Diag}(\widehat{\triangle}_P\widehat{\triangle}_P^T)^{-1}$ are calculated to scale the updates (Cichocki et al., 2009). Similar quantities are calculated for $\hat{X}_N$ and $\widehat{\triangle}_N$. With this basic framework in mind, we can iteratively, for as many as P cycles, update the positive (or negative) decomposition by

$$\hat{X}_P \leftarrow \mathrm{ReLU}(\hat{X}_P + (-\hat{X}_P\widehat{\triangle}_P\widehat{\triangle}_P^T + \nabla_\Theta \bar{l}_P\widehat{\triangle}_{Pk}^T)D_\triangle), \text{ and}$$

$$\widehat{\triangle}_P \leftarrow \mathrm{ReLU} + (\widehat{\triangle}_P + (-\widehat{\triangle}_P\hat{X}_P^T\hat{X}_P + \nabla_\Theta \bar{l}_P\hat{X}_{Pk})D_X)).$$

The number of iterations, P, will depend on the desired level of convergence as well as the initialization. The number of iterations can be reduced by streaming the current best estimates for $\hat{X}_P$, $\widehat{\triangle}_P$ and $\hat{X}_N$, $\widehat{\triangle}_N$ from batch to batch after the first random initialization, as we do in our case. In this work, we explored using a fixed, 200 iterations to understand the impact of NMF factorization on training. We also studied doing these operations with one iteration to see how streaming would impact training, see Section 3 and **Supplementary Figure 2**.

After convergence, the new left gradient matrix $\widehat{\nabla} l_P = \hat{X}_P \cdot \widehat{\triangle}_P$ and right matrix $\widehat{\nabla} l_N = \hat{X}_N \cdot \widehat{\triangle}_N$ would be generated. At the end, the low-rank matrix approximation is $\widehat{\nabla}_\Theta^{(k,B)} l = \widehat{\nabla} l_P - \widehat{\nabla} l_N$. It is important to understand that, while this method produces a potentially optimal and non-oscillating decomposition, it still relies on summing and reconstructing the batch gradient. This makes it much more computationally complex than the Streaming Batch PCA algorithm. However, its memory overhead could be improved and its hardware mapping will be explored in the future. For this work, we are primarily interested in the impact of the decomposition on training.

Assuming the sequential least squares minimization (e.g., HALS) is done in p iterations, the FLOPs required for NMF scales with $3mn + 2mk + 2nk + 2mnk\,(n-1) + 2p\left(k^2((m+n)^2 - m - n) + mnk\,(m+n-4) + 4k\,(m + n + \frac{1}{2})\right.$. The $\left(k^2((m+n)^2 - m - n) + mnk\,(m+n-4) + 4k(m+n+\frac{1}{2})\right)$ calculations are for the $\hat{X}_P$ and $\widehat{\triangle}_P$ or $\hat{X}_N$ and $\widehat{\triangle}_N$ updates in one iteration. As noted previously, the overall computational complexity scales as $k^2(m+n)^2$ making it at this time more computationally complex than MBGD. However, should this performance be improved, it would be very advantageous since the NMF algorithm has a better performance when training networks rank-by-rank, or using the *rank-seq* operation as discussed below.

Rank Gradient Recomposition Methods

The contrast between the oscillatory behavior of the streaming batch PCA and the additivity of the NMF decomposition methods becomes significant when considering the memristor weight updates in hardware. How these updates are performed is important for understanding the choice of algorithm on performance. One option is to do gradient summation across the ranks of interest outside the analog memory crossbar array before transfer. During training, individual samples are used to update the compressed k-rank representation of the gradient $\widehat{\nabla}_\Theta^{(k,B)} l$ based on the calculated $\hat{X}$, $\hat{\Sigma}$, and $\widehat{\triangle}$. At the end of a training batch, the gradient is recomposed and then added to the matrix in total $\widehat{\nabla}_\Theta^{(k,B)} l = \hat{X} \cdot \hat{\Sigma} \cdot \widehat{\triangle}^T$ by sequentially updating each weight one by one. We call this approach the *rank-sum* update and summarize it in **Figure 2**.

However, *rank-sum* is inefficient since (a) the data must be multiplied out and summed on the array and (b) the data must be transferred one by one into each of the individual memristor devices. The estimated computational complexity of this operation, as noted in **Figure 1**, is 2kmn. A more efficient implementation for pipelining requires the gradient summation inside the array using the update properties of the memristor devices. After producing an approximation of the gradient, the weight matrix is updated rank by rank, and the gradient is summed on the memory devices using outer product update operations. Outer product operations can be done in multiple ways, either using pulses on the rows and columns (Kataeva et al., 2015; Gokmen and Vlasov, 2016) or by relying on an exponential dependence on the applied bias on the rows and columns to multiply out the gradient (Kataeva et al., 2015). Outer product operations restrict the updates, because of the limited row/column access, to rank-1 updates. Consequently, $\widehat{\nabla}_\Theta^{(j,B)} l$ is a rank-1 matrix for the jth rank from the matrix product for the column j in $\hat{X}$, $\hat{\Sigma}$, and $\widehat{\triangle}$: $\widehat{\nabla}_\Theta^{(j,B)} l = \hat{X}_{m,j}\hat{\Sigma}_j\widehat{\triangle}_{n,j}^T$. The column number is less than or equal to rank k. Unlike the *rank-sum* method, the *rank-seq* method does not pre-sum $\widehat{\nabla}_\Theta^{(k,B)} l$ for all the ranks k. The matrix $\widehat{\nabla}_\Theta^{(j,B)} l$ is used to calculate the necessary updates for rank j to be transferred to the memristor matrix where the gradient is recomposed at the physical level. We call this method the *rank-seq* update and show its principles in **Figure 3**.

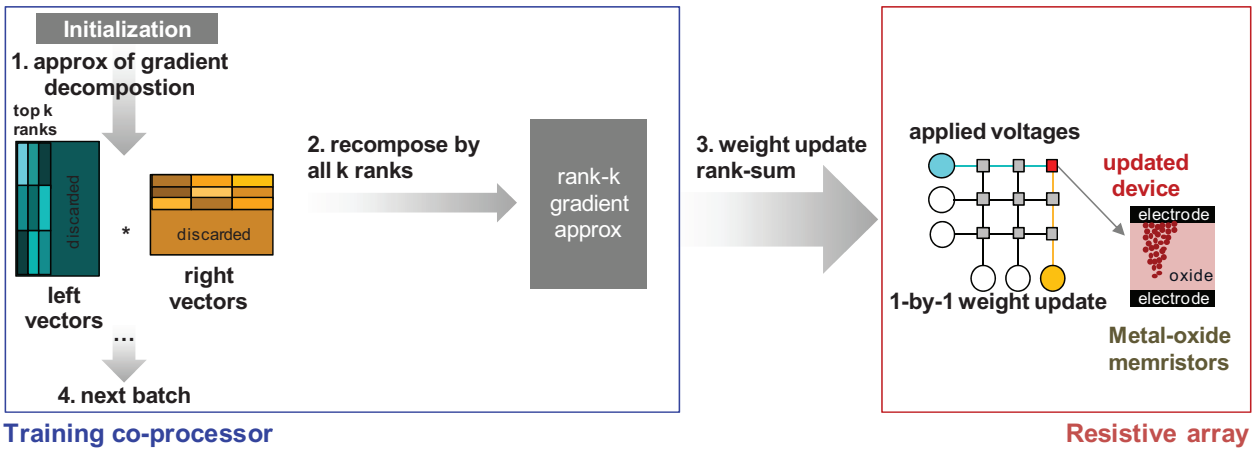


FIGURE 2 | Transfer principle of the gradient approximation information to the memristor array for the rank summation outside the array (rank-sum). The training co-processor would have to support rank-k gradient recomposition, thus increasing the hardware overhead.

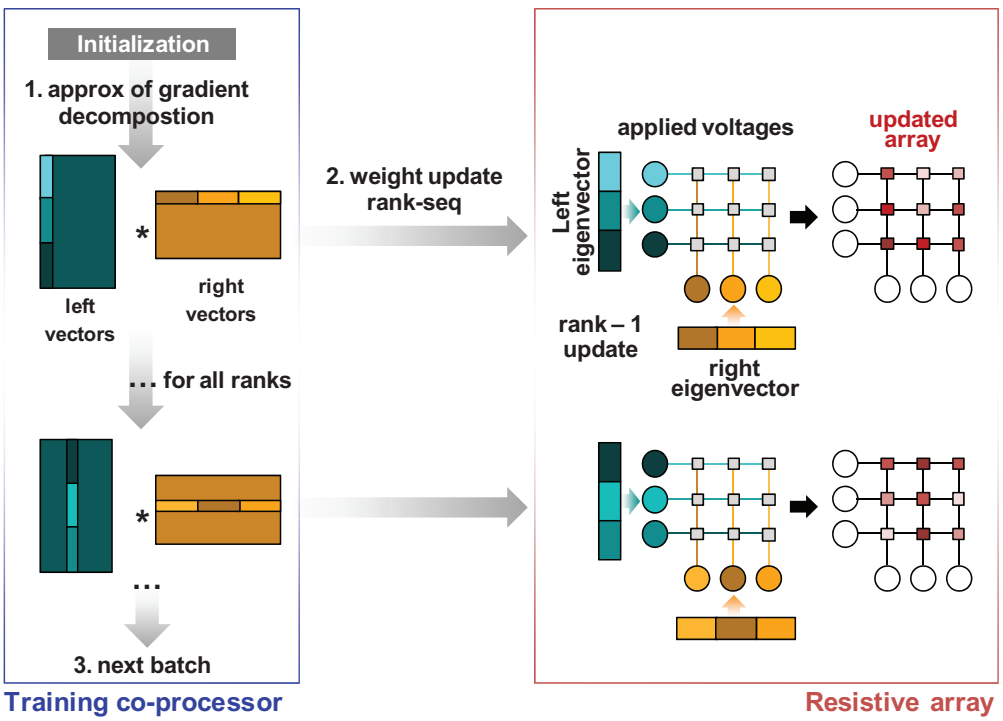


FIGURE 3 | Transfer principle of the gradient approximation information to the memristor array via rank-by-rank (rank-seq) transfer. The gradient recomposition can be done by physical summation of rank-1 updates at the array level, thus reducing the hardware overhead for the training co-processor.

It is worth pointing out that in a traditional floating-point software implementation, the two algorithms are equivalent within rounding error. However, when the gradient information needs to be transferred to a non-ideal memristor circuit, the two methods differ. *Rank-sum* updates the gradient information to the memristor crossbar only once, while the *rank-seq* needs k updates for k ranks. Updates to non-ideal memristors are accompanied by a loss in gradient precision, which is the reason that *rank-seq* to be expected to have lower accuracy than *rank-sum* for non-overlapping ranks. However, *rank-seq* is more efficient since it requires less digital computation and hardware overhead.

Stochastic Rounding

As part of the gradient transfer, the accuracy of the quantization of the weight update is also investigated in relation to the device properties. Although in theory the memristor has analog programmability to any desired state between the ON and the

OFF, the device in practice has low bit precision. The reason for low bit precision is that each state can naturally decay and can be impacted by reading disturbs or be impacted by the programming of neighboring devices (Lin et al., 2019). Therefore, the number of conductance levels reliably accessible and distinguished from each other is limited. This quantization of the weight update introduces errors due to the lower bit precision. Since the memristor conductance change is related to the number of applied pulses (an integer), the respective weight modification needs to be rounded appropriately to a lower bit precision. Rounding-to-nearest is the method commonly used (Chen et al., 2017). However, it seems to cause a premature conversion to sub-optimal accuracies at higher batch sizes due to small gradients and low bit precision causing delta weight approximation to zero.

In this work, stochastic rounding is investigated instead to overcome this quantization error vanishing gradient issue in limited precision weights. Stochastic rounding, proposed in the 1950s and 1960s (Forsythe, 1950; Barnes et al., 1951; Hull and Swenson, 1966), can be particularly useful in deep network training with low bit precision arithmetic (Gupta et al., 2015). A real value r which lies between floor value (r_1) and ceiling value (r_2) is stochastically rounded up to r_2 with probability $(r\text{-}r_1)/(r_2\text{-}r_1)$ and down to r_1 with probability $(r_2\text{-}r)/(r_2\text{-}r_1)$. The average error of this rounding method is zero, since the expected value of the result of stochastically rounding r is r itself. Using this stochastic rounding method, some of the sub-bit information that is discarded by a deterministic rounding scheme can be maintained.

RESULTS

Network Structure and Simulation Environment

A multi-layer perceptron to be trained on the MNIST dataset is chosen. It has high software accuracies and weight matrices map directly to memristor crossbars, making it suitable for exploring device-algorithm interactions. The impact of the proposed methods can be quantified without any interfering effects from a training optimizer, potentially unoptimized deep network or an overly challenging dataset. The network structure is 400 (input layer) - 100 (hidden layer) - 10 (output layer). The hardware mapping and training on the MNIST dataset is available in NeuroSim V3.0. NeuroSim V3.0 is an open-source integrated simulation framework based on C++ for benchmarking synaptic devices and array architectures via system-level learning accuracy and hardware performance indicators (Chen et al., 2017). As part of this work, modules for MBGD, streaming batch PCA and NMF as well as two weight transfer methods: *rank-sum* and *rank-seq* were implemented and integrated with the existing NeuroSim V3.0 capabilities.

The algorithmic flow between the modules and the device models used are shown in **Figure 4**.

The gradient information obtained during backpropagation is decomposed according to the desired method. The desired weight update is calculated in the form of pulses to update the conductance in hardware. This paper uses the ideal device model and the non-ideal (real) device model with the 1T1R configuration of NeuroSim V3.0 to avoid leakage effects. The ideal device model assumes a reproducible linear relationship between the applied number of pulses and the obtained conductance (**Figure 4B**). In the non-ideal device model, there is non-linearity between the applied pulses and the conductance, which leads to imperfect weight programming and variability in the operation. The nonlinearity values for long term potentiation (LTP) and long term depression (LTD) are 2.40 and −4.88, respectively. The cycle-to-cycle variation is 3.5%. This stochasticity is sufficiently large that sending an "increase weight" pulse can even randomly lead to a "decreased weight" and vice versa (**Figure 4C**). Other hardware parameters are the default values of NeuroSim, for example, the read noise is 0, and the minimum and maximum conductance are ∼3nS and 38 nS, respectively. These default values are extracted from fitting experimental weight update data derived from Ag:a-Si devices (Jo et al., 2010; Chen et al., 2017). For this work, a device with 500 levels is assumed (approximately 9-bit precision). Each change in level is assumed to correspond to one update pulse, with 500 pulses ultimately putting the device in the fully OFF or fully ON state.

Rounding Effects of the Weight Update

Figure 5 shows the training on the MLP network with software (64-bit floating-point precision), ideal memristor device (500 levels, 9-bit) and real device model (500 levels, 9-bit with cycle-to-cycle variability and non-linearity). The MNIST testing accuracies in the regular round-to-nearest truncation vs. the stochastic truncation is determined across various batch sizes in a logarithmic search of the learning rate domain. It can be observed that a network implemented with limited precision memristor devices, but no other non-idealities, achieves SGD accuracy 96.5% similar to a traditional software floating-point implementation. However, the quantization of the weight update shrinks the learning rate window dramatically. Whereas the floating-point implementation can achieve an accuracy > 95% for any learning rate between 0.001 and 1, the low precision memristor-based network can only train with a learning rate between 0.1 and 1 (**Figure 5A**). When stochastic rounding is used, the learning rate window for the quantized memristor model widens significantly, resembling the floating-point implementation (**Figure 5B**). This result highlights the importance of hyperparameter optimization and hardware-sensitive rounding in these low-precision networks.

Learning rate optimization was used to obtain these best accuracy results. The convergence curves for different device models, different batch sizes, and the two rounding methods were run for learning rates spanning eight orders of magnitude from 10^{-6} (0.000001) to $10^{1.6}$ ($\approx$ 40). To optimize the search, this range was explored in logarithmic steps. The learning rates corresponding to the best accuracy for each test set are plotted in **Figure 5E**. As the batch size increased, the value of the optimal learning rate also increased. The learning rates for the round-to-nearest method are higher than the stochastic rounding method, despite their accuracies being similar. This might be due to the fact that stochastic rounding applied to these limited-bit

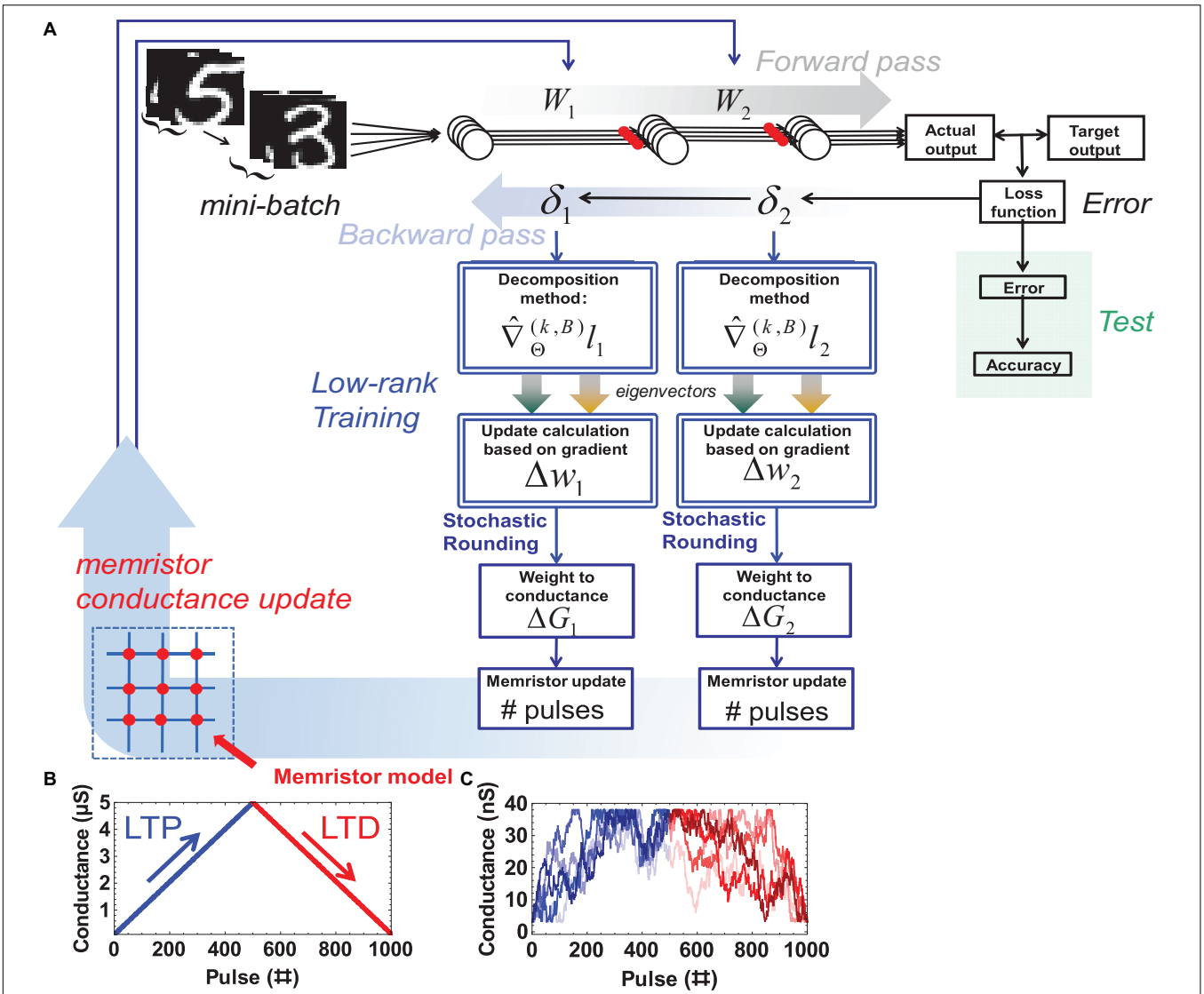


FIGURE 4 | The code structure and device models **(A)** expanded NeuroSim v3.0 with added modules and functions highlighted with a double border. **(B)** Ideal memristor with linear symmetric and reproducible weight update and a large ON / OFF ratio vs. **(C)** Non-ideal memristor model with smaller ON / OFF ratio, weight update nonlinearity, and variability (5 cycles shown).

precision systems can still, over many operations in the time series, on average, keep track of some sub-bit information. The stochastic rounding applied across the weights in the array can preserve statistically more gradient information and carry it over to the next back propagation iterations (Gupta et al., 2015). By comparison, the round-to-nearest truncation discards such gradient information.

Overall, it can be observed that the accuracy increases almost linearly with the log of the batch size for medium batch sizes (up to 128) for both round-to-nearest and stochastic rounding (**Figure 5F**). It plateaus at higher batch sizes converging to the MBGD floating-point software accuracy for higher batch sizes (**Table 1**). For our implementation, the MBGD at large batch sizes are similarly needed to overcome the gradient noise due to the non-ideal memristor synaptic weights. These results show that ideal memristor behavior, while desirable, is not needed on a single layer perceptron. The effects are likely to be even more apparent in larger fixed precision networks due to compounding effects as seen by related work (Gupta et al., 2015). Existing memristors can be used successfully despite their non-idealities and neural networks implemented with real memristor models can achieve software equivalency using appropriate algorithmic methods for training

Streaming Batch PCA With Ideal vs. Non-ideal Weights

An in-depth investigation was done to explore how the accuracy changes with the rank, batch size and transfer method, and the difference between streaming batch PCA algorithm and full rank

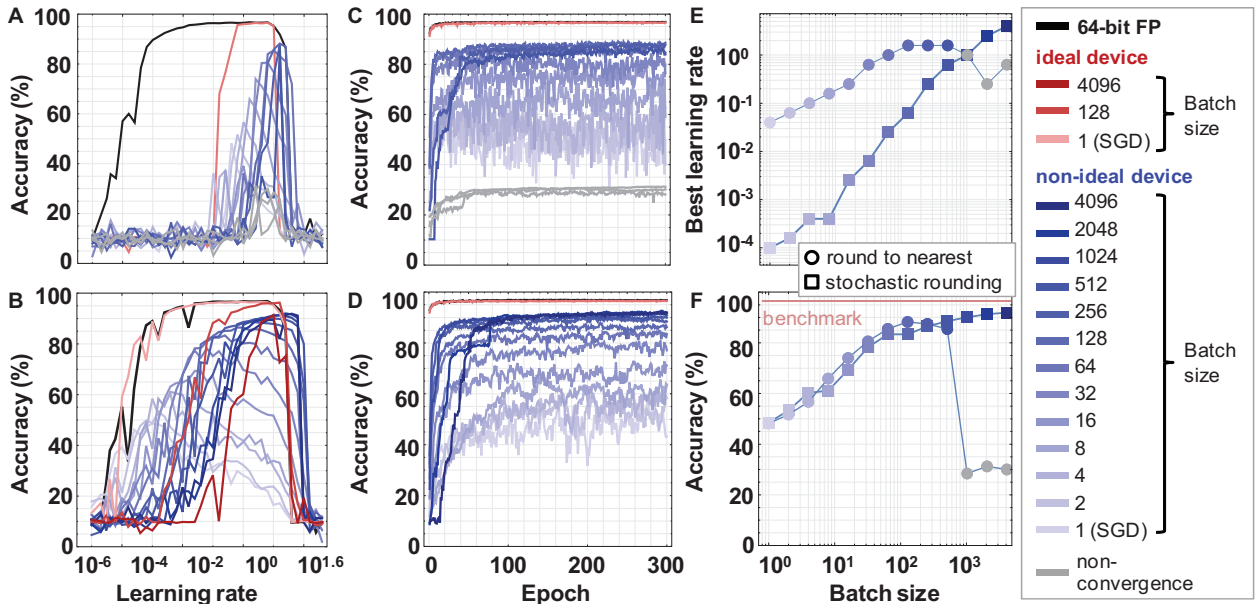


FIGURE 5 | The MBGD results for rounding to nearest and stochastic rounding. Learning rate windows for ideal and non-ideal device models and batch sizes using **(A)** rounding-to-nearest vs. **(B)** stochastic rounding of the delta weight. Best convergence curves for the optimal learning rate for each batch size using **(C)** rounding-to-nearest vs. **(D)** stochastic rounding. **(E)** Optimal learning rate corresponding to the highest accuracy vs. the batch size for non-ideal devices. It is observed that the learning rate increases supra-linearly with the batch size. **(F)** Best accuracy at the optimal learning rate vs. the batch size. In **(E)** and **(F)**, the colorful plot markers represent results at convergence. The faded gray markers for the rounding-to-nearest case represent the inability of the network to train properly due to vanishing gradients in this low precision model. The values for the batch sizes 1 to 4096 are obtained using the realistic memristor model. Representative results for software and ideal device model are also included for comparison.

TABLE 1 | Best accuracy observed between the rounding methods at different batch sizes.

		MBGD Rounding-to-nearest		MBGD Stochastic Rounding	
Synaptic weight	**B**	**Best LR**	**Best Accuracy (%)**	**Best LR**	**Best Accuracy (%)**
64-FP (benchmark)	1	$10^{-0.8} = 0.1585$	96.81	Same	
Ideal device	1	$10^{-0.2} = 0.6310$	96.53	$10^{-0.4} = 0.3981$	96.5
Real device	1	$10^{-1.4} = 0.0398$	48.17	$10^{-4} = 0.0001$	48.45
	2	$10^{-1.2} = 0.0631$	51.71	$10^{-3.8} = 0.0002$	53.34
	4	$10^{-0.1} = 0.1000$	56.68	$10^{-3.4} = 0.0004$	60.28
	8	$10^{-0.8} = 0.1585$	65.88	$10^{-3.4} = 0.0004$	61.13
	16	$10^{-0.6} = 0.2512$	74.02	$10^{-2.6} = 0.0025$	69.34
	32	$10^{-0.2} = 0.6310$	80.58	$10^{-2.2} = 0.0063$	78.24
	64	$10^{1} = 1.0000$	85.50	$10^{-1.6} = 0.0251$	83.35
	128	$10^{0.2} = 1.5849$	88.24	$10^{-1.2} = 0.0631$	86.49
	256	$10^{0.2} = 1.5849$	87.48	$10^{0.2} = 0.2512$	88.53
	512	$10^{0.2} = 1.5849$	85.43	$10^{-0.2} = 0.6310$	90.06
	1024	$10^{0} = 1.0000$	28.38	$10^{0} = 1.0000$	91.28
	2048	$10^{-0.6} = 0.2512$	31.20	$10^{0.4} = 2.5119$	91.99
	4096	$10^{-0.2} = 0.6310$	29.99	$10^{0.6} = 3.9811$	91.93
64-FP	4096	$10^{0.2} = 1.5849$	93.42	Same	
	8192	$10^{0.2} = 1.5849$	90.73	Same	

MBGD. Two batch sizes were investigated: 128 and 4096. The learning rates used for this streaming batch PCA investigation correspond to the best accuracies obtained by these batch sizes for MBGD. The decomposition method was applied to both layers at the same rank. The ranks investigated were 1, 3, and 10.

Figure 6 summarizes the *rank-sum* results and as expected, the accuracy of the rank 1 results was lower than that of rank 3 and rank 10 for both batch size 128 and 4096 for both the device models. The performance for the ideal device model shows that the performance slightly decrease for MBGD at high batch size.

However, the performance for non-ideal devices increases with the batch size. The rank-3 decomposition does seem to perform well by comparison with MBGD, particularly at the larger batch size. The convergence performance of rank 10 is at the same level as that of MBGD for the *rank-sum* transfer method. Additionally, with the increase in the rank, the convergence curve tends to smoothen and converge somewhat faster, achieving the desired accuracy in ≈ 25 epochs. The investigation of the impact of block size *b* is included as **Supplementary Figure 1**.

While the *rank-sum* weight transfer method works very well for streaming batch PCA and achieves close to MBGD performance, its full hardware implementation would be difficult since the gradient approximation needs to be recomposed externally prior to being transferred to the memristor matrix. By contrast, gradient recomposition of *rank-seq* requires minimal hardware overhead. The results for *rank-seq* are summarized in **Figure 7**.

For ideal device, the accuracies are similar for both rank-sum and rank-seq. By comparison, for the non-ideal devices, accuracies around ≈ 70% are obtained for ranks 3 and 10 at both batch sizes 128 and 4096. This is 15 to 20 percentage points lower than the *rank-sum* and full rank MBGD results. These results show that the streaming batch PCA using *rank-seq* transfer method cannot approximate the MBGD results, even at high ranks. This is likely because the principal components of streaming batch PCA can have positive and negative elements, creating an oscillatory effect due to the programming of the non-ideal memristive weights (Scholz et al., 2008). This effect is observed indirectly in the noisy convergence curves.

Streaming Batch PCA With Ideal vs. Non-ideal Weights

Figure 8 summarizes the *rank-sum* NMF results for the different ranks at the two different batch sizes and compares them with full rank MBGD. For ideal device, the rank 1 has lower performance, but rank 3 and rank 10 can approximate MBGD well, particularly at batch size 128. For non-ideal devices, the NMF can approximate the gradient information fairly well, particularly at rank 10. Rank 1 has extremely poor performance, similar to SGD. Rank 3 performs well and can converge, but its accuracy is still ≈ 5% to 10% lower than the equivalent MBGD result at the respective batch size. It is also worth noting that a decline in the accuracies of these lower ranks can be observed as the training progresses. Higher rank is needed to observe satisfactory accuracy and training stability. The result of rank 10 was only 1% to 2% lower than that of the MBGD algorithm. One reason for the high accuracy in the case of rank 10 is that because the second layer has only 10 neurons, rank 10 is actually equivalent to full rank training in the last layer, though not in the first layer.

The results for the *rank-seq* transfer method applied to NMF are shown in **Figure 9**. For the ideal device, the accuracies are similar to rank-sum as expected. By comparison, for the non-ideal devices, rank 3 achieves ≈ 70% accuracy at batch size 128 and ≈ 80% accuracy at batch size 4096. For rank 10, the NMF algorithm performs within 2% to 3% degradation of the MBGD results for the respective batch size. Overall, the *rank-seq* results are similar with the *rank-sum* ones at the equivalent rank. This is likely due to the fact that there is minimal overlap between the ranks for this additive decomposition method (Lee and Seung, 2000).

Comparison Between the Algorithms

The streaming batch PCA shows the most efficient compression of the batch gradient information. It obtains better accuracies than NMF for all ranks and batch sizes when the *rank-sum* transfer method is used. Streaming batch PCA *rank-sum* for rank 10 has an accuracy equivalent to MBGD ≈ 91.5% for batch size 4096. This result is around 5 percentage points lower than the traditional 64-bit floating-point algorithmic implementation for MNIST training at batch size 1 (SGD) which is the target / benchmark result for this work. This result, summarized in **Figure 10** and **Table 2**, shows that decomposition methods in conjunction with large batch size MBGD training can overcome memristive synaptic device non-idealities and achieve close to software-equivalent accuracies.

However, streaming batch PCA has its challenges. The main problem is that it operates on the eigenspace of the entire synaptic weight matrix, statistically representing the direction of largest variance, but there is no clear spatial explanation for negative numbers. Therefore the transfer of the gradient information into the memristor matrix by mapping the gradient data to number of pulses for the update (open loop transfer) is challenging as principal components can have positive and negative signs leading to inefficient oscillatory programming. For this reason, rank-by-rank weight transfer *rank-seq* underperforms by comparison with *rank-sum* for streaming batch PCA. It is important to point out that oscillatory behavior *per se* can be supported by resistive crossbar arrays via successive increase and decrease in conductance. The devices can be tuned with desired precision, but it might take very long trains of pulses and it is not desirable from a speed perspective when using devices with non-linearity and variability. If positive and negative updates to the weight are needed in rapid succession, the device programming becomes very inefficient. Therefore the transfer of the gradient information into the physical device matrices by mapping the gradient data to number of pulses for the update is challenging. In comparison, NMF calculates an approximate matrix factorization with separate positive and negative gradient information which causes the updates to avoid overlapping with one another.

By avoiding overlapping ranks, NMF has superior performance at high ranks by comparison with streaming batch PCA. For example, at rank 3, NMF *rank-seq* outperforms streaming batch PCA *rank-seq* by ≈ 5%. At rank 10, the gap is 17%. The best *rank-seq* accuracy is obtained by NMF rank 10 (88.87%) and it is less than 2% lower than the best *rank-sum* accuracy obtained via streaming batch PCA at rank 10 (90.65%). This means in practice that the NMF factorization produces the set of optimally efficient rank 1 update operations to training memristor neural networks.

The main drawback of applying the proposed methodology is related to accuracy. MBGD, particularly at large batch sizes, has lower accuracy than lower batch sizes (Goyal et al., 2017;

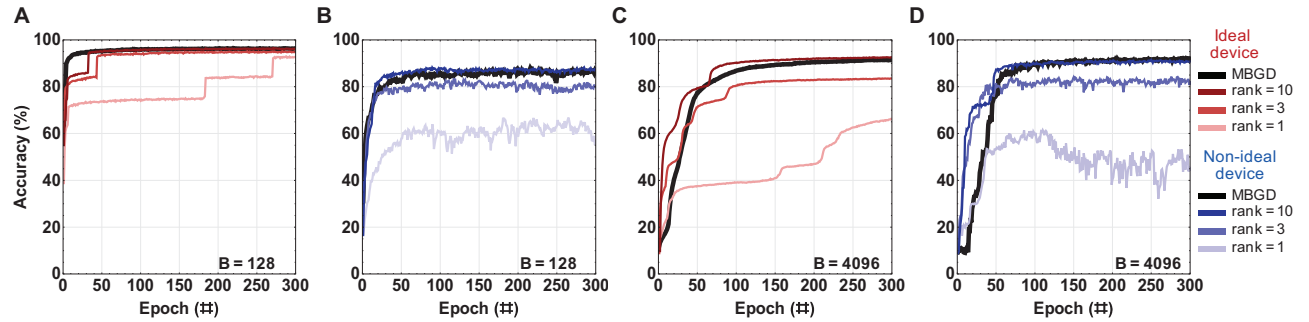


FIGURE 6 | The rank-sum streaming batch PCA results for a multi-layer perceptron with ideal vs. non-ideal memristive weights at different ranks and batch sizes. **(A,B)** Convergence curves using the rank-sum weight transfer method at batch size = 128; and similarly **(C,D)** at batch size = 4096. The results show that a low-rank gradient decomposition can approximate the MBGD results fairly well, particularly for ranks 3 and 10. For all these experiments, the rounding method is stochastic and block size = 32.

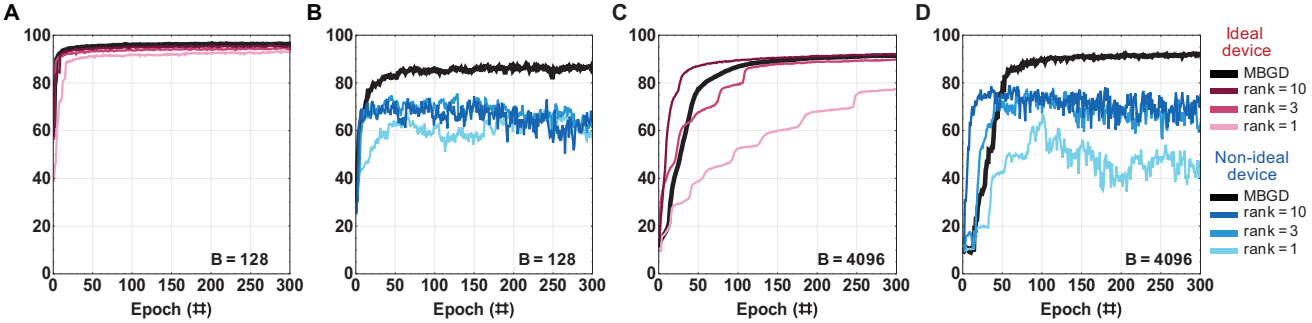


FIGURE 7 | The rank-seq streaming batch PCA results for a multi-layer perceptron with ideal vs. non-ideal memristive weights at different ranks and batch sizes. **(A,B)** Convergence curves using the rank-seq weight transfer method at batch size = 128; and similarly **(C,D)** at batch size = 4096. The results show that the streaming batch PCA using rank-seq transfer method cannot approximate the MBGD results, even at high ranks. For all these experiments, the rounding method is stochastic and block size = 32.

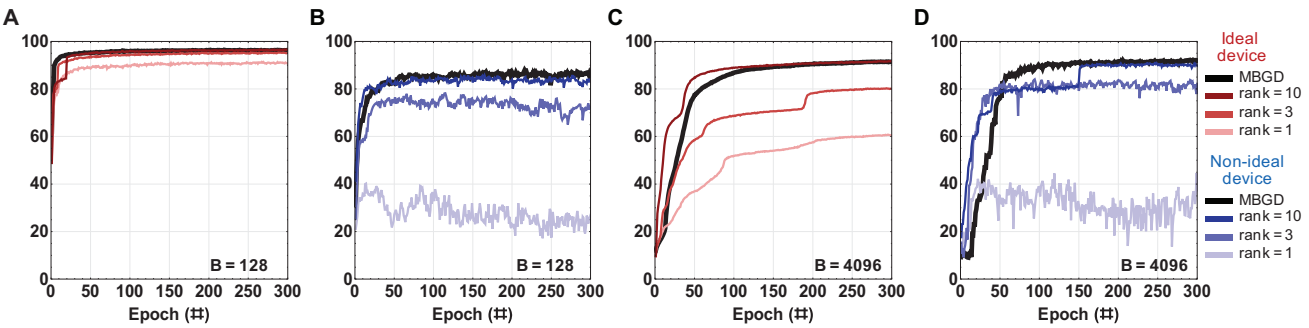


FIGURE 8 | The rank-sum Non-Negative Matrix Factorization (NMF) results for a multi-layer perceptron with ideal vs. non-ideal memristive weights at different ranks and batch sizes. **(A,B)** Convergence curves using the rank-sum weight transfer method at batch size = 128; and similarly **(C,D)** at batch size = 4096. The results show that the NMF can approximate fairly well the MBGD results, particularly for ranks 3 and 10. For all these experiments, the rounding method is stochastic.

Golmant et al., 2018). Furthermore, low rank decompositions of the MBGD gradient information can negatively affect accuracy when large networks and complex datasets are used for training. The results of this work show that it is possible to obtain low rank decomposition accuracies as close as 2% to 3% from the MBGD accuracies when large batch sizes are used. This slight penalty in accuracy comes at the potential advantage of large storage capacity for the network parameters. This tradeoff needs to be investigated further by taking accuracy targets, hardware overhead, network layer sizes, and other hyperparameters into consideration.

However, their full potential can only be explored on dedicated hardware co-processors. For example, the Streaming Batch PCA algorithm requires computationally intensive QR

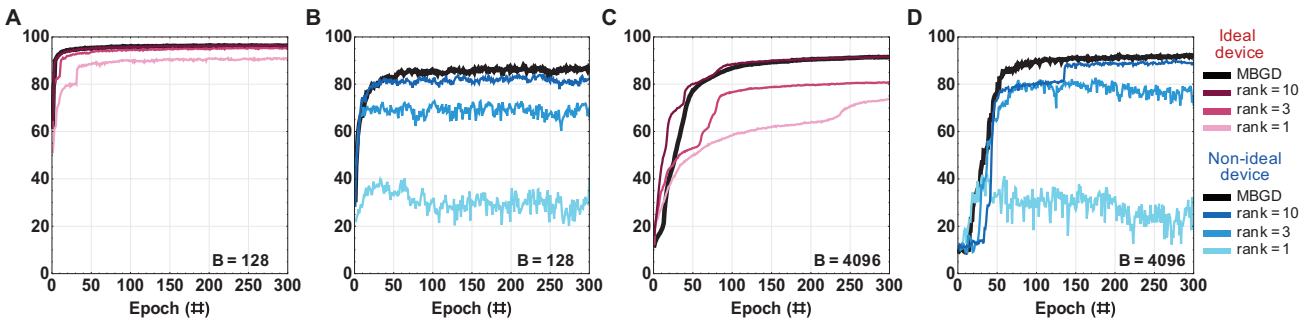


FIGURE 9 | The rank-seq NMF results for a multi-layer perceptron with ideal vs. non-ideal memristive weights at different ranks and batch sizes. **(A,B)** Convergence curves using the rank-seq weight transfer method at batch size = 128; and similarly **(C,D)** at batch size = 4096. The results show that the NMF using rank-seq transfer method can approximate the MBGD results well at high ranks. For all these experiments, the rounding method is stochastic.

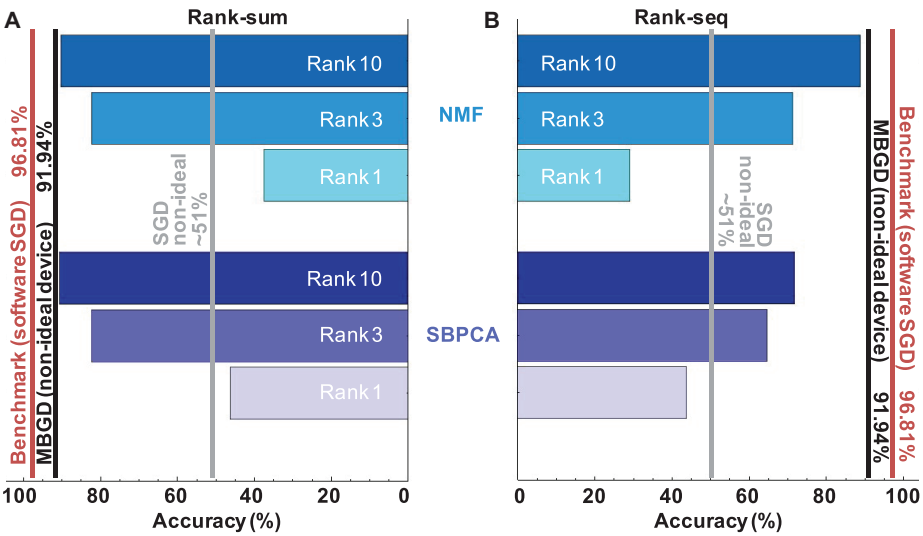


FIGURE 10 | Comparison of streaming batch PCA and NMF results for rank-sum and rank-seq for a multi-layer perceptron with non-ideal memristive weights at different ranks and batch sizes. The results show that the streaming batch PCA using rank-sum slightly outperforms NMF rank-sum and can approximate very well the MBGD results at high ranks. By comparison, NMF rank-seq significantly outperforms streaming batch PCA rank-seq for higher ranks (e.g., 3 and 10) and can approximate well the MBGD results. For all these experiments, the rounding method is stochastic, batch size = 4096 and block size = 32.

factorization (Huang et al., 2020a). The NMF algorithm requires explicit calculation of the full batch matrix to get the separate non-negative components. Optimized NMF algorithms mappable to hardware co-processors need to be developed, e.g., streaming variants (see Section 2 and **Supplementary Figure 2**). These limitations can be overcome in dedicated hardware accelerators, e.g., based on systolic arrays. A discussion of the hardware considerations is included in the **Supplementary Material**. Issues related to energy efficiency and speed need hardware models for the decomposition modules to be integrated with the existing circuit and device models as part of a comprehensive design verification framework (Hoskins et al., 2021).

Applicability and Scale-Up Potential

In general, the proposed algorithms should be broadly applicable to any family of weights arrays in a matrix where the weights are trained by gradient descent. These simulation results highlight the potential of low-rank gradient decompositions in neural networks using memristor weights and are the first steps toward training co-processors to support the scale-up of machine learning models in such hardware. Several recent works demonstrate the applicability of memristor crossbars to recurrent and convolutional neural networks (Li et al., 2019; Wang et al., 2019; Lin et al., 2020). The same decomposition and implementation principles could be applied to fully connected recurrent layers. For a convolutional network, the fully connected layers performing the classification in a deep network can benefit from these decomposition methods. It is therefore possible to consider the application of the proposed methods to deeper, more complex networks.

For spiking neural networks, this property can prove important since gradient based methods have recently taken on renewed popularity in the training of such networks, especially

TABLE 2 | Summary of the best results for different ranks, batch sizes and truncation methods for streaming batch PCA vs. NMF.

		Streaming Batch PCA		NMF	
Data type		**Rank-sum accuracy (%)**	**Rank-seq accuracy (%)**	**Rank-sum accuracy (%)**	**Rank-seq accuracy (%)**
SGD			51.04		
MBGD 128			86.49		
MBGD 4096			91.94		
Batchsize 128Block 32	Rank1	58.60	58.08	24.15	31.02
	Rank3	80.04	61.97	71.87	70.06
	Rank10	87.30	64.76	83.26	82.03
Batchsize 4096Block 32	Rank1	46.37	43.71	37.38	29.09
	Rank3	82.33	64.71	82.15	71.27
	Rank10	90.65	71.78	89.93	88.87
Batchsize 4096Block 128	Rank1	37.49	51.84	25.26	26.02
	Rank3	81.28	63.80	81.44	76.1
	Rank10	90.78	69.52	90.12	89.39
Batchsize 4096Block 512	Rank1	40.30	52.37	18.63	27.37
	Rank3	85.31	63.19	76.83	76.10
	Rank10	91.03	71.41	90.33	88.17
	Rank1	45.29	48.37	27.48	25.69
Batchsize 4096Block 1024	Rank3	84.40	65.81	75.77	77.79
	Rank10	91.48	72.10	90.28	89.08

Realistic device model used for all these results.

through the use of surrogate gradient methods (Neftci et al., 2019). An increasingly common practice, despite the lack of biological plausibility, is to use mini-batch GPU acceleration of spiking networks to train them more rapidly (Neftci et al., 2017; Payvand et al., 2020). While researchers cite that future hardware will be able to more efficiently train using batch sizes of 1 (Stewart et al., 2020), this has also frequently been proposed as the ideal batch size for using memristor-based artificial neural networks due to the memory overhead associated with gradient data. However, as shown in this work, low batch size training leads to catastrophically poor performance and larger batch sizes are needed to improve training of non-ideal hysteretic devices.

Our approach to compress gradient based information as presented here could be an important step toward developing biologically plausible batch averaging during long term learning. The methods can be adapted to require only local neuronal information, thus leading to methods resilient to local nanodevice non-idealities. Compression algorithms similar to the ones studied here, e.g., Oja's learning rule (Oja, 1982), were initially introduced as biologically plausible means to learn incoming data. Therefore, they could be used in a realistic way to efficiently learn surrogate gradients during the training of spiking neural networks.

CONCLUSION

This paper investigated mini-batch training and gradient decomposition algorithmic methods to overcome the hardware non-idealities of memristor-based neural networks. By testing two different decomposition methods (streaming batch PCA and NMF) and two different weight transfer methods (*rank-sum* and *rank-seq*) for different memristor device models and ranks, we showed that the combination of the above methods is a feasible method for training the fully connected networks implemented with non-ideal devices via rank 1 updates. Our results indicate that stochastic rounding can overcome the loss of precision due to the quantization error from the vanishing gradient issue, which is of particular importance when it comes to the synaptic devices of low-bit precision, such as memristors. While the low-rank decomposition methods both produced accuracies close to those of full-rank MBGD, the choice of the update method was particularly significant for the gradient information transfer to the memristor matrix hardware. Overall, NMF produced a less efficient compression of the batch gradient than that of streaming batch PCA. However, we speculate that it was better for the rank-by-rank transfer to the memristor crossbar since all the gradient components were additive, thus eliminating the effect of device update hysteresis, though this needs further investigation. The *rank-seq* NMF is more in line with the physical constraints of memristor synaptic weights and may represent the optimal set of rank-1 updates that can be used to train a memristor array in an open loop fashion.

Future work will focus on expanding these results to deeper networks, including other types of layers, such as recurrent layers and applicability to spiking neural networks. In addition, other hardware-aware decomposition methods will be investigated. This methodology can be applied to neural networks implemented with other types of non-volatile memory devices such as phase change memory and flash technology. Ultimately, the goal is to test these proposed algorithms in full

hardware implementations in memristor-based accelerators that demonstrate software equivalency despite device non-idealities.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

JZ and SH developed the decomposition code modules and performed the simulations. OY helped with the execution time analysis. YG helped with the mini-batch gradient descent code. BH provided guidance and support. GA supervised the work. All authors participated in data analysis, discussed the results, and co-edited the manuscript.

FUNDING

The authors acknowledge funding support from the ONR/DARPA grant N00014-20-1-2031, the GW University Facilitating Fund and NIST.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.749811/full#supplementary-material

REFERENCES

Adam, G. C., Khiat, A., and Prodromakis, T. (2018). Challenges hindering memristive neuromorphic hardware from going mainstream. *Nat. Commun.* 9, 1–4. doi: 10.1038/s41467-018-07565-4

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., Di Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* 558, 60–67. doi: 10.1038/s41586-018-0180-5

Argall, F. (1968). Switching phenomena in titanium oxide thin films. *Solid State Electron.* 11, 535–541. t doi: 10.1016/0038-1101(68)90092-0

Baek, I., Lee, M., Seo, S., Lee, M., Seo, D., Suh, D.-S., et al. (2004). "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proceedings of the IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004*, (San Francisco, CA: IEEE), 587–590. doi: 10.1109/IEDM.2004.1419228

Barnes, R., Cooke-Yarborough, E., and Thomas, D. (1951). An electronic digital computor using cold cathode counting tubes for storage. *Electron. Eng.* 23, 286–291.

Berdan, R., Marukame, T., Ota, K., Yamaguchi, M., Saitoh, M., Fujii, S., et al. (2020). Low-power linear computation using nonlinear ferroelectric tunnel junction memristors. *Nat. Electron.* 3, 1–8. doi: 10.1038/s41928-020-0405-0

Boybat, I., Le Gallo, M., Nandakumar, S., Moraitis, T., Parnell, T., Tuma, T., et al. (2018). Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* 9, 1–12. doi: 10.1038/s41467-018-04933-y

Burrello, A., Marchioni, A., Brunelli, D., and Benini, L. (2019). "Embedding principal component analysis for data reduction in structural health monitoring on low-cost iot gateways," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, (Alghero: ACM), 235–239. doi: 10.1145/3310273.3322822

Ceze, L., Hasler, J., Likharev, K., Seo, J.-S., Sherwood, T., Strukov, D., et al. (2016). "Nanoelectronic neurocomputing: status and prospects," in *Proceedings of the 2016 74th Annual Device Research Conference (DRC)*, (Newark, DE: IEEE), 1–2. doi: 10.1109/DRC.2016.7548506

Chang, M.-F., Chiu, P.-F., Wu, W.-C., Chuang, C.-H., and Sheu, S.-S. (2011). "Challenges and trends in low-power 3D die-stacked IC designs using RAM, memristor logic, and resistive memory (ReRAM)," in *Proceedings of the 2011 9th IEEE International Conference on ASIC*, (Xiamen: IEEE), 299–302.

Chen, P.-Y., Peng, X., and Yu, S. (2017). "NeuroSim+: an integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *Proceedings of the 2017 IEEE International Electron Devices Meeting (IEDM)*, (San Francisco, CA: IEEE), 6.1.1–6.1.4.

Chen, W.-H., Li, K.-X., Lin, W.-Y., Hsu, K.-H., Li, P.-Y., Yang, C.-H., et al. (2018). "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *Proceedings of the 2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, (San Francisco, CA: IEEE), 494–496.

Chen, Y. (2020). ReRAM: history, status, and future. *IEEE Trans. Electron Devices* 67, 1420–1433. doi: 10.1109/TED.2019.2961505

Cichocki, A., and Phan, A.-H. (2009). Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 92, 708–721. doi: 10.1587/transfun.E92.A.708

Cichocki, A., Zdunek, R., Phan, A. H., and Amari, S.-I. (2009). *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*. Hoboken, NJ: John Wiley & Sons. doi: 10.1002/9780470747278

Dearnaley, G., Stoneham, A., and Morgan, D. (1970). Electrical phenomena in amorphous oxide films. *Rep. Prog. Phys.* 33:1129. doi: 10.1088/0034-4885/33/3/306

Forsythe, G. E. (1950). "Round-off errors in numerical integration on automatic machinery-preliminary report", in: bulletin of the American mathematical society: AMER MATHEMATICAL SOC 201 CHARLES ST. *Providence* 0294, 61–61.

Gao, Y., Wu, S., and Adam, G. C. (2020). "Batch training for neuromorphic systems with device non-idealities," in *International Conference on Neuromorphic Systems 2020*, (New York, NY: ACM), 1–4. doi: 10.1145/3407197.3407208

Garipov, T., Podoprikhin, D., Novikov, A., and Vetrov, D. (2016). Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv* [Preprint] arXiv:1611.03214,

Gokmen, T., and Haensch, W. (2020). Algorithm for training neural networks on resistive device arrays. *Front. Neurosci.* 14:103. doi: 10.3389/fnins.2020.00103

Gokmen, T., and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* 10:333. doi: 10.3389/fnins.2016.00333

Golmant, N., Vemuri, N., Yao, Z., Feinberg, V., Gholami, A., Rothauge, K., et al. (2018). On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv* [Preprint] arXiv:1811.12941,

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2017). Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv* [Preprint] arXiv:1706.02677,

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning: PMLR*, (Lille : JMLR.org), 1737–1746.

Haensch, W., Gokmen, T., and Puri, R. (2018). The next generation of deep learning hardware: analog computing. *Proc. IEEE* 107, 108–122. doi: 10.1109/JPROC.2018.2871057

Hickmott, T. (1962). Low-frequency negative resistance in thin anodic oxide films. *J. Appl. Phys.* 33, 2669–2682. doi: 10.1063/1.1702530

Hirtzlin, T., Penkovsky, B., Klein, J.-O., Locatelli, N., Vincent, A. F., Bocquet, M., et al. (2019). "Implementing binarized neural networks with magnetoresistive ram without error correction," in *Proceedings of the 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, (Qingdao: IEEE), 1–5. doi: 10.1109/NANOARCH47378.2019.181300

Hoskins, B. D., Daniels, M. W., Huang, S., Madhavan, A., Adam, G. C., Zhitenev, N., et al. (2019). Streaming batch eigenupdates for hardware neural networks. *Front. Neurosci.* 13:793. doi: 10.3389/fnins.2019.00793

Hoskins, B. D., Ma, W., Fream, M., Liu, M., Daniels, M. W., Madsen, R., et al. (2021). "Design for verification in a resistive neural network prototype," in *Proceedings of the International Conference on Neuromorphic Systems (ICONS) July 27–29, 2021*, Knoxville, TN. doi: 10.1145/3477145.3477260

Hu, M., Graves, C. E., Li, C., Li, Y., Ge, N., Montgomery, E., et al. (2018). Memristor-based analog computation and neural network classification with a dot product engine. *Adv. Mater.* 30:1705914. doi: 10.1002/adma.201705914

Huang, S., Hoskins, B. D., Daniels, M. W., Stiles, M. D., and Adam, G. C. (2020a). Memory-efficient training with streaming dimensionality reduction. *arXiv* [Preprint] arXiv:2004.12041,

Huang, S., Hoskins, B. D., Daniels, M. W., Stiles, M. D., and Adam, G. C. (2020b). Streaming batch gradient tracking for neural network training (student abstract). *Proc. AAAI Conf. Artif. Intell.* 34, 13813–13814.

Hull, T. E., and Swenson, J. R. (1966). Tests of probabilistic models for propagation of roundoff errors. *Commun. ACM* 9, 108–113. doi: 10.1145/365170.365212

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* 10, 1297–1301. doi: 10.1021/nl904092h

Kataeva, I., Merrikh-Bayat, F., Zamanidoost, E., and Strukov, D. (2015). "Efficient training algorithms for neural networks based on memristive crossbar circuits," in *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN)*, (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280785

Kim, W., Bruce, R., Masuda, T., Fraczak, G., Gong, N., Adusumilli, P., et al. (2019). "Confined PCM-based analog synaptic devices offering low resistance-drift and 1000 programmable states for deep learning," in *Proceedings of the 2019 Symposium on VLSI Technology*, (Kyoto: IEEE), T66–T67. doi: 10.23919/VLSIT.2019.8776551

Langston, J. (2020). *Microsoft Announces New Supercomputer, Lays Out Vision for Future AI Work. Microsoft*. Available online at: https://blogs. microsoft.com/ai/openai-azure-supercomputer/ (accessed August 27, 2020).

Lee, D. D., and Seung, H. S. (2000). "Algorithms for non-negative matrix factorization," in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, (Cambridge, MA: MIT Press), 535–541.

Lee, D. D., and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature* 401, 788–791. doi: 10.1038/44565

Li, C., Wang, Z., Rao, M., Belkin, D., Song, W., Jiang, H., et al. (2019). Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* 1, 49–57. doi: 10.1038/s42256-018-0001-4

Lin, P., Li, C., Wang, Z., Li, Y., Jiang, H., Song, W., et al. (2020). Three-dimensional memristor circuits as complex neural networks. *Nat. Electron.* 3, 225–232. doi: 10.1038/s41928-020-0397-9

Lin, Y.-H., Wang, C.-H., Lee, M.-H., Lee, D.-Y., Lin, Y.-Y., Lee, F.-M., et al. (2019). Performance impacts of analog ReRAM non-ideality on neuromorphic computing. *IEEE Trans. Electron Devices* 66, 1289–1295. doi: 10.1109/TED.2019.2894273

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Nugent, M. A., and Molter, T. W. (2014). AHaH computing–from metastable switches to attractors to machine learning. *PLoS One* 9:e85175. doi: 10.1371/journal.pone.0085175

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *J. Math. Biol.* 15, 267–273. doi: 10.1007/BF00275687

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Netw.* 5, 927–935. doi: 10.1016/S0893-6080(05)80089-9

Oxley, D. P. (1977). Electroforming, switching and memory effects in oxide thin films. *Electrocomp. Sci. Technol.* 3, 217–224. doi: 10.1155/APEC.3.217

Paatero, P., and Tapper, U. (1994). Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics* 5, 111–126. doi: 10.1002/env.3170050203

Pagnia, H., and Sotnik, N. (1988). Bistable switching in electroformed metal–insulator–metal devices. *Phys. Status Solidi* 108, 11–65.

Payvand, M., Fouda, M. E., Kurdahi, F., Eltawil, A. M., and Neftci, E. O. (2020). On-chip error-triggered learning of multi-layer memristive spiking neural networks. *IEEE J. Emerg. Sel. Top. Circ. Syst.* 10, 522–535. doi: 10.1109/JETCAS.2020.3040248

Payvand, M., Nair, M. V., Müller, L. K., and Indiveri, G. (2019). A neuromorphic systems approach to in-memory computing with non-ideal memristive devices: from mitigation to exploitation. *Faraday Discuss.* 213, 487–510. doi: 10.1039/C8FD00114F

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 61–64. doi: 10.1038/nature14441

Rohde, C., Choi, B. J., Jeong, D. S., Choi, S., Zhao, J.-S., and Hwang, C. S. (2005). Identification of a determining parameter for resistive switching of Ti O 2 thin films. *Appl. Phys. Lett.* 86, 262907. doi: 10.1063/1.1968416

Schein, A., Zhou, M., Blei, D., and Wallach, H. (2016). "Bayesian poisson tucker decomposition for learning the structure of international relations," in *Proceedings of the 33rd International Conference on Machine Learning: PMLR June 19–24, 2016*, New York, NY. 2810–2819.

Scholz, M., Fraunholz, M., and Selbig, J. (2008). "Nonlinear principal component analysis: neural network models and applications," in *Principal manifolds for data visualization and dimension reduction*, eds A. N. Gorban, B. Kégl, D. C. Wunsch, and A. Y. Zinovyev (Berlin: Springer), 44–67. doi: 10.1007/978-3-540-73750-6_2

Seo, S., Lee, M., Seo, D., Jeoung, E., Suh, D.-S., Joung, Y., et al. (2004). Reproducible resistance switching in polycrystalline NiO films. *Appl. Phys. Lett.* 85, 5655–5657. doi: 10.1063/1.1831560

Serb, A., Redman-White, W., Papavassiliou, C., and Prodromakis, T. (2015). Practical determination of individual element resistive states in selectorless RRAM arrays. *IEEE Trans. Circ. Syst. I Regul. Papers* 63, 827–835. doi: 10.1109/TCSI.2015.2476296

She, X., Long, Y., and Mukhopadhyay, S. (2019). "Improving robustness of reram-based spiking neural network accelerator with stochastic spike-timing-dependent-plasticity," in *Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN)*, (Budapest: IEEE), 1–8. doi: 10.1109/IJCNN.2019.8851825

Stewart, K., Orchard, G., Shrestha, S. B., and Neftci, E. (2020). "On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor," in *Proceedings of the 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, (Genova: IEEE), 223–227. doi: 10.1109/AICAS48895.2020.9073961

Strubell, E., Ganesh, A., and McCallum, A. (2020). Energy and policy considerations for modern deep learning research. *Proc. AAAI Conf. Artif. Intell.* 34, 13693–13696. doi: 10.1609/aaai.v34i09.7123

Vogels, T., Karimireddy, S. P., and Jaggi, M. (2019). PowerSGD: practical low-rank gradient compression for distributed optimization. *Adv. Neural Inform. Process. Syst.* 32, 14236–14245.

Wang, D., Gao, X., and Wang, X. (2015). Semi-supervised nonnegative matrix factorization via constraint propagation. *IEEE Trans. Cybern.* 46, 233–244. doi: 10.1109/TCYB.2015.2399533

Wang, Z., Li, C., Lin, P., Rao, M., Nie, Y., Song, W., et al. (2019). In situ training of feed-forward and recurrent convolutional memristor networks. *Nat. Mach. Intell.* 1, 434–442. doi: 10.1038/s42256-019-0089-1

Wong, H.-S. P., Lee, H.-Y., Yu, S., Chen, Y.-S., Wu, Y., Chen, P.-S., et al. (2012). Metal–oxide RRAM. *Proc. IEEE* 100, 1951–1970. doi: 10.1109/JPROC.2012.2190369

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers in Neuroscience

REVIEW
published: 02 December 2021
doi: 10.3389/fnins.2021.728086

Brain-Inspired Hardware Solutions for Inference in Bayesian Networks

Leila Bagheriye and Johan Kwisthout*

Foundations of Natural and Stochastic Computing, Donders Institute for Brain, Cognition and Behaviour, Radboud University, Nijmegen, Netherlands

The implementation of inference (i.e., computing posterior probabilities) in Bayesian networks using a conventional computing paradigm turns out to be inefficient in terms of energy, time, and space, due to the substantial resources required by floating-point operations. A departure from conventional computing systems to make use of the high parallelism of Bayesian inference has attracted recent attention, particularly in the hardware implementation of Bayesian networks. These efforts lead to several implementations ranging from digital circuits, mixed-signal circuits, to analog circuits by leveraging new emerging nonvolatile devices. Several stochastic computing architectures using Bayesian stochastic variables have been proposed, from FPGA-like architectures to brain-inspired architectures such as crossbar arrays. This comprehensive review paper discusses different hardware implementations of Bayesian networks considering different devices, circuits, and architectures, as well as a more futuristic overview to solve existing hardware implementation problems.

Keywords: brain inspired computing, Bayesian inference, spiking neural networks (SNN), nonvolatile, stochastic computing

OPEN ACCESS

Edited by:
Melika Payvand,
ETH Zürich, Switzerland

Reviewed by:
Mohammed Fouda,
University of California, Irvine, United States
Alex James,
Indian Institute of Information Technology and Management Kerala (IITMK), India

****Correspondence:***
Leila Bagheriye
Leila.Bagheriye@donders.ru.nl

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *20 June 2021*
Accepted: *11 October 2021*
Published: *02 December 2021*

Citation:
Bagheriye L and Kwisthout J (2021) Brain-Inspired Hardware Solutions for Inference in Bayesian Networks. Front. Neurosci. 15:728086. doi: 10.3389/fnins.2021.728086

INTRODUCTION

Bayesian inference (i.e., the computation of a posterior probability given a prior probability and new evidence; Jaynes, 2003) is one of the most crucial problems in artificial intelligence (AI), in areas as varied as statistical machine learning (Tipping, 2003; Theodoridis, 2015), causal discovery (Heckerman et al., 1999), automatic speech recognition (Zweig and Russell, 1998), spam filtering (Gómez Hidalgo et al., 2006), and clinical decision support systems (Sesen et al., 2013). It is a powerful method for fusing independent (possibly conflicting) data for decision-making in robotic, biological, and multi-sensorimotor systems (Bessière et al., 2008). Bayesian networks (Pearl, 1988) allow for a concise representation of stochastic variables and their independence and the computation of any posterior probability of interest in the domain spanned by the variables. The structure and strength of the relationships can be elicited from domain experts (Druzdzel and van der Gaag, 1995) or, more commonly, learned from data using algorithms such as expectation-maximization or maximum likelihood estimation (Heckerman et al., 1995; Ji et al., 2015). However, both the inference problem (Cooper, 1990) and the learning problem (Chickering, 1996) are NP-hard problems in general.

As a result, an efficient implementation of Bayesian networks is highly desirable. Although the implementation of inference on a large Bayesian network on conventional general-purpose computers provides high precision, it is inefficient in terms of time and energy consumption. Several complex floating-point calculations are required to estimate the probability of occurrence of a variable since the network is composed of various interacting causal variables (Shim et al., 2017). Moreover, the high parallelism feature of Bayesian inference is not used efficiently in conventional computing systems (F. Kungl et al., 2019). Conventional systems need exact values throughout the computation, preventing the use of the stochastic computing paradigm that consumes less power (Khasanvis et al., 2015a). To realize stochastic computing-based Bayesian inference especially using emerging nanodevices, it is highly needed to develop a robust hardware structure to overcome the characteristic imperfection of these new technologies. On the other hand, the practical realization and usage of large Bayesian networks has been problematic due to the abovementioned intractability of inference (Faria et al., 2021). Therefore, any hardware implementation of Bayesian inference needs to pay attention to a hierarchy of device, circuit, architecture, and algorithmic improvements.

Various approaches and architectures for Bayesian network hardware implementations have been developed; in the literature, approaches such as probabilistic computing platforms based on Field Programmable Gate Arrays (FPGAs), fully digital systems with stochastic digital circuits, analog-based probabilistic circuits, mixed-signal approaches, stochastic computing platforms with scaled nanomagnets, and Intel's Loihi chip have been proposed.

In this overview paper, we describe these different approaches as well as the pros and cons of each of them. To this end, in *Section* "*Bayesian Networks and the Inference Problem*," some basic preliminaries on Bayesian networks will be explained. *Section "Probabilistic Hardware-Based Implementation of Bayesian Networks"* describes several probabilistic neuronal circuits for Bayesian network variables using different nonvolatile devices. We explain neural sampling machines (NSMs) for approximate Bayesian inference. In *Section "New Computing Architecture With Nonvolatile Memory Elements for Bayesian Network Implementation*," different systems for the implementation of Bayesian networks will be discussed that make use of new nonvolatile magnets and CMOS circuit elements. *Section "Bayesian Inference Hardware Implementation With Digital Logic Gates"* explains digital implementations of Bayesian inference algorithms as well as the definition of a standard cell-based implementation. At the end of this section, probabilistic nodes based on CMOS technology will be discussed. *Section "Crossbar Arrays for Bayesian Networks Implementation"* represents two brain-inspired hardware implementations of naïve Bayesian classifiers in the crossbar array architecture, in which memristors are employed as nonvolatile elements for algorithm implementation. Also, Bayesian reasoning machines with magneto-tunneling junction-based Bayesian networks are described. In *Section "Bayesian Neural Networks*," employing Bayesian features in neural networks is represented. First Bayesian neural networks are explained. Then, Gaussian synapses for probabilistic neural networks (PNNs) will be introduced. Afterward, PNN with memristive crossbar circuits is described. Approximate computing to provide hardware-friendly PNNs and an application of probabilistic artificial neural networks (ANNs) for analyzing transistor process variation are explained. In *Section "Hardware Implementation of Probabilistic Spiking Neural Networks*," employing Bayesian features in Spiking Neural Network (SNN) is represented. The feasibility of nonvolatile devices as synapses in SNNs architectures will be discussed for Bayesian-based inference algorithms. A scalable sampling-based probabilistic inference platform with spiking networks is explained. Then, a probabilistic spiking neural computing platform with MTJs is explained. Afterward, high learning capability of a probabilistic spiking neural network implementation and hardware implementation of SNNs utilizing probabilistic spike propagation mechanism are described. At the end of this section, memristor-based stochastic neurons for probabilistic SNN computing and Loihi based Bayesian inference implementation are represented. In *Section "Discussion*," we provide an overall discussion of the different approaches. Finally, *Section "Conclusion"* concludes the paper.

BAYESIAN NETWORKS AND THE INFERENCE PROBLEM

A discrete joint probability distribution defined over a set of random (or stochastic) variables assigns a probability to each joint value assignment to the set of variables; this representation, as well as any inference over it, is exponential in the number of variables. For most practical applications, however, there are many independences in the joint probability distribution that allows for a more concise representation. There are several possible ways to represent such independences in probabilistic graphical models, representing a probabilistic model with a graph structure (Korb and Nicholson, 2010). The commonly described graphical models are Hidden Markov Models (HMMs), Markov Random Fields (MRFs), and Bayesian networks. MRFs use undirected graphs to represent conditional independences and capture stochastic relations in potentials. Bayesian networks use directed a-cyclic graphs, capturing stochastic relations in conditional probability tables (CPTs). Both structures can represent different subsets of conditional independence relations. HMMs are dynamic Bayesian networks that efficiently model endogenous changes over time, under the assumption of the Markov property.

A simple Bayesian network with four variables (Pearl, 1988) has been shown as a running example in **Figure 1** in which Bayesian networks are represented by a directed acyclic graph composed of nodes and edges and a set of CPTs. The nodes in the graph model random variables, whereas the edges model direct dependencies among the variables.

The four binary variables (denoted True or False, or equally "1" or "0") "C," "R," "S," and "W" represent whether it is cloudy, it is rainy, the sprinkler is on, and the grass is wet, respectively. The conditional probabilities (given in the CPTs) describe the conditional dependencies between parent and child nodes. Based

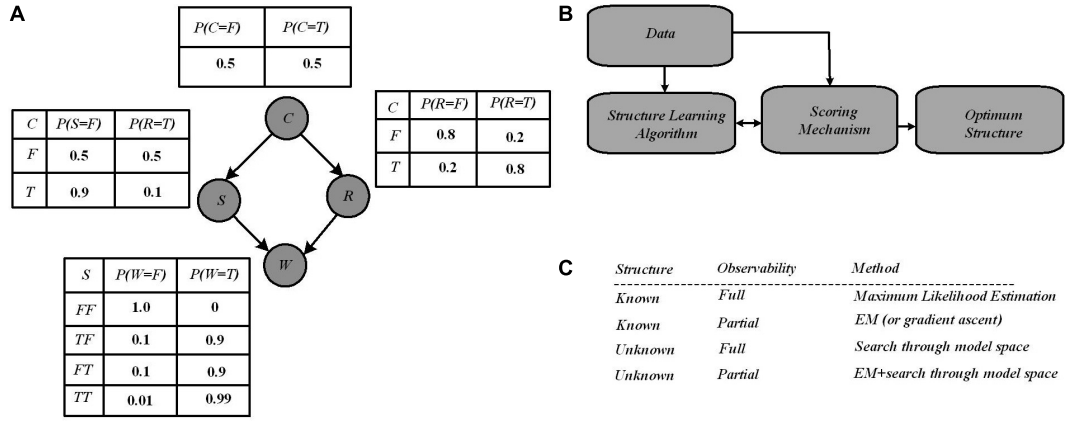


FIGURE 1 | (A) A Bayesian network with four variables where independence between variables has been reported *via* conditional probability tables (CPTs). All posterior probabilities of interest in this network can be computed using the laws of probability theory, notably Bayes' rule (1) that allows inferring the causes of effects observed in the network. **(B)** Structure learning flow in Bayesian networks through which Learning algorithm provides graphs from data, and are scored by the scoring mechanism. Finally, an optimum structure is selected after iteratively improvements of the score over the graph structure (Kulkarni et al., 2017b). **(C)** Several learning case in Bayesian networks (Murphy, 1998).

on the network structure, the inference operation estimates the probability of the hidden variables, based on the observed variables (Pearl, 1988). For example, suppose one observes that the grass is wet, then the inference operation seeks to compute the probability distribution over the possible causes. There are possibly two hidden causes for the grass being wet: if it is raining or the sprinkler is on. Bayes' rule defined in Equation (1), is used to calculate the posterior probability of each cause when wet grass has been observed; it allows us to compute this distribution from the parameters available in the CPTs:

$$P(S|W) = \frac{P(W|S)P(S)}{P(W)} \tag{1}$$

For data analysis, the graphical model provides several benefits. Different methods are utilized for data analysis, which are rule bases, decision trees, and ANNs. Different techniques for data analysis are density estimation, classification, regression, and clustering. Then, what do Bayesian methods provide? One, it readily handles the missing of some data entries since the model encodes dependencies among all variables. Two, a Bayesian network paves the way to understanding about a problem domain and predicting the consequences of intervention *via* learning the causal relationships. Three, the model provides a causal and probabilistic semantics, though which an ideal representation for combining prior knowledge and data is possible. Four, with Bayesian statistics as well as Bayesian networks, the overfitting of data can be solved (Heckerman, 2020).

Learning a Bayesian network has two major aspects, i.e., discovering the optimal structure of the graph and learning the parameters in the CPTs. Learning a Bayesian network from data requires two steps of structure learning and parameter learning. There are a few works focusing on hardware implementation for structure learning. In order to find an optimal structure, exploring all possible graph structures for a given dataset is necessary. As shown in **Figure 1B**, for structure learning, based on the data, an algorithm starts with a random graph, then a scoring mechanism determines how well the structure can explain the data, where this quality is typically a mix of simplicity and likelihood. The graph structure is updated based on the score, and as a graph provides a better score, it is accepted. Several algorithms have been proposed in the literature for structure learning, with the two major scoring mechanisms being Bayesian scoring and information-theoretic scoring (Kulkarni et al., 2017b). Most of the information-theoretic scoring methods are analytical, and then complex mathematical computations are required. These methods are currently performed by software and the required time for structure learning is impacted significantly. Equation (2) represents the Bayesian scoring that uses the Bayes' rule to compute the quality of a given Bayesian network structure. Using the Bayes rule, for a given data, and for a structure, the Bayes score is defined by:

$$\mathbf{P(structure|Data)} \propto \mathbf{P(Data|structure)} \times \mathbf{P(Data)} \tag{2}$$

The score of a structure as shown by Equation (2) is proportional to how closely it can describe observed data and on the prior probability of the structure (which could be uniform or provided by a domain expert). The Bayesian score is calculated *via* stochastic sampling through which a model of the graph is generated with the CPT values set, and sampling over each node for several iterations is performed. For example, for a probability value of 0.5 for a node, with 10 sampling iterations, it is expected to show "True" in 5 iterations. To calculate the Bayesian score of the graph (i.e., defining the correlation degree between the sampled data and learning data), the inference data taken from the stochastic sampling process are utilized. Once the structure of the network has been learned from the data, parameter learning (i.e., using data to learn the distributions of a Bayesian network) can be performed efficiently by estimating

the parameters of the local distributions implied by the structure obtained in the previous step. There are two main approaches to the estimation of those parameters in literature: one based on maximum likelihood estimation and the other based on Bayesian estimation (Heckerman, 2020). Parameter estimation still could be challenging when the sample sizes are much smaller than the number of variables in the model. This situation is called "small n, large p," which brings a high variability unless particular care is taken in both structure and parameter learning. As mentioned above, the graph topology (structure) and the parameters of each CPT can be inferred from data. However, learning structure is in general much harder than learning parameters. Also, learning when some of the nodes are hidden, or in case data are missing, is harder than when everything is observed. This gives rise to four distinct cases with increasing difficulty shown in **Figure 1C**.

Bayesian network learning involves the development of both the structure and the parameters of Bayesian networks from observational and interventional datasets; Bayesian inference on the other hand is often a follow-up to Bayesian network learning and deals with inferring the state of a set of variables given the state of others as evidence. The computation of the posterior probabilities shown above (**Figure 1A**) is a fundamental problem in the evaluation of queries. This allows for diagnosis (computing P(cause| symptoms)), prediction (computing P(symptoms| cause)), classification (computing P(class| data)), and decision-making when a cost function is involved. In summary, Bayesian networks allow for a very rich and structured representation of dependencies and independencies within a joint probability distribution. This comes at the price of the intractability of both inference (i.e., the computation of posterior probabilities conditioned on some observations in the network) and learning (i.e., the establishment of the structure of the model and/or the conditional probabilities based on data and a learning algorithm). One can deal with this intractability either by reducing the complexity of the model or by accepting approximate results. Examples of the former are reducing the tree width of the network model (Kwisthout et al., 2010), reducing the structure of the model to a polytree describing a hidden state model and observable sensors (Hidden Markov model) (Baum and Petrie, 1966), or assuming mutual independence between features (Naïve Bayesian classifiers) (Maron and Kuhns, 1960). Examples of the latter are approximation algorithms such as Metropolis-Hastings (Hastings, 1970) and Likelihood weighting (Shachter and Peot, 1990).

PROBABILISTIC HARDWARE-BASED IMPLEMENTATION OF BAYESIAN NETWORKS

This section represents several probabilistic neuron circuits for Bayesian network variables by using different nonvolatile devices connected to CMOS circuit elements. To this end, the first two abstraction layer-based implementations and then a direct implementation of probabilistic circuits will be discussed. Then, a NSM for approximate Bayesian inference is explained.

Probabilistic Spin Logic-Based Implementation of Bayesian Networks

The first approach we will discuss is the mapping of CPTs to probabilistic circuits constructed by probabilistic bits (p-bits) (Faria et al., 2018; Debashis et al., 2020). In this approach, each variable in a Bayesian network is modeled by a stochastic circuit, representing a specific conditional probability. This probability is represented by the input that comes from its parent nodes, *via* the weights of the links between nodes. For the p-bit implementation, the Bayesian network is translated into probabilistic spin logic (PSL). PSL is a behavioral model, represented by biasing (h) and interconnection (J) coefficients (shown in **Figure 2A**). Then, PSL is translated into electronic devices.

The reported p-bit implementation in Faria et al. (2018) as shown in **Figure 2B** uses a stochastic spintronic device, i.e., magnetic tunnel junction (MTJ), connected to the drain of a transistor.

Table 1 reports the required equations for the PSL translation into a circuit whose output m_1 is related to its input I_2 (the synapse generates the input I_2 from a weighted state of m_2, **Figure 2A**), Equation (3). Based on Equation (4A), a random number generator (RNG) (rand) and a tunable element (tanh) construct m_2. The RNG is the MTJ and the tunable component is the NMOS transistor; r_{MTJ} is a correlated RNG and the NMOS transistor resistance r_{T} is approximated as a tanh function found by fitting based on I–V characteristics. The PSL model is then translated into electronic components (shown in **Figure 2B**) where each node (represented by m) is connected to other nodes and biased through voltages V_{bias} and conductances G; V_0 is a fitting parameter. Biasing (h) and interconnection (J) coefficients of PSL model have been reported in **Table 1** by Equations (5A)–(7A), due to its corresponding P-bit circuit in **Figure 2B**.

Note that individual p-bits require sequencers in software implementations to be programmed in a directed order. The p-bit in Faria et al. (2018) and Faria et al. (2021) is an autonomous, asynchronous circuit that can operate correctly without any clocks or sequencers, in which the individual p-bits need to be carefully designed and the interconnect delays, from one node to another node, must be much shorter than the nanomagnet fluctuations of the stochastic device. This condition is met as magnetic fluctuations occur at approximately the 1-ns time scale. However, in asynchronous operations, updating the network as well as dealing with variations in the thermal barriers or interconnect delays necessitates further study.

Debashis et al. (2020) present another alternate p-bit implemented with inherently stochastic spintronic devices based on a nanomagnet with perpendicular magnetic anisotropy. This device utilizes the spin orbit torque from a heavy metal (HM) under-layer to be initialized to its hard axes. Equations (4B)–(7B) in **Table 1** show the relation between the stochastic variables m_1 and m_2 based on the corresponding p-bit circuit in **Figure 2C**. Here, σ defines the sigmoidal activation function for the device in m_2. Equation (4B) explains the conditional dependencies. The probability of m_2 being 1 given m_1 being 1 is calculated through Equation (4B) while setting $m_1 = 1$. The parameters B_0 and h_2 represented in Equations (5B)–(7B) can be tuned to change

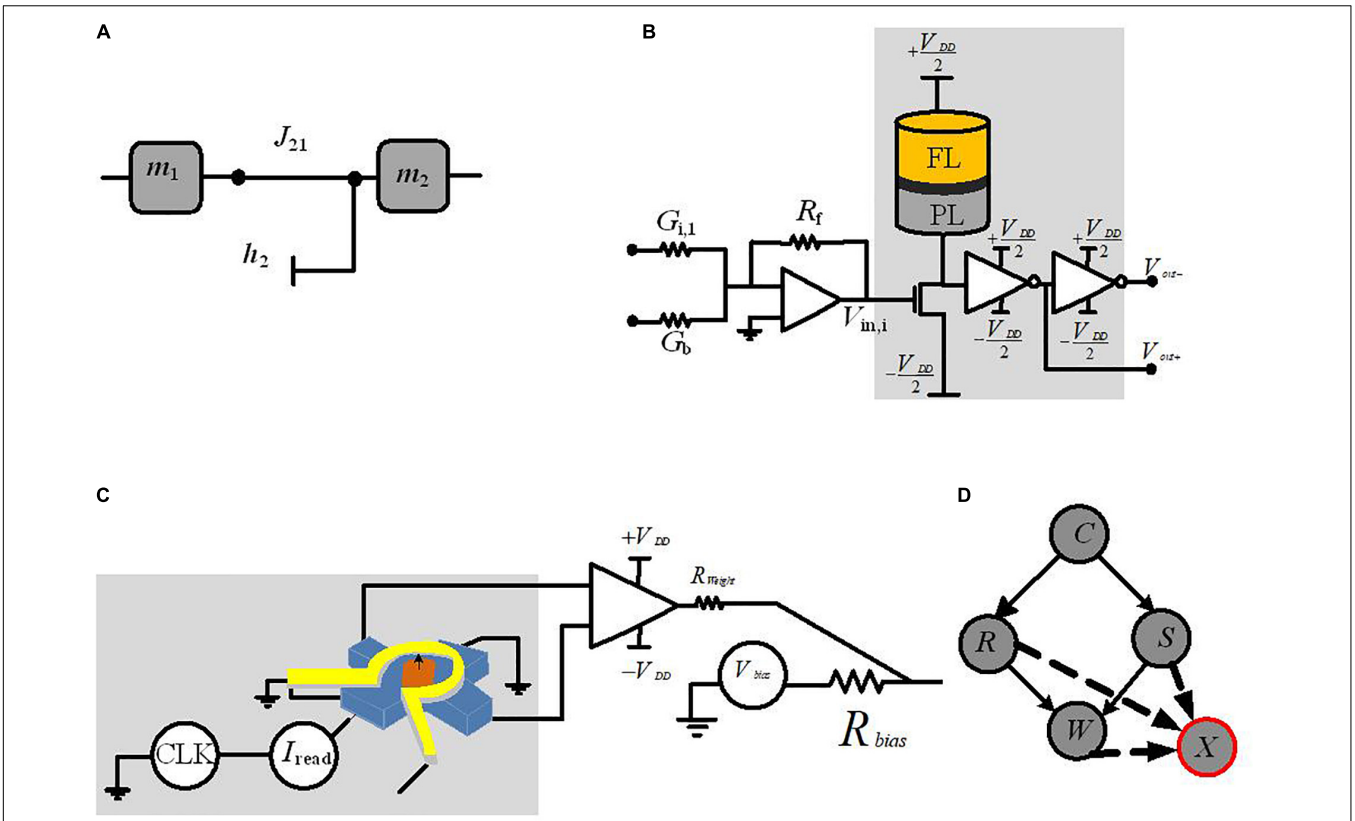


FIGURE 2 | Circuit implementation of a p-bit block. **(A)** PSL-based representation of two-node Bayesian network. **(B)** The p-bit design based on MTJ p-circuit with connection weight and bias to be connected to another node (Faria et al., 2018). **(C)** The p-bit design, based on nanomagnet p-circuit with connection weight and bias to be connected to another node (Debashis et al., 2020). **(D)** The required auxiliary node, *X*, for representing the four-node Bayesian network.

TABLE 1 | PSL to circuit translation requirements.

PSL elements	***P-bit design-1*** **(Faria et al., 2018)**	***P-bit design-2*** **(Debashis et al., 2020)**
PSL	$I_2 = J_{21}m_1 + h_2$ (3)	
Given CPT: $m_1 = 0, p(m_2 = 1) = a$ $m_1 = 1, p(m_2 = 1) = b$	$m_2(t + \Delta t) = sgn(-r_{MTJ}(t + \Delta t) + r_T(t + \Delta t))$ (4A)	$m_2 = \sigma(I_2) = \sigma(J_{21}m_1 + h_2)$ (4B)
J_{21}	$J_{21} = \frac{G_{21}}{G_b}$ (5A)	$J_{21} = \pm\mu_0\frac{V_{DD}}{2}B_0R_{weight}$ (5B)
h_2	$h_2 = \frac{V_{bias,2}}{\frac{V_{DD}}{2}}$, (6A)	$h_2 = \pm\mu_0\frac{V_{bias}}{2}B_0R_{bias}$ (6B)
I_2	$I_2 = \frac{V_{in,2}}{V_0}$ (7A)	$I_2 = \pm\left(\mu_0\frac{V_{DD}}{2}B_0R_{weight}\right)m_1 + (\mu_0\frac{V_{bias}}{2}B_0R_{bias})$ (7B)

the shape and offset of the sigmoidal activation function (while presenting the CPTs *via* the connection weights).

To implement the four-node Bayesian network by p-bits, **Figure 2D**, using PSL behavioral models in Faria et al. (2018) and Debashis et al. (2020), requires an auxiliary p-bit defined by node "*X*." The CPT of node "*W*" has four conditional probability distributions (based on nodes "*R*" and "*S*," see **Figure 1**); based on the principles of linear algebra, this CPT needs four independent parameters. The interconnection weights J_{WR} and J_{WS} and the bias parameter to the node "*W*" (h_W) are three parameters of four. The fourth parameter has been implemented with the interconnection to node "*X*." Nodes with *N* parents need a total of $(N+1)$ parameters and $2N$ equations to meet the PSL model requirement. Based on the number of linearly independent equations, it is needed to represent the appropriate number of auxiliary variables (Faria et al., 2018). Utilizing the auxiliary nodes in p-bit-based implementation of Bayesian networks adds extra area/energy overhead and requires further studies.

Spintronic Devices for Direct Hardware Implementation of Bayesian Networks

In Shim et al. (2017), a direct implementation of Bayesian networks has been proposed with a stochastic device that is based

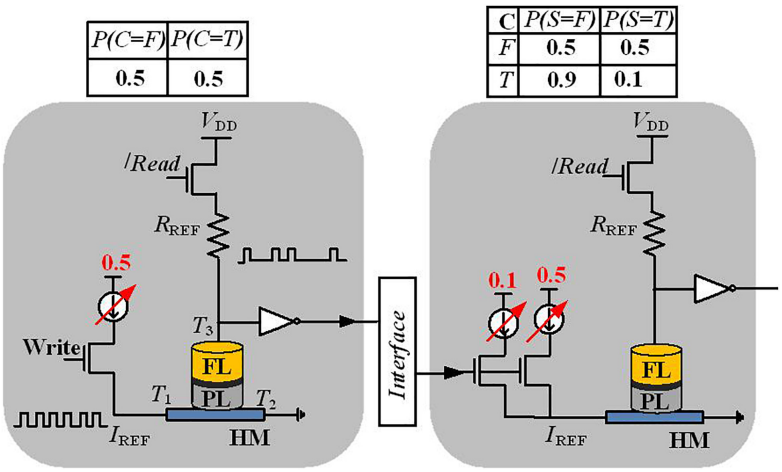


FIGURE 3 | Detailed implementation view of a Bayesian network variable and the interconnection between the two nodes C and S in the four-node Bayesian network of **Figure 1**.

on a three-terminal device structure, shown in **Figure 3**. The proposed stochastic device can be developed by fabricating an MTJ stacked on top of the ferromagnet-HM layers. The stochastic switching of the device in the presence of thermal noise has been employed to implement a Bayesian network. This MTJ with two permanent states (represent two different resistance levels) models stored values by the resistance levels (Shim et al., 2017). The MTJ is composed of a tunneling barrier (TB) sandwiched between two ferromagnetic layers, namely, the free layer (FL) and the pinned layer (PL). The relative magnetization direction of two ferromagnetic layers defines the MTJ state; MTJ shows low (or high) resistance when the relative magnetization direction is parallel (or anti-parallel) (Bagheriye et al., 2018). Based on the write current through terminals T_1 and T_2, which probabilistically switches the magnet (with a probability controlled by the current magnitude), the read path through the terminals T_3 and T_2 controls the final state of the magnet.

In order to represent a variable of the Bayesian network, a Poisson pulse train generator translates the probability data into the frequency of the output pulses. Thanks to the controllable stochastic switching of the nanomagnet, along with current sources and some circuit elements [reference resistor (R_{ref}) and separate write and read paths], the Poisson spikes can be generated as shown in **Figure 3**. A reference resistor is used to generate a Poisson spike train, where the number of spikes encodes information about the probability. For instance, if 30 spikes are observed at the output of the "S" node in 100 write cycles, this determines that the probability of "S is True" is 30%. Moreover, for more complex inference, extra arithmetic building blocks using CMOS circuits between two Poisson pulses are needed.

Neural Sampling Machine for Approximate Bayesian Inference

In biological neural networks, synaptic stochasticity occurs at the molecular level, and due to the presynaptic neuronal spike, the neurotransmitters at the synaptic release site release with a probability of approximately 0.1. Dutta et al. (2021) presented a neuromorphic hardware framework to support a recently proposed class of stochastic neural networks called the neural sampling machine (NSM), which mimics the dynamics of noisy biological synapses. NSM incorporates a Bernoulli or "blank-out" noise in the synapse to being multiplicative, which has an important role in learning and probabilistic inference dynamics. This performs as a continuous DropConnect mask on the synaptic weights, where a subset of the weights is continuously forced to be zero. Stochasticity is switched off during inferencing in DropConnect, whereas it is always on in an NSM providing probabilistic inference capabilities to the network. **Figure 4** shows the hardware implementation of NSM using hybrid stochastic synapses. These synapses consist of an embedded non-volatile memory, eNVM [a doped HfO2 ferroelectric field-effect transistor (FeFET)-based analog weight cell] in series with a two-terminal Ag/HfO2 stochastic selector element. By changing the inherent stochastic switching of the selector element between the insulator and the metallic state, the Bernoulli sampling of the conductance states of the FeFET can be performed. Moreover, the multiplicative noise dynamics has a key side effect of self-normalizing, which performs automatic weight normalization and prevention of internal covariate shift in an online fashion. The conductance states of the eNVM in the crossbar array (which performs row-wise weight update and column-wise summation operations in a parallel fashion) are adapted by weights in the Deep Neural Network (DNN). In order to implement an NSM with the same existing hardware architecture, selectively sampling or reading the synaptic weights Gij with some degree of uncertainty is required. A selector device as a switch has been employed, stochastically switching between an ON state (representing $\xi ij = 1$, ξijgenerated for each of the synapse and is a random binary variable) and an OFF state ($\xi ij = 0$). **Figure 4B** depicts an input voltage V_{in3} applied to a row of the synaptic array with conductance states $G = \{G_1, G_2, G_3, G_4, \ldots, G_N\}$, and based on the state of the selectors in the cross-points, an

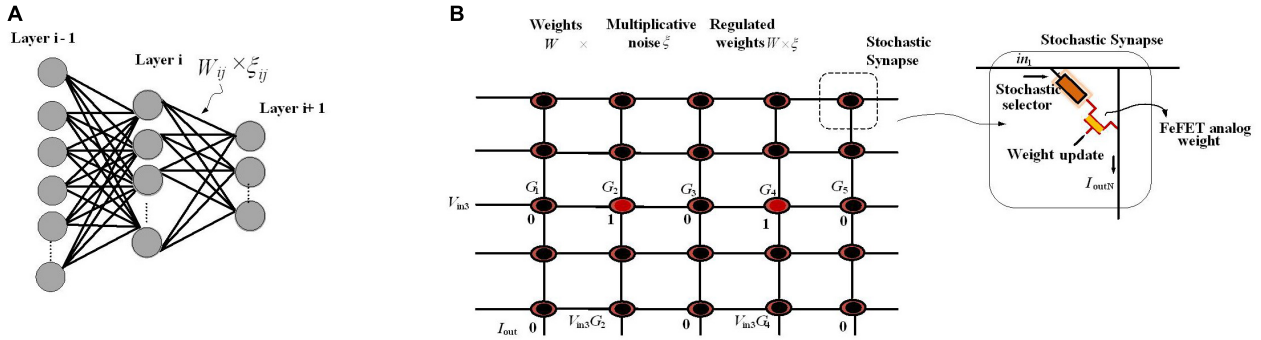


FIGURE 4 | An NSM implemented in hardware using crossbar array architecture. **(A)** The utilized NSM with an input layer, three hidden layers, and an output layer. **(B)** The stochastic selector device used for injecting Bernoulli multiplicative noise is placed at each cross-point connected to an analog weight cell implemented using eNVMs. The stochastic selector element provides the selectively sampling or reading the synaptic weights G_{ij} with some degree of uncertainty controlled by random binary variables ξ_{ij}.

output weighted sum current I_{out} = {0, $G_2.V_{in3}$, 0, $G_4.V_{in3}$, 0} is obtained, which is exactly the same as multiplying the weight sum of $W_{ij}Z_j$ (Z_j, is the activation function of the neuron j) with a multiplicative noise ξ_{ij}.

For Bayesian inference, the hardware NSM captures uncertainty in data and produces classification confidence. To this end, in Dutta et al. (2021), the hardware NSM has been trained on the full MNIST dataset. During the inference mode, the performance of the trained NSM on continuously rotated images has been evaluated where, for each of the rotated images, 100 stochastic forward passes are performed and the softmax input (output of the last fully connected hidden layer in **Figure 4A**) as well the softmax output were recorded. The NSM will correctly predict the class corresponding to an input neuron if the softmax input of a particular neuron is larger than all the other neurons. However, as the images are rotated more, even though the softmax output can be arbitrarily high for, e.g., neuron 2 or 4 predicting that the image are 2 or 4, respectively, the uncertainty in the softmax output is high, showing that the NSM can account for the uncertainty in the prediction. The uncertainty of the NSM has been quantified by looking at the entropy of the prediction, defined as $H = -\sum_P * log_{(P)}$, where p is the probability distribution of the prediction. When the NSM makes a correct prediction, the uncertainty measured in terms of the entropy remains 0. However, in the case of wrong predictions, the uncertainty associated with the prediction becomes large. This is in contrast to the results obtained from a conventional MLP network (deterministic neural network) where the network cannot account for any uncertainty in the data.

NEW COMPUTING ARCHITECTURE WITH NONVOLATILE MEMORY ELEMENTS FOR BAYESIAN NETWORK IMPLEMENTATION

In this section, several Bayesian network implementation systems will be discussed that make use of new nonvolatile magnets and CMOS circuit elements. We will first explain FPGA-like architectures and then discuss developed spintronic-based inference systems.

Direct Physical Equivalence Implementation of Bayesian Networks

In Khasanvis et al. (2015a), in addition to transistors, strain-switched magneto tunneling junctions (S-MTJs) are used for a Bayesian hardware implementation. S-MTJs as nonvolatile devices provide low switching energy (Atulasimha and Bandyopadhyay, 2013). As shown in **Figure 5A**, it has four terminals and the resistance between reference and output terminals can be changed by the two input digital voltage terminals change. It shows hysteresis in resistance vs. voltage characteristics that provides non-volatility. Khasanvis et al. (2015a) represents a mindset of physical equivalence, which means each digit in the probability representation is mapped directly (without any abstraction layer) to S-MTJ resistance with equivalent digital voltage representation (**Figure 5B**), while the proposed work in Faria et al. (2018) and Debashis et al. (2020) need the PSL abstraction level to map Bayesian networks in hardware.

For encoding, n spatially distributed digits $p_1, p_2, \ldots, p_n$ have been defined (**Figure 5B**), each digit p_i can be any one of k values, the number of states of the physical device (e.g., for devices with two states, $k = 2$ and $p_i \in \{0, 1\}$). The encoded probability P is defined by: $P = \frac{\sum_{i=1}^{n} p_i}{n(k-1)}$, which is called a flat linear representation (resolution is determined by the number of digits n). These digits have been physically represented in resistance domains using two-state S-MTJs, where R_{low} represents digit 1 and R_{high} represents digit 0. For Bayesian computations in hardware, it is necessary to have analog arithmetic functions such as probability addition and multiplication. **Figures 5C–E** depict arithmetic composers, which are operating intrinsically on probabilities as elementary building blocks. **Figures 5C–E** show the addition composer, multiplication composer, and add–multiply operation composer, respectively, as well as support circuits such as amplifiers, implemented with CMOS operational amplifiers.

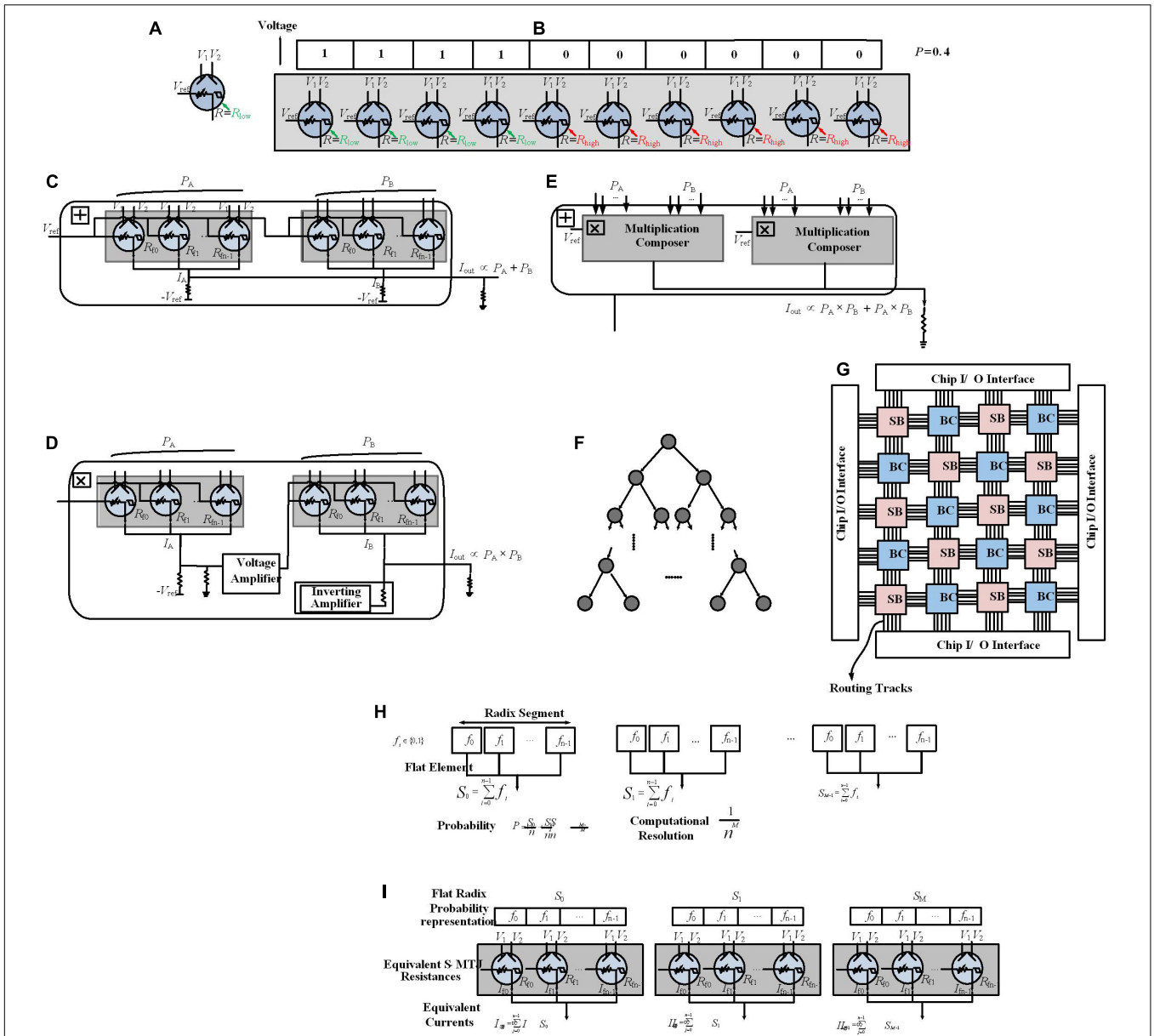


FIGURE 5 | Probability encoding *via* Strain-switched Magneto-Tunneling Junction (S-MTJ) device. **(A)** A S-MTJ Device. **(B)** Probability data encoding by spatially distributed digits, and two-state S-MTJs for physically equivalent representations. Probability value $P = 0.4$ has been encoded with 10 digits (resolution of 0.1). Probability composer framework. **(C)** Addition composer. **(D)** Multiplication composer. **(E)** Add–Multiply composer. Physically Equivalent Architecture for Reasoning (PEAR). **(F)** Tree Bayesian Network. **(G)** Mapping every node in a Bayesian network graph to a Bayesian cell (BC) on PEAR. **(H)** Flat-Radix information representation pattern. Probability is encoded in segments where each segment has a radix arrangement and contains flat elements. **(I)** Probability data encoding by the proposed information representation scheme, and S-MTJs for physically equivalent representations.

In Khasanvis et al. (2015a), thanks to the analog arithmetic composers, a paradigm departure from the Von Neumann paradigm has been developed that uses a distributed Bayesian cell (BC) architecture. In this architecture, each BC maps a Bayesian variable in hardware as physical equivalence, shown in **Figures 5F,G**, named Physically Equivalent Architecture for Reasoning (PEAR). BCs are constructed from probability arithmetic composers and are used to include CPTs, likelihood vectors, belief vectors, and prior vectors; BCs locally store these values continuously and perform inference operation, removing the need for external memory (Khasanvis et al., 2015a). Metal routing layers are used for BC interconnection. This connectivity is programmable through reconfigurable switch boxes (SBs) (similar to FPGAs) to map arbitrary graph structures.

Bayesian networks using binary trees, as shown in **Figures 5F,G**, have been mapped directly in hardware on PEAR. This computing architecture scales the number of variables to a million. Although for a resolution of 0.1, it gains

three orders of magnitude efficiency improvement in terms of runtime, power, and area over implementation on 100-core microprocessors, it does not support efficient scaling for higher resolutions. To increase the resolution, it is needed to change the abovementioned flat linear representation that increases area linearly (where a single probability value requires multiple physical signals). To this end, as shown in **Figures 5H,I**, another S-MTJ-based circuit paradigm leveraging physical equivalence with a new approximate circuit-style has been reported (Kulkarni et al., 2017a), where the computation resolution is $1/(n^M)$ where M is the number of Radix segments where each segment is composed of flat elements. This is a new direction on scaling computational resolution, which is a hybrid method for representing probabilities, aiming to provide networks with millions of random variables. Here, precision scaling provides much lower power and performance cost than in Khasanvis et al. (2015b) for PEAR implementation *via* offering area overhead at a logarithmic vs. linear scale. Results show a 30× area reduction for a 0.001 precision vs. prior direction (Khasanvis et al., 2015a) while obtaining three orders of magnitude benefits over 100-core processor implementations.

Stochastic Hardware Frameworks for Learning Bayesian Network Structure

A Bayesian network has two major aspects: the structure of the graph and the parameters in the CPT and determining the structure of a Bayesian network is known as structure learning. In Kulkarni et al. (2017b), the stochastic behavior of emerging magneto-electric devices is used to accelerate the structure learning process of Bayesian learning, which results in reducing the runtime by five orders of magnitude for Bayesian inference. For structure learning, based on the data, an algorithm starts with a random graph, then a scoring mechanism determines how well the structure can explain the data, where this quality is typically a mix of simplicity and likelihood. The graph structure is updated based on the score; as a graph provides a better score, it is accepted. To perform scoring, the framework should support mapping of arbitrary Bayesian networks; hence, configurability is necessary. The proposed design employs an FPGA-like reconfigurable architecture constructed from a set of programmable SBs and Stochastic Bayesian Nodes (SBNs). For scoring a Bayesian structure, nodes are mapped into SBN and the connectivity between nodes is implemented by SBs (**Figures 6A,B**).

Stochastic Bayesian Nodes represents a node in a Bayesian network. The node consists of multiplexers, a digital pulse width modulator (DPWM), and perpendicular magnetic anisotropy spin transfer torque magnetic tunnel junctions (PMA-STT MTJs). The switching operation of PMA-STT MTJ is probabilistic and directly controlled *via* modulating the duration of the applied current; this unique property has been employed to design circuits to perform probabilistic operations. As shown in **Figure 6A**, the CPT values are preconfigured in the SRAM cell. An appropriate CPT value to be sent to the DPWM is selected by multiplexers based on the output of the parent SBN. A DPWM generates voltage pulses with precise duration. Once the pulse corresponding to the CPT value is fed to the MTJ, the output is stored in a flip-flop. The output of the flip-flop is available for read-out and is also sent to the next node. The configured Bayesian structure is stochastically sampled to reach sufficient statistics. The sampled data are employed to calculate the Bayesian score of that structure, through Equation (2).

Through this hardware acceleration of the structure discovery (*via* scoring mechanism) process of Bayesian learning, the runtime for Bayesian network inference has been highly reduced (Kulkarni et al., 2017b). This property attracts more attention to structure learning acceleration and turns out to be a promising field to be studied.

Stochastic Bitstream Generator Blocks for Bayesian Network Implementation

In Jia et al. (2018), the inherent stochastic behavior of spintronic device, MTJs, has been used to build a stochastic bitstream generator (SBG), which is critical for the Bayesian inference system (BIS).

Figures 7A,B describe the diagram of the proposed BIS in Jia et al. (2018). A SBG block consists of a RNG and a comparator, which together generate the corresponding bit stream (**Figure 7A**). The input of BIS is shown in **Figure 7B**, which is a series of bias voltages proportional to evidence or likelihood. These evidences or likelihoods may come from sensors of different platforms. The SBG matrix and the SC architecture are two key components of a BIS. The SBG matrix is employed to translate the input voltages to stochastic bitstreams. The stochastic computing architecture is constructed by simple logic gates such as AND gate and scaled addition implemented by a multiplexer (MUX) and takes SBs as inputs. Stochastic computing block implements Bayesian inference by a novel arrangement of logic gates.

For an SBG, the small margin input voltages (Jia et al., 2018) is highly problematic when it generates the output probability. Digital-to-analog converters (DACs) with high precision are needed for precise mapping from digital probabilities to voltages. In addition, tackling the nonlinear relationship between probabilities and voltages is difficult and a slight noise or process variation may translate a probability to a wrong voltage value. In order to address these limitations, for the specified applications a prebuild SBG array utilizing SBG sharing strategy is employed. It is implemented by hybrid CMOS/MTJ technologies named spintronic-based Bayesian inference system (SPINBIS) (**Figure 7C**). The aim of proposing the SPINBIS is to enhance the stability of SBG and to use a smaller number of SBGs (Jia et al., 2020). The outcome probability of each SBG is predetermined and is then multiplexed through the switch matrix, which is a crossbar array. This crossbar array is constructed from transistors implemented at cross points, which are controlled by the switch controller. Since the SBG array is prebuilt, it should provide enough kinds of bitstreams to have an accurate stochastic computing block. In order to improve the energy efficiency and speed of SBG circuit, a state-aware self-control mechanism is utilized. Thanks to the SBG sharing property, the inputs with the same evidence can be modeled by the bitstream of the same

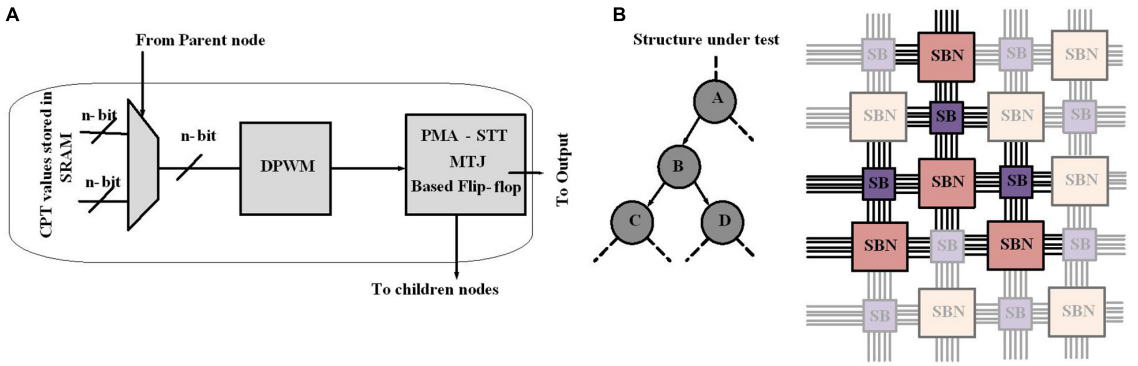


FIGURE 6 | Design flow for structure learning in Bayesian networks. **(A)** Building blocks of Stochastic Bayesian Nodes (SBN). **(B)** Every node in a Bayesian network is mapped to an SBN reconfigurable framework. The links of the Bayesian network are implemented with metal routing layers and the connectivity programmed through switch boxes. Stochastic bitstream generator blocks for Bayesian network implementation.

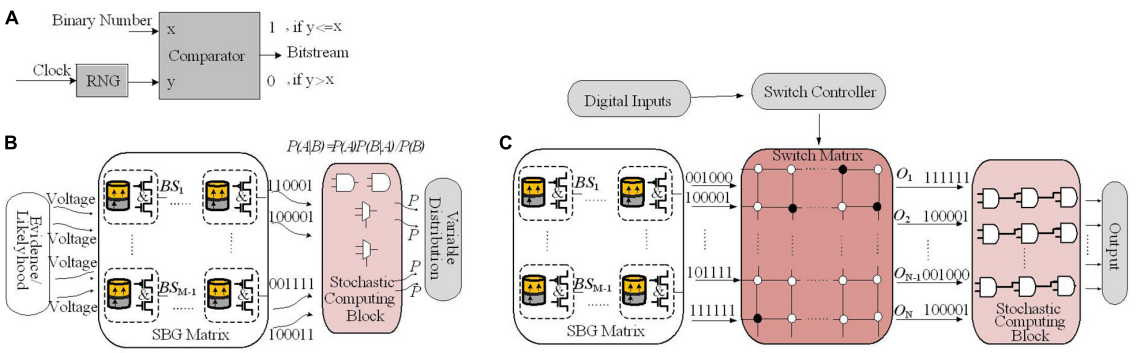


FIGURE 7 | (A) Stochastic bitstream generator (SBG) block design. **(B)** Stochastic Bayesian inference system. **(C)** Spintronic-based Bayesian Inference System (SPINBIS) diagram work in this figure (switch matrix).

SBG. However, for the inputs that are related together by one or more logic gates, which are called conflicting inputs, sharing the same SBG is problematic and is not allowed. The SBG sharing mechanism provides a much smaller number of SBGs compared with the input terminals of stochastic computing logic since the non-conflicting terminals are allowed to share the same bitstream. For data fusion applications, SPINBIS provides 12× less energy consumption compared to the MTJ-based approach (Jia et al., 2018) with 45% area overhead and about 26× compared to the FPGA-based implementation. On the other hand, the relation between probability and voltage is not very smooth; as a result, the stability of the proposed SBG needs improvement. Although the scale can be reduced, the switch matrix can show a congestion problem; hence, the reduction of the scale of SPINBIS is also worth exploring.

BAYESIAN INFERENCE HARDWARE IMPLEMENTATION WITH DIGITAL LOGIC GATES

In this section, digital implementation of Bayesian inference will be discussed. First, we describe an implementation of Bayesian inference on HMM structures in digital logic gates. Next, an approximate inference algorithm based on a novel abstraction defined by stochastic logic circuits and some other hardware implementations of MRFs will be explained. Then, we describe C-Muller circuits as implemented with standard cells for Bayesian inference. Finally, we discuss probabilistic nodes based on CMOS technology. Hardware implementation of Bayesian inference employs the HMM network.

Hardware Implementation of Bayesian Inference Employing Hidden Markov Model Network

In Thakur et al. (2016), a hardware implementation of an HMM network has been proposed that utilizes sequential Monte Carlo (SMC) in SNNs. An HMM shown in **Figure 8A** models a system defined by a process that generates an observable sequence depending on the underlying process (Yu et al., 2018). In an HMM, X_t and Y_t represent the signal process and the observation, respectively. In a first order HMM, Y_t, is considered as a noisy function of X_t and the development of a hidden state depends only on its current state. X_t is computed by its posterior distribution based on the noisy measurements or Y_t.

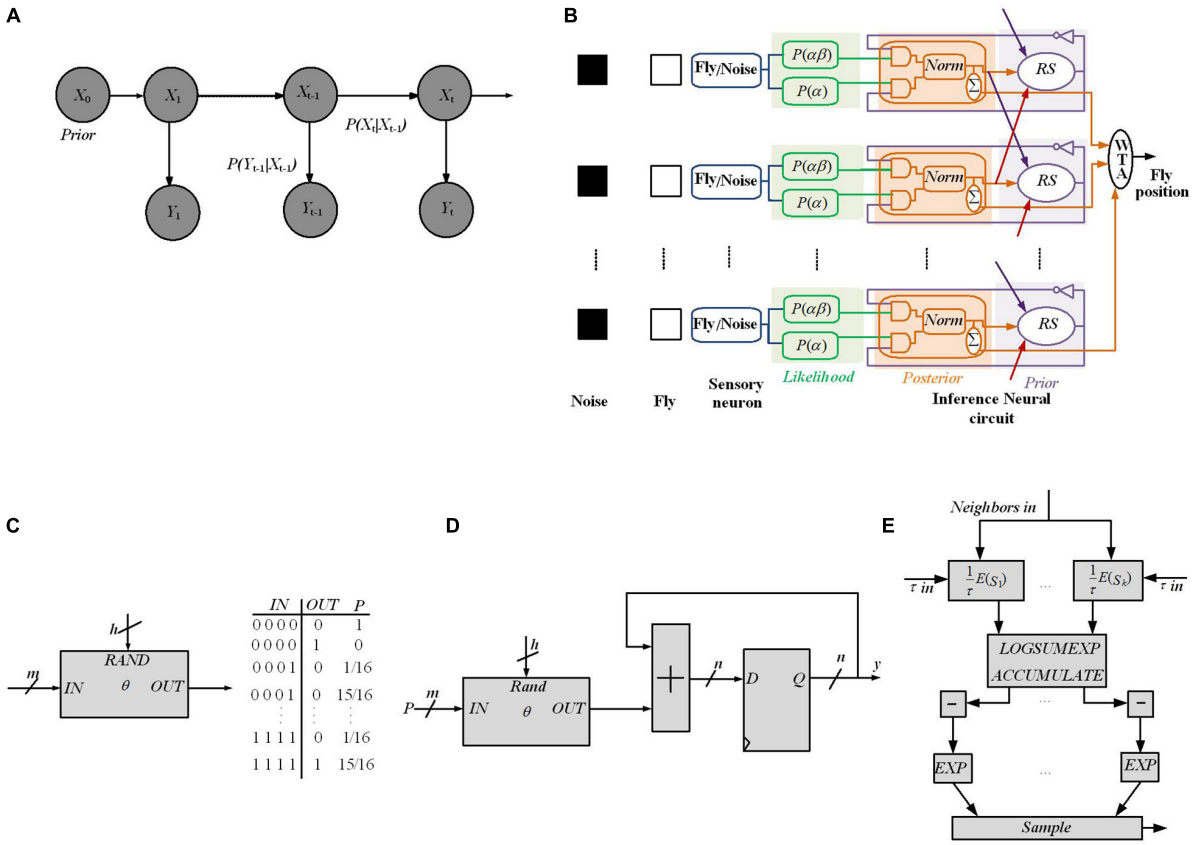


FIGURE 8 | FPGA implementation of different Bayesian networks **(A)** An HMM architecture, in which the random variables X_t and Y_t are the hidden state at time *t* and the observation at time *t*, respectively (conditional dependencies are shown by arrows). **(B)** Digital Hardware implementation of HMM algorithm for Bayesian inference in Thakur et al. (2016). Implement the Gibbs sampling algorithms for Markov random fields in Mansinghka et al. (2008). **(C)** The schematic and CPT for a *O* gate, which flips a coin by specifying the weight on *IN* as a binary number [e.g., for *IN* = 0111, *P*(*OUT* = 1| *IN*) = 7/16]. A comparator that outputs 1 if RAND ≤ *IN* has been employed for implementing the *O* gate. **(D)** The proposed circuit for sampling, for a binomial distribution, utilizes *nh* bits of entropy to perform sampling while considering *n* flips of a coin of weight. **(E)** Gibbs pipeline, for Gibbs sampler depicting the required operations to numerically sample an arbitrary-size variable.

For a discrete-time problem, Equation (8) defines the first-order HMM, in which d_t and v_t denote random noise sequences.

$$\boldsymbol{X_t} = \boldsymbol{f}(\boldsymbol{X_{t-1}}, \boldsymbol{d_t}), \boldsymbol{Y_t} = \boldsymbol{g}(\boldsymbol{X_t}, \boldsymbol{v_t}) \tag{8}$$

The posterior density function $P(X_t|\ Y_{1:t})$ is computed recursively in two steps (i) prediction, and (ii) update. In the prediction step, the next state is estimated based on the current state utilizing the state transition model, without making use of new observations [see Equation (9)]. In contrast, the predicted state is updated utilizing the new observations at time *t* as shown in Equation (10).

$$\boldsymbol{P(X_t|Y_{1:t\text{-}1})} = \sum_{\boldsymbol{X_{t\text{-}1}}} \boldsymbol{P(X_t|X_{t\text{-}1})P(X_{t\text{-}1}|Y_{1:t\text{-}1})} \tag{9}$$

$$\boldsymbol{P(X_t|Y_{1:t})} = \frac{\boldsymbol{P(Y_t|X_t)P(X_t|Y_{1:t\text{-}1})}}{\sum_{X_t} \boldsymbol{P(Y_t|X_t)P(X_t|Y_{1:t-1})}} \tag{10}$$

In Thakur et al. (2016), to estimate a fly's position at time t, a digital framework working based on the HMM rule shown in **Figure 8B** is utilized, through which a dragonfly tracks a fruit fly in a randomly flickering background. The sensory afferent neurons of the dragonfly fire probabilistically, when there is a fruit fiy or a false target (noise).

Dividing the state space (X_t) into M discrete states reflects the fly's (discretized) position at time t. A sensory neuron and an inference neuronal circuit demonstrate each discrete state. The fruit fly's position at time t is predicted by the dragonfly's central nervous system through utilizing the statistics of the output spikes of the sensory afferent neurons until time (t-1), and updates the prediction when it receives a new observation (Y_t), at time t. Utilizing prediction and update Equations (9) and (10), it can be written as:

$$P(X_t \mid Y_{1:t}) \propto P(Y_t \mid X_t) \sum_{X_{t-1}} P(X_t \mid X_{t-1}) P(X_{t-1} \mid Y_{1:t-1}) \tag{11}$$

where $P(Y_t \mid X_t)$ is the likelihood, $P(X_t \mid X_{t-1})$ is the transition probability, and $P(X_{t-1}|\ Y_{1:t-1})$ is the posterior at the previous time step. Here, $Y \in R^M$, and M denotes the total number of states. At each time step, for each state, the probability of the

fly is computed. To this end, a WTA circuit is used to predict the fruit fly position by finding the maximum a posteriori of the probability distribution over states.

To predict the position of the fruit fly, the posterior probabilities of the state space is employed. To this end, an algorithm is utilized, which is similar to the SMC technique as a Monte Carlo method, useful for sequential Bayesian inference (Gordon, 1993). In the proposed framework, spikes denote a probability distribution over a set of states (i.e., the probability of a state is proportional to the sum of its spikes) and the RS (resampling) neuron block encodes the transition model, $P(X_t \mid X_{t-1})$ through spatial connections (**Figure 8B**).

The likelihood generator block has a Poisson neuron (PN), generating spike trains based on its intrinsic firing rate, α and αβ (α: the probability of firing of the *k*th sensory neuron, either due to a fruit fly or a distractor; αβ is a spike when there is no fly, but a distractor instead). To implement the posterior generator block, the two subblocks of the Coincidence Detector (CD) neurons along with the normalization (norm) neural circuits have been utilized. Since the likelihood spike train does not depend on the prior spike train, a simple AND logic gate for the CD neuron can be utilized for the posterior implementation. The output spike trains of the CD neurons as the posterior probabilities of not having a fly and having a fly, respectively, are sent to the norm block to normalize spike trains.

Recurrent connection weights in the framework (shown by red, orange, and purple arrows in **Figure 8B**) are based on the transition probabilities. Spikes from the posterior distributions of adjacent norm neural circuits by considering their transition probabilities are sampled for a pathway by the RS block utilizing an inverse transform sampling approach in a discrete distribution.

Through collecting statistics of the spikes over many HMM time steps, the observation model parameters, α and αβ, are computed. At the start of the learning process, through stochastic exponential moving average filters (SEMAs), the parameters α and αβ are initialized and updated at each HMM time step for each location. An RNG circuit is implemented by the commonly used linear feedback shift register (LFSR) circuit. Neuronal building blocks used for implementing the HMM in **Figure 8B** are the PN, CD neuron, division, and normalization neural circuit, LFSR, and SEMA, which all are implemented on FPGA while all pathways are implemented in parallel on the FPGA hardware too. The implementation of these frameworks using simple logic gates will pave the way for stochastic computing to have digital hardware implementation of Bayesian inference using other approximation inference algorithms in spiking networks.

Hardware Implementation of Approximate Inference Algorithm Using MCMC With Stochastic Logic Gates

By employing a novel abstraction, called combinational stochastic logic, probabilities are directly mapped to digital hardware in a massively parallel fashion (Mansinghka et al., 2008). On each work cycle, the output of a Boolean logic gate is a Boolean function of its inputs. Each gate represents a truth table whereas stochastic gates represent CPTs. **Figure 8C** shows the CPT and schematic for a gate called *O*, which generates flips of a weighted coin by specifying the weight on its input lines (*IN*) with h random bits on *RAND*. A comparator is utilized to implement the O gates where the output will be 1 if $RAND \leq IN$.

Figure 8D shows a serial circuit composed of a stochastic logic gate, an accumulator, and a D flip-flop to implement the Gibbs sampling algorithms for MRFs. For a binomial distribution, this circuit utilizes *nh* bits of entropy to perform sampling while considering n flips of a coin of weight. It provides $O(\log(n))$ space and $O(n)$ time complexity. For a given variable, in order to implement a Gibbs MCMC kernel, a pipeline platform depicted in **Figure 8E** has been proposed (Mansinghka et al., 2008). Each possible setting while considering its neighbors under the joint density of the MRF has been scored by the pipeline and those scores have been tempered. Then, it computes the (log) normalizing constant and normalizes the energies. The normalized energies are translated to probabilities, and finally the pipeline outputs a sample. This pipeline can provide linear time complexity in the size of the variable by utilizing standard techniques and with a stochastic accumulator for sampling (using the circuit in **Figure 8D**). To this end, a fixed-point format is utilized to represent the state values, energies (i.e., unnormalized log probabilities), and probabilities. The logsumexp(*e*1,*e*2) function used for adding and normalizing the energies and the $exp(e1)$ function used for converting the energies to probabilities are approximated. Then, the pipeline samples by exact accumulation. Moreover, numerically tempering a distribution, i.e., exponentiating it to some, can be utilized as energy bit shifting.

The proposed stochastic circuits have been implemented on Xilinx Spartan 3 family FPGAs. Typically large quantities of truly random bits are needed for stochastic circuit implementation. In almost all Monte Carlo simulations high quality pseudorandom numbers are used. For the FPGA implementation in Mansinghka et al. (2008), the XOR-SHIFT pRNG (Marsaglia, 2003) is used.

In order to develop more sophisticated circuits, such as circuits for approximate inference in hierarchical Bayesian models, which is a challenging research field, it is needed to combine the stochastic samplers with stack-structured memories and content-addressable memories (Shamsi et al., 2018; Guo et al., 2019). Moreover, directly using sub-parts from the proposed Gibbs pipeline to implement more sophisticated algorithms, including SMC methods and cluster techniques like Swendsen-Wang, is a promising research effort for the future.

There are a couple of works that provide MRF implementation for different applications *via* utilizing FPGA, application-specific integrated circuit (ASIC), graphics processor unit (GPU), and hybrid implementation *via* CPU+FPGA. Gibbs sampling as a probabilistic algorithm is utilized to solve problems represented by an MRF. In Gibbs sampling method, all random variables in MRF are iteratively explored and updated until converging to the final result (Bashizade et al., 2021). Ko and Rutenbar (2017) explores sound source separation while considering real-time execution and power constraints to isolate human voice from background noise on mobile phones. The implementation

uses MRFs and Gibbs sampling inference, which demonstrates a real-time streaming FPGA implementation that achieves a speedup of 20× over a conventional software implementation. In addition, the approach also has a preliminary ASIC design-based implementation, which requires fewer than 10 million gates, with a power consumption of 52× better than an ARM Cortex-A9 software reference design. For more ASIC optimization, it is necessary to use a lower-power technology library and design optimization for lower memory usage.

In Seiler et al. (2009), an optimization framework utilizing a hierarchical Markov-random field (HMRF) implemented on a GPU is presented to deal with prediction/simulation of soft tissue deformations on medical image data. A method that combines mechanical concepts into a Bayesian optimization framework has been proposed (Seiler et al., 2009). This method has been implemented on a GPU and has been defined and solved under an HMRF approach. Providing an HMRF feature is an appealing technique that is able to solve the proposed stochastic problem since it was found that local minima are avoided. Where using a hierarchical approach and in addition, the nature of the hierarchical approach leads to a straightforward implementation in the GPU. It is assumed that the number of hierarchical levels on the number of iterations for the model to converge has a strong influence, which can be further explored in the future.

In Choi and Rutenbar (2013, 2016) to demand fast and high-quality stereo vision, a custom hardware-accelerated MRF system has been proposed for 3D gesture recognition and automotive navigation. The stereo task has been modeled as statistical inference on an MRF model and shows how to implement streaming tree-reweighted message-passing style inference at video rates. To provide the required speed, the stereo matching procedure has been partitioned between the CPU and the FPGAs. This partitioning provides using both function-level pipelining and frame-level parallelism. Experimental results show that this system is faster than several recent GPU implementations of similar stereo inference methods based on belief propagation.

As can be seen, there are still open windows to utilize new emerging nonvolatile devices and crossbar arrays to implement MRFs rather than just utilizing FPGA, ASIC, GPU, and hybrid implementations (CPU + FPGA). Moreover, refining the algorithms to make them more amenable to hardware implementations is needed while keeping the accuracy high.

Muller C-Element Based Bayesian Inference

In order to calculate the probability of an event V, Bayesian inference incorporates the probability of V given the prior $P(V)$ and evidence input $E1$ as in Equation (12), where, with parameter as defined by Equation (13), Equation (14) gets rewritten as Equation (15).

The Muller C-element reported in Friedman et al. (2016), a two-input memory element, characterized by the truth table of **Figure 9A**, and shown in **Figure 9B**, performs the complete inference of Bayes' rule. The output Z keeps its state, Z_{prev} while both inputs X and Y are opposite the current output state; afterward, it switches to the shared input value. A Muller C-element is able to compute Equation (14), thereby enabling efficient inference circuits. Note that input signals i with switching probabilities a_{i} and b_{i} for 0- > 1 and 1- > 0 switching, respectively, show no autocorrelation if $a_{\text{i}} + b_i = 1$. Then, considering no autocorrelation for input signals, the output probability is defined by Equation (15) for C-element, where $P^*(E1)$, $P(V)$, and $P(V \mid E1)$ are substituted by for $P(X)$, $P(Y)$, and $P(Z)$. The reported Equation (15) is equivalent to Equation (14), representing the Bayesian inference provided by C-elements.

$$\boldsymbol{P(V|E1)} = \frac{\boldsymbol{P(E1|V)P(V)}}{\boldsymbol{P\,(E1 \mid V)\,P\,(V) + P(E1|\overline{V})P(\overline{V})}} \tag{12}$$

$$\boldsymbol{P^*(E1)} \equiv \frac{\boldsymbol{P(E1|V)}}{\boldsymbol{P(E1|V) + P(E1|\overline{V})}} \tag{13}$$

$$\boldsymbol{P(V|E1)} = \frac{\boldsymbol{P^*(E1)P(V)}}{\boldsymbol{P^*(E1)P\,(V) + (1-P^*\,(E1)\,)(1-P\,(V)\,)}} \tag{14}$$

$$\boldsymbol{P\,(Z)} = \frac{\boldsymbol{P(X)P(Y)}}{\boldsymbol{P(X)P(Y) + (1-P(X))(1-P(Y))}} \tag{15}$$

Clocked bitstreams in stochastic computing are utilized to encode probabilistic signals permitting complex computations with minimal hardware and significantly improve the computation power consumption and inference speed when compared with conventional methods. Stochastic computing is not an exact computing technique and the slight loss of accuracy arises from several reasons. Compared to fixed or floating-point methods, in stochastic computing, the probability values P are usually translated to a stochastic bitstream with a lower quantization accuracy and the correlations between bitstreams usually lead to the loss of accuracy, since these bitstreams are usually generated by pseudo RNGs. Addressing this inherent imprecision and correlations need novel design techniques.

In Friedman et al. (2016), the number of "1"s in a bitstream encodes its probability and has nothing to do with the position of the 1 bits. In a stochastic bitstream, to represent a state switching probability a (b), i.e., the dynamics of a 0- > 1(1- > 0) switching, the probability R defined as R = a/(a+b). For an uncorrelated bitstream (i.e., a+b = 1), the probability is equivalent to R = a, where being "1" has a probability of R and being "0" has a probability of 1-R. Then, the switching rate for an uncorrelated bitstream is defined by Equation (16):

$$\boldsymbol{S = 2R(1\text{-}R) = 2a(1\text{-}a)} \tag{16}$$

The C-element outputs a stochastic bitstream, which is probabilistic and converging more slowly toward the exact Bayesian inference. If the switching rate of the output was low, the longer "domains" of consecutive "0"s and "1"s are needed and it leads to a more imprecise bitstream. Hence, more computation time is required to provide a precise output.

For multi-input Bayesian inference calculation, utilizing multi-stage C-element circuits is necessary, which would need one additional cycle per stage to compute a bitstream. On the other hand, the floating-point circuit provides a highly precise

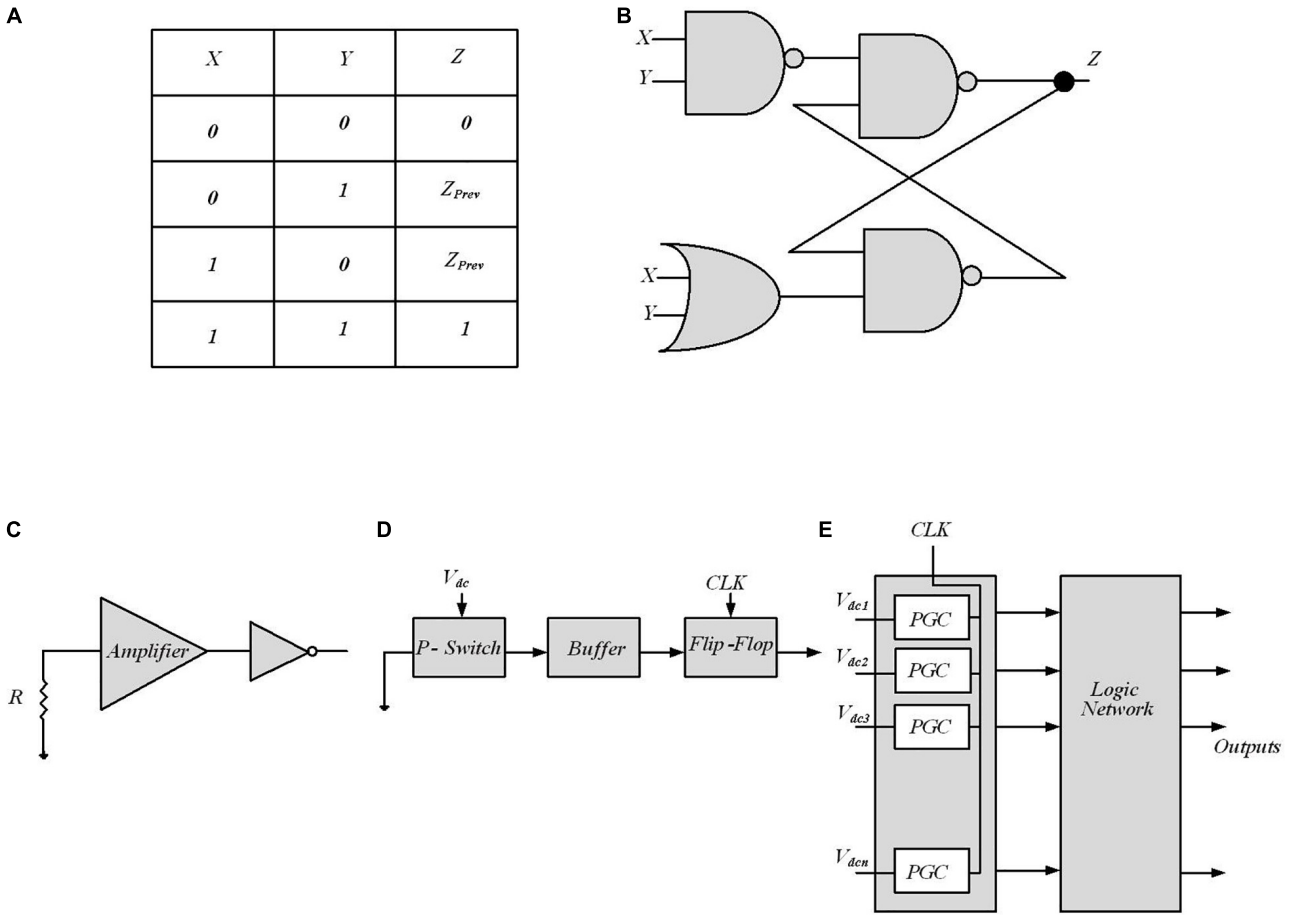


FIGURE 9 | Digital implementation of Bayesian inference. **(A)** Muller C-element truth table in Friedman et al. (2016). **(B)** Standard cell C-element design of Muller C-element. PCMOS-based Bayesian network in Weijia et al. (2007). **(C)** p-switch circuit implementation block. **(D)** Probabilistic generating cells (PGCs) block. **(E)** The inference system utilizes a probabilistic generating block and a logic network.

output while needing multiple pipelined computations and a long characteristic delay time. Hence, the C-element structure's performance benefit is dependent on the required precision for the specific application.

For embedded decision circuits, where different independent sources of evidence are considered, for computing the probability of an event, C-element trees can provide direct stochastic hardware implementation. However, exploring the autocorrelation and inertia mitigation through signal randomization is required for further studies. For extreme inputs with low switching rates, the loss of accuracy is significantly increased. By increasing the length of the bitstream, the output signals converge in a polynomial manner to Bayesian precise inference. In addition, C-element trees have larger errors for opposing extreme input combinations. It is mentioned that this type of input and the error can be considered as strong conflicting evidence and the inference uncertainty, respectively.

The standard cells from Synopsys (Synopsys, 2012) SAED-EDK90-CORE library are used (Tziantzioulis et al., 2015) for C-Muller module implementation. For a two-input Bayesian inference implementation the standard cells have been employed and the simulation results showed that the floating-point circuit utilizes 16,000× area more than a C-element. This is due to the fact that for a two-input inference problem, just one C-element is required while the conventional floating-point circuit needs addition, multiplication, and division units. Also, for multi-input Bayesian inference, the C-element still outperforms the floating-point circuit.

Probabilistic CMOS Based Bayesian Inference

In Weijia et al. (2007), probabilistic CMOS (PCMOS) technology has been used to implement RNGs to create a highly randomized bit sequence suitable for inference in a Bayesian network. A PCMOS-based RNG is composed of the PCMOS switch or p-switch, which is a CMOS switch with a noise source coupled at its input node. **Figure 9C** shows a p-switch block. The resistor is employed as a source of thermal noise, which follows the Gaussian distribution. An amplifier is used to amplify the noise signal to have a comparable signal with supply voltage.

The inference system is shown in **Figure 9E**, composed of probabilistic generating block and logic network. The probabilistic generating block generates random bits with different probabilities, and the logic network defines the edges between the nodes in a Bayesian network. The probabilistic generating block is composed of a number of probabilistic generating cells (PGCs), each of which generates a "1" bit with a probability. A PGC shown in **Figure 9D** is made up of a p-switch, a buffer, and a flip-flop. The buffer constructed from two inverters strengthens the output signal of the switch. The flip-flop, formed by two D latches, synchronizes the PGCs. Arithmetic operations (addition and multiplication in Bayesian network) computed in computers require a lot of time and energy. Here, two simple logic gates (an AND gate and an OR gate), together with some inverters, are employed to construct the logic network. To determine the approximate probability of the output at each node, a simulation has been performed to generate a 10,000-bits sequence at each node and then measure the "1"-bits in each sequence. The PCMOS-based hardware implementation of the Bayesian network outperforms the software counterpart in terms of energy consumption, performance, and quality of randomness. However, making use of mixed-signal implementations needs paying attention to noise and variation sources as well as examining the multiple independent sources of evidence for embedded decision circuits that require circuit design remedies.

CROSSBAR ARRAYS FOR BAYESIAN NETWORKS IMPLEMENTATION

In this section, two brain-inspired hardware implementations of inference in naïve Bayesian (NB) classifiers will be discussed. These implementations use memristors as nonvolatile elements for the inference algorithm implementation. Bayesian reasoning machine with magneto-tunneling junction-based Bayesian graph is explained.

Crossbar Arrays for Naïve Bayesian Classifiers

A crossbar array of memristors is a promising hardware platform for Bayesian processing implementation in a massively parallel and energy-efficient way (Yang et al., 2020). **Figure 10A** depicts a schematic view of a memristor cell, in which a storage layer is sandwiched between the top and bottom electrodes, and the conductance of the device is dependent on the applied voltage. **Figure 10B** shows a crossbar array; it represents a maximum area efficiency of $4F^2$ per cell (Wu et al., 2019). Memristor crossbar arrays provide a natural implementation of matrix-vector multiplication (MVM). The current flowing through a memristor cell at the wordline x and bitline y is equal to $V_x g_{(x,y)}$. Here, V_i is the voltage applied to the wordline x and $g(x, y)$ is the conduction of the cell. The total current through the bitline y is $\sum_x V_x\, g(x, y)$, which implements a dot product of V_x. $g(x, y)$. The algorithmic complexity of MVM is reduced from $O(n^2)$ to O(1), which makes them a promising computing paradigm for different machine learning applications (Wu et al., 2019).

To perform Bayesian inference, **Figure 10C** shows a memristive crossbar array where a discrete distribution represented by a voltage is injected to the wordlines, the conditional probability $P(B|\ A)$ translate to the memristor conductance, and all bit-lines are virtually earthed. Utilizing the current summing action of the crossbar bitlines, the current of each memristor is proportional to $P(B|\ A)\cdot P(A) = P(A, B)$, which is marginalized to $P(B)$. Finally, inputs are multiplied by memristor conductances (g_k) and exit as currents.

In analog systems, due to the noise, mismatch, and other variation sources, the input vectors do not necessarily meet the fact that the probability distributions of random variables must sum up to 1. To this end, the "normalizer" circuit is employed as a supporting module. Moreover, utilizing a linear method to convert the probability into voltage levels or memristor resistive states limits the dynamic range of the probability. That is, very small probability values may be translated into voltages below the noise levels in the system (Serb et al., 2017). However, the normalizers could scale these values when they are very low, but similar. It turns out to be problematic if there are very large probability values as well as very low ones in the same distribution. To solve this issue, it has been suggested that the resistive state/voltage needs to be mapped to the log probability domain (Wu et al., 2019).

Naïve Bayesian classifiers assume that the feature variables are all independent of each other (Serb et al., 2017) and the classification is based on the Bayesian theorem. For a test instance x, represented by an attribute value vector (A, B), the NB finds a class label c that provides the maximum conditional probability of c given A, B.

In Serb et al. (2017), a small graphical model for the prediction of potential health issues (**Figure 10D**) has been supposed to be implemented in memristor crossbar arrays, where A shows the air quality as $A \in$ {bad, medium, good}, and B shows the corresponding heartbeat of the patient for two different activities $B \in$ {resting, exercising}. Then, by considering random variables A, B, in order to predict the probability of a health crisis, and thus to clarify whether to warn the patient, i.e., $C \in$ {safe, crisis} with a classic NB classifier, the goal of NB is to find a class label c that has the maximum conditional probability of c given A, B (as attributes):

$$\boldsymbol{C}^* = \boldsymbol{max}_{\boldsymbol{C}}\{\boldsymbol{P}(\boldsymbol{C}|\boldsymbol{A}, \boldsymbol{B})\}$$
$$\textit{where } \boldsymbol{P}(\boldsymbol{C}|\boldsymbol{A}, \boldsymbol{B}) \propto \boldsymbol{P}(\boldsymbol{A}|\boldsymbol{C})\,\boldsymbol{P}(\boldsymbol{B}|\boldsymbol{C})\,\boldsymbol{P}(\boldsymbol{C}) \qquad (17)$$

C^* is defines as the maximum *a posteriori* estimate.

Figure 10E depicts the hardware implementation process of the proposed NB framework. With air quality level A as input and the crisis level prediction C as an output, first, a crossbar stores $P(A, C) = P(A|\ C)P(C)$, before factoring heart rate B in. Then, the output is sent in parallel to two arrays of memristors that maintain $P(B = \text{resting}|\ C)$ and $P(B = \text{exercising}|\ C)$, respectively. Based on the heartbeat B, one of the two outputs would be selected to put into the normalizer to calculate $P(C|\ A, B)$. Finally, the max-finder module finds the estimate C^*. This inference platform depicts that with the crossbar arrays as well

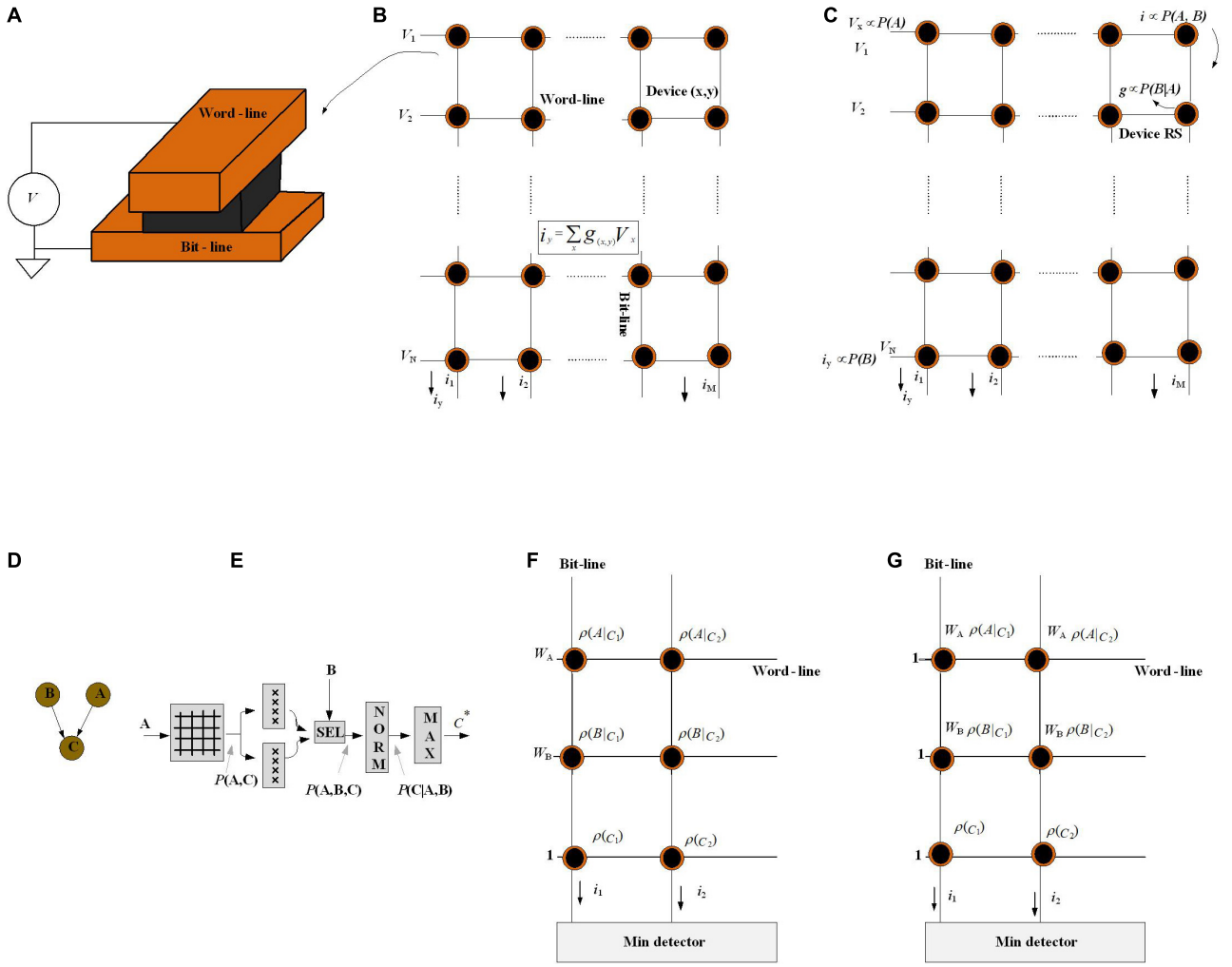


FIGURE 10 | (A) Schematic of the memristor device in which the device's active material is surrounded by two electrodes [top (wordline) and bottom (bitline)]. **(B)** Ohm's law: $i = g.V$ is utilized to perform multiplier operation. **(C)** The crossbar array is used as a Bayesian inference system. **(D)** Graphical model for Bayesian network. **(E)** Implementation of the Bayesian classifier. **(F)** Implementation of the naïve Bayesian classifier for a network of two attributes. **(G)** Implementation of a different way to calculate I_C. In this method, the weight of the ith attribute (w_i) is stored in the cell resistance.

as utilizing a cascade of small modules, it is able to scale to more complicated graphical models.

As discussed above, by directly employing the multiply accumulate capabilities of the crossbar array, the inference can be performed. During learning, as new data arrives, the conditional probability matrix needs to be updated; thus, the devices in the crossbar need to be programmed. The conductance stability and the energy efficiency of memristor switching, i.e., how many attempts are needed to reach the memristors desired state, determine the energy, speed, and circuit complexity cost of the probability updates (Serb et al., 2017). In Equation (17), it has been assumed, given the class, that all attributes (A, B) are fully independent of each other. The classification accuracy would be harmed when this assumption is violated in reality.

Wu et al. (2019) propose another analog crossbar computing architecture to implement the NB algorithm while considering the abovementioned concerns. It assigns every attribute a different weight to indicate different importance between each other. This assignment relaxes the conditional independence assumption. The prediction formula is formally defined as:

$$C^*(x) = max\{P(c)P(A|c)^{w_A}P(B|c)^{w_B}\}, c \in C \qquad (18)$$

where w_A and w_B are the weight of attributes A and B, respectively. The NB classifier in Equation (17) is a special case of the Weighted NB (WNB) classifier when w_A and w_B are equal to 1.

Naïve Bayesian formula [Equation (18)] transformation to the crossbar array Equation (18) cannot be directly applied to the Memristor crossbar array (concern 1). So, a log(•) operation is applied because $P(\bullet) \in (0, 1)$. log $P(\bullet)$ is a negative value that cannot be represented by the conductance of memristor cells as the conductance is always positive; then, $\rho(\bullet)$ denotes -log $P(\bullet)$

and then Equation (18) is rewritten as:

$$C^*(x) = min\{\rho(c)+w_A \cdot \rho(A|c) + w_B.\rho(B|c)\} \quad c \in C$$
$$q(c) = \rho(c)+w_A.\rho(A|c)+w_B.\rho(B|c) \tag{19}$$

The $q(c)$ rewritten in the form of a dot product $\vec{v} \cdot \vec{g}$, where $\vec{v} = [1\ w_A, w_B]$ and $\vec{g} = [\rho(c), \rho(\mathbf{A} \mid \mathbf{c}), \rho(\mathbf{B} \mid \mathbf{c})]$. Hence, it is feasible to compute q of every class by the MVM.

After training, every prior probability $\rho(c)$ is stored, as well as every conditional probability $\rho(A|\ c)$ in the crossbar array in the form of memristor conductance, where $c \in C$.

For attribute A, voltage w_A is applied to the wordline (**Figure 10F**) and the current gathered on this sub-bitline ($I^A{}_c$). With the addition of $\rho(c)$, the final result is obtained as current on one bitline. Multiple bitlines together give answers of Equation (19) to all classes. Optimization has also been proposed to the input voltage. Due to the I–V nonlinearity of the ReRAM cell, the analog input voltage (i.e., w_i) might result in inaccuracy. The weight w_i is included in the cell conductance shown in **Figure 10G**.

The simulations show that the design offers a high runtime speedup up with negligible accuracy loss over the software-implemented NB classifier. This brain-inspired hardware implementation of NB algorithm as well as providing insights from techniques like mean-field approximation (Yu et al., 2020) will help to find an optimal balance between structure and independence, using hardware feasibility considerations and independence assumptions as mutually constraining objectives, which can be a promising research field.

Bayesian Reasoning Machine With Magneto-Tunneling Junction-Based Bayesian Network

Predictions from Bayesian networks can be accelerated by a computing substrate that allows high-speed sampling from the network. Nasrin et al. (2020) provide the development of such a platform to map an arbitrary Bayesian network through an architecture of the MTJ network along with circuits to writing, switching, and interactions among MTJs. By these means, electrically programmable sub-nanosecond probability sample generation, voltage-controlled magnetic anisotropy (VCMA), and spin-transfer torque (STT) have been co-optimized. As **Figure 11A** shows for programmable random number generation, VCMA, STT (applied *via* the voltage VCMA), and magnetostriction, i.e., strain (injected with the voltage V_{St}), in an MTJ are co-optimized. To stochastically couple the switching probability of one MTJ depending on the state of the other, as **Figure 11B** depicts, MTJ integration is required, in which dipole coupling, controlled with local stress, is applied to one MTJ. This results in electrically tunable correlation between the bits "*A*" and "*B*" (encoded in the resistance states of the two MTJs), without requiring energy-inefficient hardware like OP-AMPS, gates, and shift-registers for correlation generation. To compute posterior and marginal probabilities in Bayesian networks *via* stochastic simulation methods, samples of random variables are drawn to determine the posterior probabilities. For the platform, mere stochasticity in devices is not enough, and for a scalable Bayesian network, "electrically programmable" stochasticity to encode arbitrary probability functions, $P(x)$; $x = 0$ or 1, is required; moreover, this "electrically programmable" stochasticity is necessary for stochastic interaction among devices for conditional probability, $P(x|\ y)$. In the presence of thermal noise at room temperature, the "flipping" is stochastic, i.e., the magnetization will precess when V_{VCMA} is turned on and can either return back to the original orientation or flip to the other orientation. By adjusting the magnitude of V_{VCMA}, the probability of flipping can be tuned. Therefore, the voltage V_{VCMA} as a knob controls the probability of getting either "0" or "1." The MTJ grid in **Figure 11C** only enables the nearest-neighbor correlation, and each node can only have binary states. For nodes with more than two states, splitting by binary coding is required. In order to run a general Bayesian network on the 2D grid, new mapping and graph partitioning/restructuring algorithms must be developed.

In **Figure 11C**, an example mapping strategy is shown to run general edges in a graph. Graph nodes are duplicated by setting the coupling voltages for perfect anti-correlation. To perform independent sampling on the MTJ grid, it is required to map the parent variables on the parent MTJ column and the children on the successive columns. In the stochastic simulation, different sampling algorithms on the grid are tested to speed up the process of sample generation of random variables in a Bayesian network to compute the posterior probabilities. These algorithms speed up the inference in Bayesian networks but can still fall short of the escalating pace and scale of Bayesian network-based decision engines in many Internet of Things (IoT) and cyber-physical systems (CPS). With a higher degree of process variability, prediction error for $P(F)$ increases. By increasing the size of the components (resistive memory, current biasing transistor, etc.) as well as post-fabrication calibration, tolerance to process variability in the proposed design can be increased. The discussed platform would pave the way for a transformational advance in a novel powerful generation of ultra-energy-efficient computing paradigms, like stochastic programming and Bayesian deep learning.

BAYESIAN FEATURES IN NEURAL NETWORKS

In this section, employing Bayesian features in neural networks is represented. To this end, first Bayesian neural networks are explained. Then, Gaussian synapses for PNNs will be introduced. Afterward, a PNN with memristive crossbar circuits is described. At the end of this section, approximate computing to provide hardware-friendly PNNs and an application of probabilistic ANN for analyzing transistor process variation are explained.

Bayesian Neural Networks

Bayesian deep networks define the synaptic weights with a sample drawn from a probability distribution (in most cases, Gaussian distributions) with learnt mean and variance and inference based on the sampled weights. In Malhotra et al. (2020), the

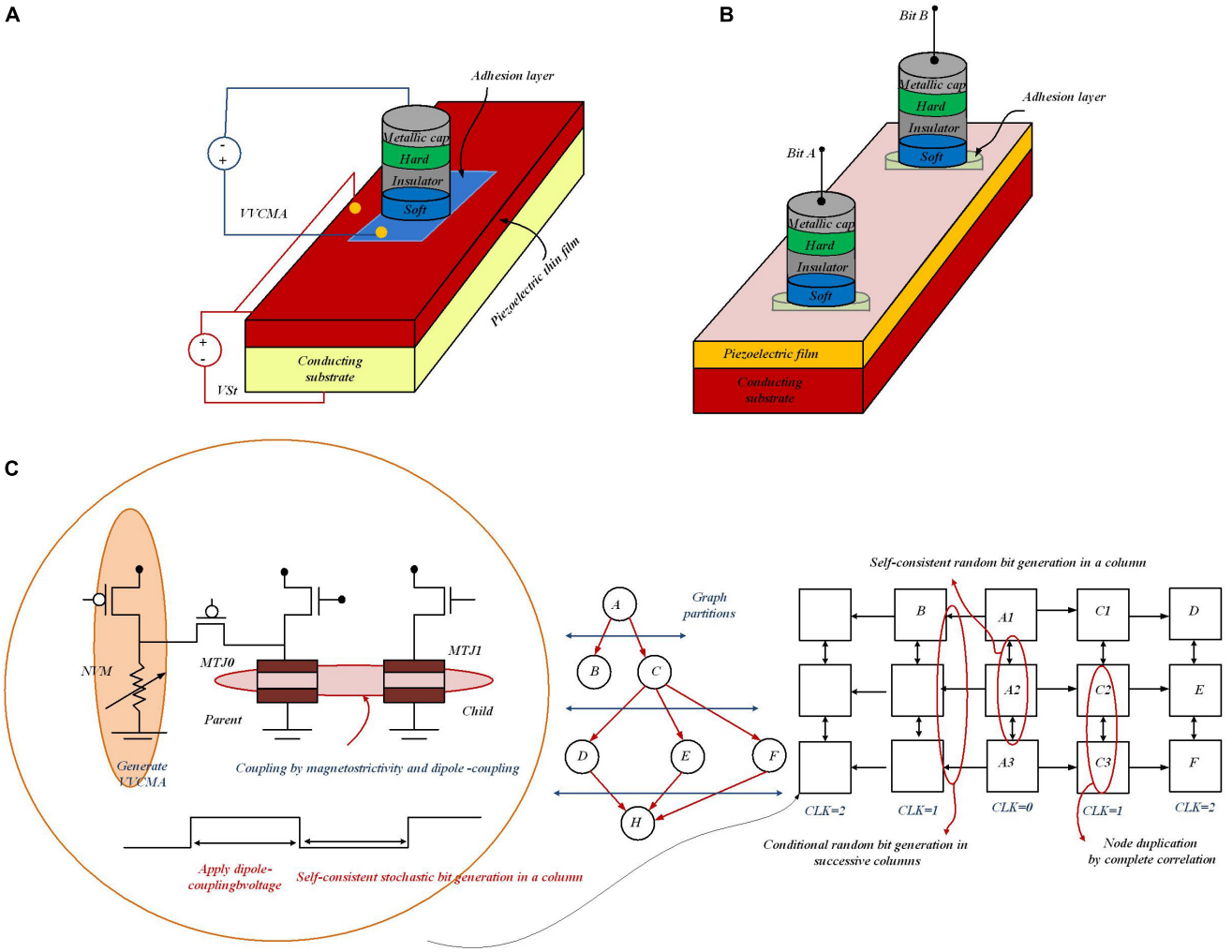


FIGURE 11 | Stochastic random number generation utilizing MTJs with programmable probability. **(A)** MTJ with VCMA and STT (applied *via* the voltage V_{VCMA}). For programmable random number generation, the magnetostriction, i.e., strain (controlled with the voltage V_{St}) can be co-optimized. **(B)** MTJ integration utilizing the effect of dipole coupling (controlled with local stress applied to one MTJ) can be used to couple the switching probability of one MTJ depending on the state of the other. Thereby, the correlation between the bits "*A*" and "*B*" can be controlled *via* the resistance states of the two MTJs. **(C)** MTJ network-based Bayesian reasoning machine to show an example mapping of Bayesian graph on 2D nanomagnet grid.

gradual reset process and cycle-to-cycle resistance variation of oxide-based resistive random access memories (RRAMs) and memristors have been utilized to perform such a probabilistic sampling function.

Unlike standard deep networks, defining the network parameters as probability distributions in Bayesian deep networks allows characterizing the network outputs by an uncertainty measure (variance of the distribution), instead of just point estimates. These uncertainty considerations are necessary in autonomous agents for decision-making and self-assessment in the presence of continuous streaming data. In Bayesian formulation, defined by Equation (20), $P(W)$ represents the prior probability of the latent variables before any data input to the network and $P(D|\ W)$ is the likelihood, corresponding to the feedforward pass of the network. $P(W|\ D)$ is the posterior probability density where two popular approaches, variational Bayes inference methods and Markov chain Monte Carlo methods, are used to make its estimation tractable.

$$\boldsymbol{P}(\boldsymbol{W} \mid \boldsymbol{D}) = \frac{\boldsymbol{P}(\boldsymbol{D} \mid \boldsymbol{W})\,\boldsymbol{P}(\boldsymbol{W})}{\boldsymbol{P}(\boldsymbol{D})} \tag{20}$$

In Malhotra et al. (2020) and Yang et al. (2020), the variational inference approach has been used since it is scalable to large-scale problems. In the variational inference approach, to approximate the posterior distribution, a Gaussian distribution, $q(W, \theta)$, is used. $q(W, \theta)$ is characterized by parameters, $\theta = (\mu, \sigma)$ in which μ and σ, respectively, are the mean and standard deviation vectors for the probability distributions representing $P(W|\ D)$ [see Equation (21)]. The main hardware design concerns for implementation of Bayesian neural networks are Gaussian random number generation block and dot-product operation between inputs and sampled synaptic weights.

A Normal distribution with a particular mean and variance is equivalent to a scaled and shifted version of a Normal distribution with zero mean and unit variance. This consideration would allow partitioning the inference equation as shown in Equation (22). The $_{jk}$, and σ_{jk} are the mean and variance of the probability distribution of the corresponding synaptic weight. As shown in **Figure 12A**, to construct the resultant system, the domain-wall MTJ memory devices based on two crossbar arrays are used for the $_{jk}$ and σ_{jk} implementation, respectively. While the inputs of a particular layer are directly applied to the crossbar array storing the mean values, they are scaled by the random numbers generated from the RNG unit.

The output of the network, y, corresponding to input, x, is defined by Equation (21). As all the posterior distributions are learnt, the network output averages the outputs provided by sampling from the posterior distribution of the weights, W, where, $f(x,W)$ is the network mapping for input x and weights, W.

$$y = E_{P(W|D)}\left[f(x,w)\right] \sim E_{q(w,\theta)}\left[f(x,w)\right] \sim \frac{1}{S}\sum_{i=1}^{s} f(x,w^i) \quad (21)$$

The approximation is done over S independent Monte Carlo samples from the Gaussian distribution, $q(W,\theta)$. $f(x,W_i)$ for the jth neuron can be decomposed into Equation (22), by considering just a single layer and neglecting the neural transfer function.

$$\begin{aligned} f(x, w_{ij}) &= \sum_k x_k N(\mu_{jk}, \sigma_{jk}) \\ &= \sum_k x_k \cdot (\mu_{jk} + \sigma_{jk}.N(0,1)) \\ &= \sum_k x_k \cdot \mu_{jk} \sum_k x_k.N(0,1)\sigma_{jk} \end{aligned} \quad (22)$$

The proposed "all-spin" Bayesian neural processor has the potential of providing orders of magnitude area, power, and energy consumption efficiency over the state-of-the-art CMOS implementations. A significant rethinking of the co-design space of device circuits and algorithms is necessary for Bayesian deep learning since it provides a unique computing framework that combines both deterministic (dot-product evaluations of sampled weights and inputs) and stochastic computations (sampling weights from probability distributions).

Gaussian Synapse-Based Hardware Implementation for Probabilistic Neural Networks

In the computing revolution era, scaling in the semiconductor industry is inevitable and has three characteristic aspects: energy scaling, size scaling, and complexity scaling. Energy scaling satisfies the situation of the practically constant computational power budget. Through size scaling, more transistors can be fabricated in the same chip area, which consequently provides a faster and cheaper computing system. Complexity scaling ensures incessant growth in the computational power of a single on-chip processor. Considering these requirements, Sebastian et al. (2019) enable the hardware implementations of PNNs (shown in **Figure 12B**) *via* introducing a new class of analog devices, namely, the reconfigurable Gaussian synapses based on the heterostructure of atomically thin 2D layered semiconductors (shown in **Figure 12C**). The 2D materials satisfy aggressive size scaling while energy scaling is ensured *via* analog Gaussian synapses, and complexity scaling is met by PNNs. *Via* threshold engineering of the proposed device, it shows complete compatibility of amplitude, mean, and standard deviation of the Gaussian synapse. As shown in **Figure 12B**, unlike ANN, which employs multiple hidden layers with a large number of nodes in each layer, PNN proposed by Specht (1990) is a supervised learning neural network based on Bayesian decision rule and is composed of a pattern layer and a summation layer. PNNs are able to map any input pattern to any number of output classifications. Furthermore, in ANNs, activation functions such as sigmoid and rectified linear unit (ReLU) are used, where various derivatives of these functions have been utilized to determine the pattern statistics (which are extremely difficult for non-linear decision boundaries to perform with reasonable accuracy). In PNNs, parent probability distribution functions (PDFs) are used for the class probability. PDFs are approximated by a Parzen window and a non-parametric function, which is a Gaussian distribution for a Gaussian kernel (Specht, 1990). In PNNs, arbitrarily shaped decision boundaries are used, which facilitate the accurate classification of complex patterns. Moreover, since multivariate Gaussian kernels are simply generated from the product of univariate kernels, PNNs can be extended to map higher-dimensional functions. A reconfigurable Gaussian synapse, with dual-gated (DG) MoS_2 and BP FETs, is shown in **Figure 12C**. Hydrogen silsesquioxane (HSQ) was used for the fabrication of the top-gate dielectric. Nickel/gold was used (Ni/Au) for the top-gate electrode fabrication for different top-gate voltages (V_N). The back-gate threshold voltage (V_{TN} of the MoS_2 FET) is tuned by V_N. The top-gate voltage is tuned to control the height of the potential barrier for electron injection inside the MoS_2 channel. Moreover, a back-gate voltage conducts current from the source to the drain terminal. The PNN architecture has been implemented on brainwave recording data, for each type of brainwaves. The frequency pattern of the normalized power spectral density (PSD) is extracted from the fast Fourier transform (FFT) of the time domain electroencephalography (EEG) data with increasing sampling times. As the training set becomes large, the discrete frequency responses of each type of brainwave evolve into continuous spectrums representing complex patterns. The functional dependence of the PSDs on the frequency makes the system highly nonlinear. Hence, using conventional ANNs can be challenging for the classification of brainwave patterns. To provide reasonable accuracy in ANNs, optimum training algorithms and extensive feature extraction and preprocessing of the training sample are required, while PNNs provide single-pass learning. This learning mechanism happens *via* defining the class PDF for each of the brainwave patterns in the frequency domain through employing the Gaussian mixture model (GMM). As described by Equation (23), GMM is represented as the weighted sum of a finite number of

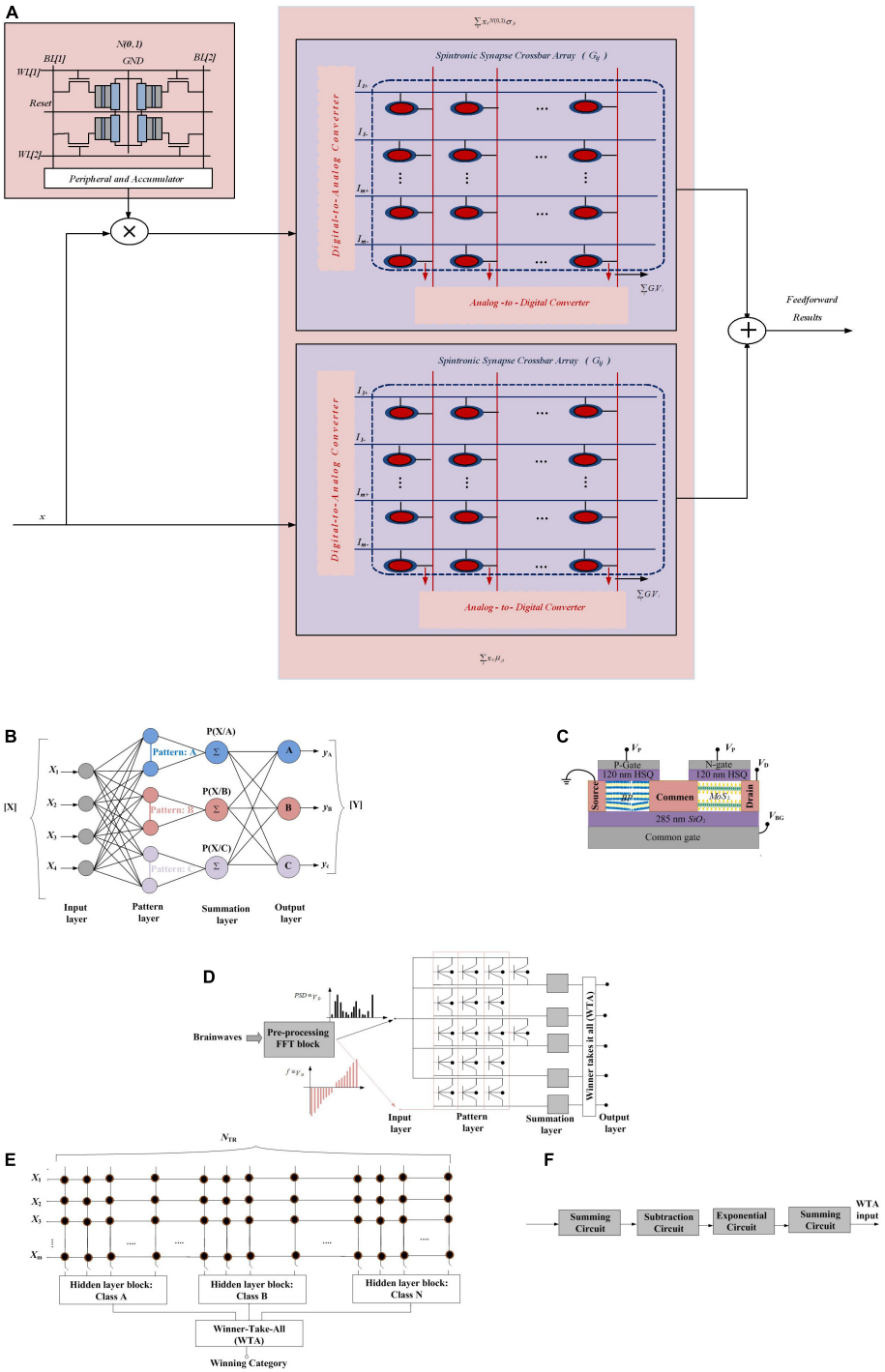


FIGURE 12 | (A) All-spin Bayesian neural network implementation. The RNG unit performs sampling operation from Gaussian random number generators and the two crossbar arrays provide the "In-Memory" computing kernels (Malhotra et al., 2020). **(B)** The structure of the Probabilistic Neural Network (PNN) model (Gaussian Synapse based PNN) translates any input pattern to any number of output classifications through using a pattern layer and a summation layer. **(C)** Schematic of a reconfigurable Gaussian synapse composed of dual-gated n-type MoS_2 and p-type black phosphorus (BP) back-gated field-effect transistors (FETs). Hydrogen silsesquioxane (HSQ) was used as the top-gate dielectric and nickel/gold (Ni/Au) was used as the top-gate electrode. **(D)** PNN Architecture for Brainwave recognition. The amplitude of the FFT data is passed from the input layer to the pattern layer as drain voltage (V_D) of the Gaussian synapses, and the frequency range is mapped to the back-gate voltage (V_G) range. The summation layer integrates the current over the full swing of V_G from the individual pattern blocks and communicates with the winner-takes-it-all (WTA) circuit and then the output layer recognizes the brainwave patterns. **(E)** The architecture of the PNN (Akhmetov and Pappachen, 2019) is implemented with crossbar arrays where each class has been implemented with a crossbar array where N_{TR} denotes the total size of the training set (classes). **(F)** Block diagram of the hidden layer block.

scaled (different variance) and shifted (different mean) normal distributions. ψ_I as component weights, μ_i as component means, and ${\sigma_i}^2$ as variances are for parameterizing a GMM with K components through which the total probability distribution must normalize to unity.

$$P(x) = \sum_{j=1}^{k} \psi_i N\left[\frac{x}{\mu_i}, \sigma_i\right]; N\left[\frac{x}{\mu_i}, \sigma_i\right]$$
$$= \frac{1}{\sqrt{2\psi_i^2}} \cdot exp^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}; \sum_{j=1}^{k} \psi_i = 1 \quad (23)$$

For each type of brainwave pattern, the GMM parameters for the K components are estimated based on the training data and utilizing the non-linear least square method. Root mean square errors (RMSEs) are calculated as a function of K. K denotes the number of Gaussian curves used in the corresponding GMMs. For each of the brainwaves, to define the non-linear decision boundary, a limited number of Gaussian functions are required. Hence, the energy and size constraints for the PNNs based on Gaussian synapses are enormously reduced. Finally, the PNN architecture shown in **Figure 12D** is evaluated for the detection of new brainwave patterns. The amplitude of the new FFT data in PNN (which consists of input, pattern, summation, and output layers) is passed as the drain voltage (V_D) of the Gaussian synapses from the input layer to the pattern layer. The frequency range is translated to the back-gate voltage (V_G) range. The summation layer collects the current over the full swing of V_G from the individual pattern blocks. After current integration in the summation layer, the currents communicate with the WTA circuit. The WTA detects the brainwave patterns in the output layer. It is shown that utilizing Gaussian synapses in PNN architecture can recognize complex neural oscillations and brainwave patterns from a large number of EEG data providing extreme energy efficiency, which will foster the feasibility of efficient hardware implementation of PNNs and subsequently high-performance and low-power computing paradigm.

Probabilistic Neural Network With Memristive Crossbar Circuits

Probabilistic neural network architecture (Specht, 1990) provides a fast training mechanism in which weights are derived from training samples directly and set in the first initialization stage. Then, the density functions of the categories are estimated based on the training dataset. The input samples are classified based on these density functions. PNNs provide the ability to converge to Bayes optimal decision surface without trapping to local minima. Moreover, a new training pattern can be added to the network that does not require any global retraining process. On the other hand, for hardware implementation of the near-edge computing devices, the processing speed, the size of the network, and the power consumption are critical. PNN's parallel computational nature and fast learning PNNs make them attractive for hardware implementation and utilization in near-edge computing devices. In Akhmetov and Pappachen (2019), a hardware implementation of the PNNs based on the memristive (ReRAM based) crossbar architecture has been proposed (shown in **Figure 12E**); to this end, a crossbar with N_{TR} dimensions is utilized to perform dot product between weights of the pattern neurons and input vector, where N_{TR} denotes the total size of the training set. The proposed circuit provides the density estimation and classification performed by the PNN. As shown in **Figure 12B**, the input layer of the PNN distributes an input to pattern neurons. The pattern layer performs a dot-product operation and exponential activation. The summation neurons integrate the outputs of pattern neurons belonging to one class and then in the output layer the decision is made. In the output layer, the density functions are scaled by their prior probability and loss function; after that, the category with the highest posterior probability is chosen as the output of the PNN. The hidden layer block shown in **Figure 12E** computes the approximate density functions of categories based on the training set and is composed of summing circuits, a subtraction circuit, and the exponential function generator (**Figure 12F**). The sub-blocks of the hidden layer block are implemented with CMOS circuits. The system-level simulation showed that the proposed implementation of the PNN is insusceptible to process variation of the ReRAM and provides a high accuracy on the MNIST dataset. Future studies should implement a memristor programming circuit (to provide on-chip learning), employ alternative kernel functions and ReRAM devices, and utilize a larger dataset.

Approximate Computing to Provide Hardware Friendly Probabilistic Neural Networks

Approximate computing greatly improves computing in computer systems *via* accomplishing more tasks under the same resource consumption. On the other hand, a large number of floating-point operations and multipliers are required in DSP hardware architectures needing a large number of hardware resources. Although by using fixed-point arithmetic implemented in hardware the DSP algorithm can process the constant multiplication simultaneously, this can reduce the accuracy of the calculation. To solve these hardware circuit design problems, the PNN hardware architecture of approximate calculation using a genetic algorithm (GA) has been proposed in Chen et al. (2019). GA realizes approximate calculation of the hardware circuit of PNN, to achieve the best balance between maintaining good classification ability and the least hardware resource consumption to reduce the hardware complexity. The key concept of GAs is to imitate the natural evolution law of natural selection in nature and to solve the optimization problem utilizing three main operators: reproduction, crossover, and mutation. Firstly, one encodes all the parameters into chromosomes, and defines a fitness function. The evolution starts from the population of completely random individuals, evaluates the adaptability of each chromosome to the environment in each iteration process, and then generates the new population through natural selection and mutation. This is to be repeated until the final break conditions are met. The hidden layer neurons of PNNs (shown in **Figure 12B**) are responsible for the computer rate density

function, which performs the nonlinear transformation from the input space to the hidden layer. The weight vector of hidden layer neurons represents a training pattern, and the probability density function is a Gaussian function in multidimensional feature space, which is a nonlinear function. Such nonlinear functions are often implemented on hardware. In addition, the Gaussian function is decided by a smoothing coefficient of its distribution scope σ. The larger σ, the wider the breadth, and the smaller σ, the narrower the breadth. If the input vector is located close to the center of the Gaussian function, the hidden layer node will generate a larger output. In practical engineering, the look-up table is often used to approximate these nonlinear functions. In Chen et al. (2019), the number of bits encoded by the smoothing parameters and probability values of a PNN is used as the gene encodes each individual in GA, and the recognition rate of the PNN classifier is used as the fitness function, using GA to optimize the parameters to obtain the circuit structure with both the correct rate and the low memory resource consumption. While ensuring that the correct rate is not affected, GA is used to search for the optimal parameters of the PNN and establish a look-up table method for nonlinear functions to simplify the complexity of the hardware architecture, reduce the use of logic gates (the Altera MAX 10 device was used for simulation), and increase the operation speed. This work provides new insights to utilize evolutionary algorithms in a Bayesian computing platform to optimize the rules and consequently improve the hardware efficiency.

Probabilistic Artificial Neural Network for Analyzing Transistor Process Variation

Line-edge-roughness (LER) is a process-induced random variation source that causes undesirable random variation in the performance of transistors such as metal oxide semiconductor field effect transistor (MOSFET), fin-shaped field effect transistor (FinFET), and gate-all-around field effect transistor (GAAFET). LER can be analyzed with technology computer-aided design (TCAD), which is fundamentally very time-consuming. A machine learning-based method to solve this issue is proposed in Lim et al. (2021), which predicts the LER variations in FinFETs, through which LER parameters (i.e., amplitude and correlation length X, Y) are provided as inputs for an ANN. ANN predicts seven parameters: off-state leakage current (Ioff), saturation drain current (Idsat), linear drain current (Idlin), low drain current (Idlo), high drain current (Idhi), saturation threshold voltage (Vtsat), and linear threshold voltage (Vtlin). To this end, a 3-D quasi atomistic model for LER was used. FinFET was simulated with MATLAB and TCAD by applying the mentioned parameters and the two-dimensional autocovariance function. Considering that the performance metrics of transistors approximately follow Gaussian distribution is not applicable due to non-ideal effects (short-channel effects in transistors) and the different distribution shapes for each LER parameter. Hence, the mixture of multivariate normal distributions (MVN) is used during the training process. Negative log-likelihood (Negloglik) was used as a loss function [see Equation (24)] instead of mean-squared error since, during the training, the weight matrices and bias vectors of ANN are updated for the given layer attached to output neurons returning the PDF of variables. The training process is run to minimize this loss function; hence, training ANN becomes the process of maximum likelihood estimation.

$$\boldsymbol{Negloglik(P,Q)} = -\sum_{k} \boldsymbol{P(x)logQ(x)} \tag{24}$$

In Equation (24), $P(x)$ and $Q(x)$ stand for the PDF of observation and hypothesis, respectively. The proposed ANN models have reduced the simulation time by ∼6 times and can pave a new road to analyzing the impact of LER to overcome the timely design process *via* simulating the electrical behavior of the transistor as well as DC behavior of critical digital circuit blocks in processors such as SRAM bit cells.

HARDWARE IMPLEMENTATION OF PROBABILISTIC SPIKING NEURAL NETWORKS

In this section, employing Bayesian features in SNN is represented, in which the feasibility of nonvolatile devices as synapses in SNN architectures will be first discussed for Bayesian-based inference algorithms. Then, a scalable sampling-based probabilistic inference platform with spiking networks is explained. Afterward, a probabilistic spiking neural computing platform with MTJs is explained. The high learning capability of a probabilistic spiking neural network implementation and utilization of the probabilistic spike propagation mechanism are described. At the end of this section, memristor-based stochastic neurons for probabilistic computing and Loihi-based Bayesian inference implementation are discussed.

Bayesian Inference Implementation in Spiking Neural Networks With Memristor Synapses

Memristors are another type of nonvolatile (i.e., the device could save its state when there is no voltage source) memory devices. They are promising circuit elements that mimic the functionality of biological synapses in a neuromorphic computing system (W. Burr et al., 2017). Their resistance can be tuned based on the spike-timing dependent plasticity (STDP) rule, which is based on the spike timing differences of the pre- and postsynaptic neurons. There are practical challenges in the fabrication of reliable nanoscale memristors. In order to address these challenges, an alternative approach is proposed to use the compound memristive synapse model, where M bistable memristors in parallel model a synapse (Bill and Legenstein, 2014) with a total weight of:

$$\boldsymbol{W_{ki}} = \omega.\boldsymbol{m_{ki}} \tag{25}$$

A compound synapse provides $M+1$ discrete weight levels from 0 to the maximum level $W_{\max} = \omega \cdot M$, where $m_{\mathrm{ki}} \in$

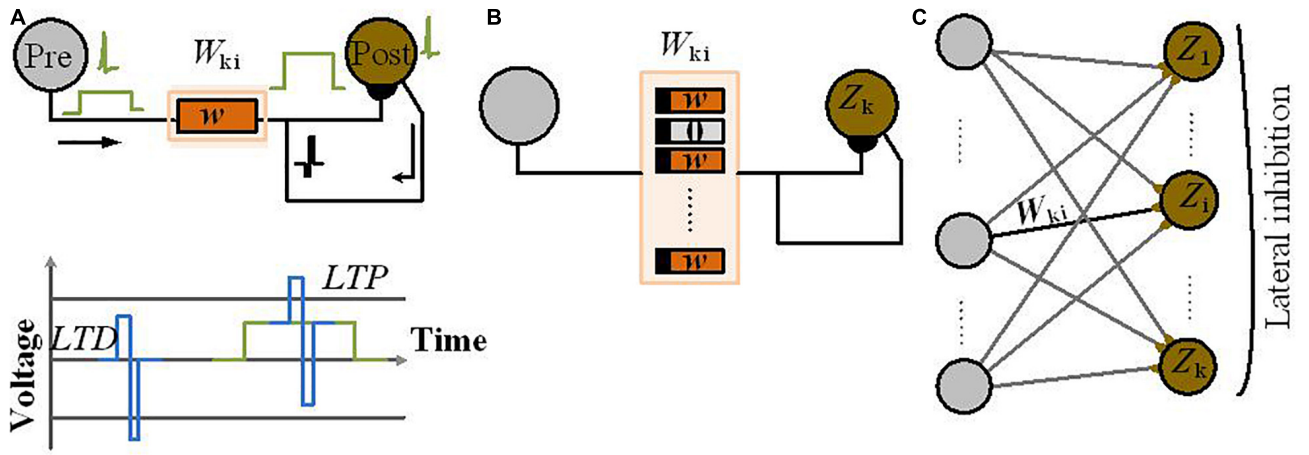


FIGURE 13 | (A) STDP pulsing scheme along with pre- and postsynaptic spikes behavior for LTP and LTD phenomena. **(B)** Stochastic memristors construct the compound memristive synapse. **(C)** The winner-take-all mechanism in SNN with compound memristive synapses.

{0, 1,..., M} in Equation (26) represents the number of active memristors. The weight change of the compound memristive synapse is controlled by pre- and postsynaptic activity. An input pulse (**Figure 13A**) of the ith input is defined by $y_i(t) = 1$ [and no presynaptic pulse by $y_i(t) = 0$] and t^{fk} denotes the spike time of the f^{th} spike of postsynaptic neuron Z_k. A neuron Z_k generates a spike train $S_k(t)$ represented as the sum of Dirac delta pulses $\delta(\cdot)$ at the spike times: $S_k(t) = \sum_f \delta(t - t_{fk})$. When a synaptic W_{ki} is subject to a stochastic long-term potentiation (LTP), where the presynaptic neuron spikes before the postsynaptic neuron, there are $(M - m_{ki})$ inactive memristors (**Figure 13B**). Each memristor independently turns into its active state with probability π_{up}, hence contributing ω to the W_{ki}. Thereby, the weight change for the LTP condition is equal to $(M - m_{ki}).\omega.\pi_{up}$. A similar argumentation applies to the long-term depression (LTD) case, where the post neuron spikes first (before the presynaptic neuron). Note that LTP (LTD) occurs when the presynaptic pulse equals $y_i(t) = 1$ [$y_i(t) = 0$], respectively. Then, the weight change of the compound memristive synapse is:

$$< \frac{d}{dt} W_{ki} > = S_{k(t)} \cdot [\underbrace{(M - m_{ki})\omega\pi_{up} y_i(t)}_{LTP} - \underbrace{m_{ki}\omega\pi_{down(1-y_i(t))}}_{LTD}] \quad (26)$$

Compound memristive synapses with the STDP property have been employed in winner-take-all (WTA) (Wang et al., 2019) networks to provide stochastic learning capability from a Bayesian perspective as an unsupervised model optimization with the expectation-maximization method (Bill and Legenstein, 2014). As shown in **Figure 13C**, N spiking input neurons, y_1,..., y_N, and K spiking network, Z_1,..., Z_K, construct the WTA network. In the WTA network, the forward synapses provide all-to-all connectivity and the network neurons perform lateral inhibition in which the network neurons are competing with each other to fire.

Network neuron Z_k, with the membrane potential u_k, integrates the inputs $y_i(t)$ and the linear membrane potential can be implemented with leaky integrators (a common neuron model in neuromorphic computing paradigm). The neurons Z_k have a stochastic firing rate $\rho_k(t)$ and spike in a Poissonian manner. $\rho_k(t)$ defined by Equation (27), is a function of the membrane potential $u_k(t)$ and lateral inhibition $u_{inh}(t)$.

$$\rho_{k(t)} = r_{net} \cdot e^{u_{k(t)} - u_{inh(t)}} \quad (27)$$

The r_{net} constant scales the overall firing rate of the network. The lateral inhibition contribution $u_{inh}(t) := \log \int_{j=1}^{k} \exp(U_j(t))$ depicts WTA competition among the network neurons to fire over a given stimulus $y_1(t)$,..., $y_N(t)$.

When one of the network neurons, Z_k, fires, the probability distribution $P_{net}(Z \mid Y)$ represents the network response that is proportional to the firing rate $\rho_k(t)$ of neuron Z_k:

$$P_{net}(Z_{k=1} | Y = y(t)) = \frac{\rho_{k(t)}}{r_{net}} = e^{u}_{k(t)} - u_{inh}(t) = \sum_{j=1}^{k} e^{b_{j(t)} + \sum_{i=1}^{N} W_{ij} \cdot y_{i(t)}} \quad (28)$$

Claiming that the response distribution $P_{net}(Z \mid Y)$ provides a Bayesian performance is valid, by considering input $y(t)$ as the observation variable and the spike response of a neuron Z_k as the hidden cause. The network is viewed as a generative model with a prior distribution $P(Z)$ over hidden causes Z_k and a set of likelihood distributions $P(Y \mid Z_k = 1)$, one for each hidden cause Z_k.

Maximum likelihood learning finds parameters that bring the implicit distribution $P(Y)$ of the generated model as close as possible to the actually observed input distribution. Likelihood distributions $P(Y \mid Z_k = 1)$ optimized by a WTA circuit with compound-synapse STDP are computed through the product of

the likelihoods of individual inputs:

$$P(Y_i = y(t)|Z_k = 1) = \prod_{i=1}^{N} P(Y_i = y_i(t)|Z_{k=1}) \quad (29)$$

where the likelihood for each individual input y_i is represented by a Gaussian distribution:

$$P(Y_i = y_i(t)|Z_k = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y_{i(t)}-\mu_{ki})^2}{2\sigma}} \quad (30)$$

For the likelihood distributions, μ_{ki} and σ are the mean values and the standard deviation, respectively, and are identified as:

$$\mu_{ki} = \frac{W_{ki}}{W_{max}} = \frac{m_{ki}}{M} \text{ and } \sigma = 1/\sqrt{W_{max}} \quad (31)$$

During online learning, a Mixture of Gaussians (MoG) generative model has been depicted by this probabilistic model of the WTA network, where compound memristive synapses show synaptic weight changes. This property on average causes the lower bound of the log-likelihood function to increase and leads to finding a local optimum. After training, Bayesian inference for the hidden causes based on the given input observation is employed; simulations have shown that even only four bistable memristors per synapse are sufficient for applications such as reliable image classification.

In hardware implementations of the WTA network, capacitors and other circuit elements have been used to implement the stochastic neurons, while synaptic weights are represented by conductance of compound memristors.

There are several challenges in this approach, mentioned below, which need further study:

- More complex inputs and plasticity mechanisms are needed to support a versatile STDP pulsing scheme; to this end, utilizing memristors with more than two stable states are required.
- Other arbitrary patterns of the input signal (y(t)) for compound memristor synapses are required to depict a clear picture of the Gaussian likelihood distributions $P(Y \mid Z)$, which has the capability of performing inference over arbitrary real-valued input states.
- In compound memristor synapses, the switching probability ($\pi_{up} \cdot W_{max}$) could be considered as the learning rate during online learning. This learning rate controls the number of samples of the input history of the implicit generative model. The number of samples is dependent on the size of the dataset. When the dataset is complex, it relies on small learning rates, i.e., on small switching probabilities, To achieve sufficiently small switching probabilities, it needs some remedies in hardware integration by using control peripherals.

Scalable Sampling-Based Probabilistic Inference With Spiking Networks

The BrainScaleS platform (Schemmel et al., 2010), a physical-model neuromorphic device, emulates networks of spiking neurons. This platform is a mixed-signal neuromorphic system, using 180-nm CMOS technology for fabrication, on which Kungl et al. (2019) proposed the first scalable implementation of sampling-based probabilistic inference with spiking networks. In order to sample from target distributions and hierarchical spiking networks with higher-dimensional input data, fully connected spiking networks have been trained. Similar to systems that operate in biological real time, it provides a higher acceleration factor that shows the advantages of brain-inspired physical computation and maintain main building blocks for large-scale neuromorphic applications. Moreover, by co-embedding the stochasticity within the same substrate, the feasibility of a fully embedded neural sampling model with highly reduced demands on off-substrate I/O bandwidth has been shown, where having a fully embedded implementation allows the runtime of the experiments to scale as O(1) with the size of the emulated network.

The most notable limitation of the BrainScaleS system for this application was the size of the emulated spiking sampling network (SSNs). The maximum connectivity is limited (synapse loss) between different locations within the area, due to limited software flexibility, system assembly, and substrate yield; hence the applicable hardware real-estate was limited to a patchy and non-contiguous area. In order to write analog parameters, significant trial-to-trial variability for any given trail is needed, which leads to a heterogeneous substrate and a low sampling accuracy. The ability of the SSN to approximate target distributions has been hindered since the symmetry in the effective weight matrix is imperfect (due to analog variability of the synaptic circuits) and the resolution of the synaptic weights is low. Hence the "jumping" behavior between approximate and target distribution in the final stages of learning has been seen. Moreover, as the underlying neuron and synapse are deterministic, for a more biologically plausible implementation, one needs to consider stochastic neurons such that the framework can be extended to sampling from arbitrary probability distributions rather than only binary random variables.

Probabilistic Spiking Neural Computing Platform With Magnetic Tunnel Junctions

In Sengupta et al. (2016), by enabling the neural computing unit with the stochastic switching behavior of an MTJ, the implementation of a deep SNN has been explored for high-accuracy and low-latency classification tasks and provided an energy improvement of 20× over a baseline CMOS design in 45-nm technology. Despite the huge success at complex recognition problems due to the high computational costs needed for training and testing of deep ANNs, researchers are motivated to develop alternative computing models; therefore, more biologically realistic SNNs have been introduced. In SNNs, information is transferred between the neural nodes as spikes rather than real-valued analog signals. Spiking networks exploit the prospects of event-based computing

which lead to the development of specialized custom hardware implementations. Sengupta et al. (2016) discusses that the technologically mature spintronic devices, such as the MTJ (being binary switching devices), with variation in the magnitude of the input current showing switching probability variation similar to the sigmoid function. An ANN-to-SNN conversion scheme has been proposed utilizing the sigmoid function like switching probability of MTJs, and assuming that the neural units generate spikes depending on a probability density function (similar to the original ANN transfer function). It has been proved that such a conversion mechanism approximates the original ANN functionality to a reasonable degree of precision, potentially paving the way for probabilistic neuromorphic platforms that employ the variability and inherent stochasticity of emerging neuromagnetic devices. Nonvolatile emerging devices based on a probabilistic neural computing platform that models complex neural transfer functions in the time domain provide high-accuracy energy-efficient cognitive recognition platforms over conventional CMOS designs.

High Learning Capability Probabilistic Spiking Neural Network Implementation

Using sequential processors to run algorithms, there is a struggle to simultaneously fulfill learning speed, learning performance, power consumption, and area requirements in portable and biomedical applications. Hence, hardware-implemented neural networks are used extensively and even though the circuit is implemented using analog very-large-scale integration (VLSI), variations in sensor fabrication, background noise, and human-dependent parameters complicate the restrictions on power consumption and area. One type of neural network that comprises spiking neurons with probabilistic parameters is called the probabilistic spiking neural network (PSNN). These PSNNs are hardware-friendly and compare with deterministic neural networks in hardware compatibility. PSNNs have relaxed weight resolution requirements and are insensitive to noise and analog process variation. A PSNN does not suffer from multiplicative linearity. In the spiking neuron model, the presynaptic spike of a neuron can be considered as a control signal, and the weight controls the postsynaptic current. As a result, when a presynaptic spike stimulates a neuron, the post synapse generates a current. In Hsieh et al. (2018, 2017), an analog implementation of PSNNs has been proposed for biomedical applications through which online learning adjusts weights by spike-based computation. The weight is saved in the long-term synaptic memory. Switched capacitor circuit structures have been utilized for the implementation of most of the circuits to provide low-power consumption and a small area and consequently provide high learning performance. This learning chip was fabricated in 0.18-μm CMOS technology and can process the e-nose and electrocardiography (ECG) data, yielding comparable accuracy to the simulated accuracy that indicates that the learning chip can be employed into portable and implantable devices, to facilitate convenient use and intelligence. This hardware implementation opens up new windows to achieve efficient portable and biomedical devices *via* utilizing PSNNs.

Hardware Implementation of Spiking Neural Networks Utilizing Probabilistic Spike Propagation

As mentioned in *Section* "Bayesian Inference Implementation in Spiking Neural Networks With Memristor Synapses," SNNs provide intrinsic desirable attributes where information is represented as discrete spike events that provide an event-driven paradigm of computation. SNNs are implemented on low-power event-driven hardware, and the time and energy consumption are proportional to the number of spike events. When processing a spike, SNNs do not require multiplication to be performed and hence provide a reduced hardware complexity compared to conventional ANNs; as a result, SNNs are not well-suited to be implemented on hardware platforms like GPUs. Spiking networks still need a large number of memory accesses although they are event-driven. It is necessary to know the fanout neurons of a spiking neuron, which determines the connectivity information that needs to be fetched along with the weights of the corresponding synapses. Then, the membrane potentials of the fanout neurons are fetched and updated. Defining techniques for reducing the number of memory accesses in SNNs is necessary for improving their energy efficiency since data fetching from memory is more expensive than arithmetic computations. The spiking activity that is measured as spike propagation along a synapse from a single source neuron to a single target neuron has a strong role in the complexity of an SNN. Nallathambi et al. (2021) introduce an approach that is named probabilistic spike propagation to optimize rate-coded SNNs. In this approach, synaptic weights are represented as probabilities, and these probabilities are utilized to regulate spike propagation. The approach reduces the propagated spikes, which cause a reduction in time and energy consumption. To this end, an SNN accelerator named probabilistic spiking neural network application processor (P-SNNAP), which supports probabilistic spike propagation, has been represented, where a probabilistic method for spike propagation to reduce the number of memory accesses in rate-coded SNNs has been proposed. This method would save both runtime and energy. The proposed probabilistic spike propagation mechanism has been realized through probabilistic synapses shown in **Figures 14A,B**. Conventionally, the weight of a synapse determines the amount by which the potential of postsynaptic neuron membranes increases whenever presynaptic neuron spikes. This weight defines how likely it is that a spike will propagate across the synapse (**Figures 14A,B**). A probabilistic synapse does not propagate all spikes to the postsynaptic neuron. Instead, only a subset of its outgoing synapses propagate the spike (which has weights above a certain threshold) as neuron spikes. P-SNNAP is shown in **Figure 14C**. The P-SNNAP architecture consists of three different modules, the Spike Neural Processing Element (SNPE), the Eval unit, postsynaptic spikes, the weight memory that stores the weights, and the state memory that stores the neuronal state variables. The Eval unit performs neuron

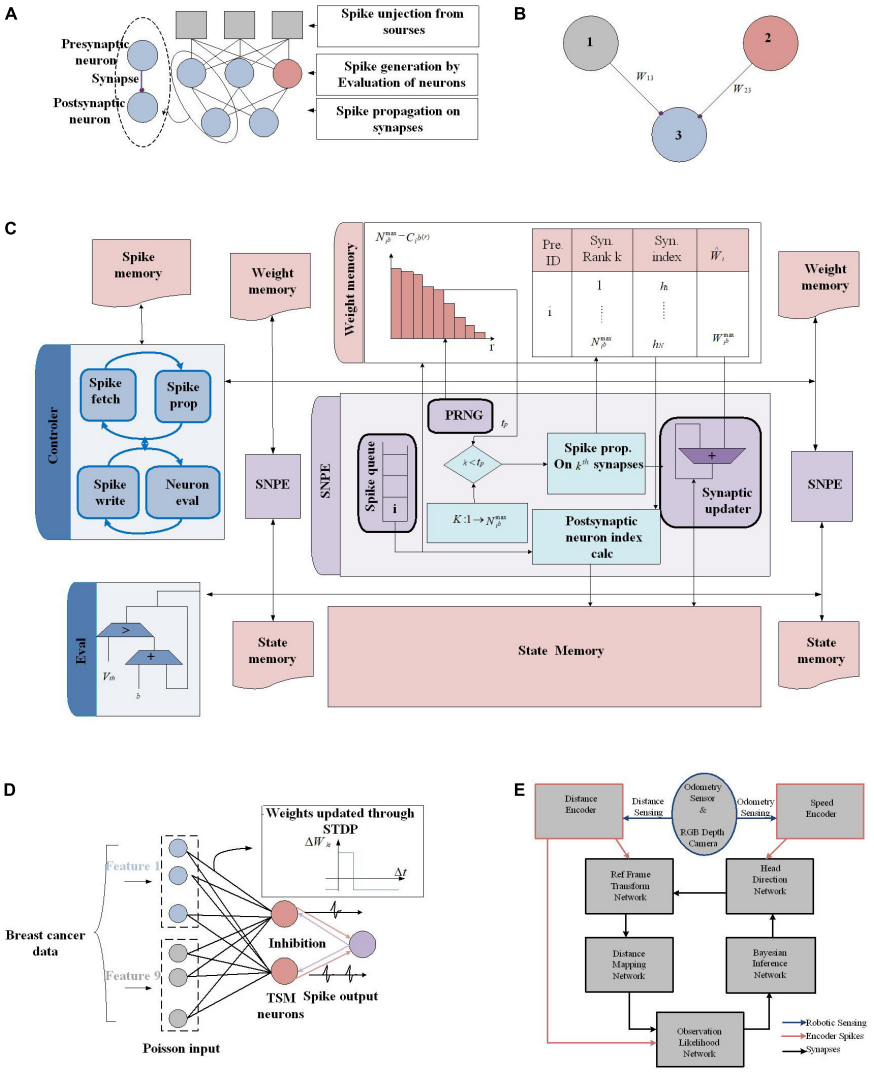


FIGURE 14 | (A) A generic structure of spiking neural networks. **(B)** Neuron 3 receives spikes from Neuron 1 and Neuron 2. **(C)** P-SNNAP accelerator architecture in Nallathambi et al. (2021). **(D)** Stochastic neuron based probabilistic spiking neural network implantation for the uncertainty quantification problem in medical diagnosis (Wang et al., 2021). The probabilistic SNN consists of the input encoding layer, the output layer, and the inhibitory layer. Neurons mimic the biological neurons behavior. Each feature of breast cancer data is encoded by the firing rate of a small population of Poisson neurons. The inset depicts the STDP learning curve of synapses connecting input neurons to the output ones; if the post-spike falls within the time window after the pre-spike, then the synaptic potentiation will occur. **(E)** Overall structure of the implemented SNN architecture on the Loihi processor (Tang et al., 2019).

evaluation. Eval unit brings membrane potentials from state memory, increases it with bias value, and compares it to the threshold. If the membrane potential goes above the threshold, a spike is generated and communicated to the controller. The controller in its first phase of operation controls the SNPEs and that in the second phase controls the Eval unit. When a layer is evaluated by Eval, the controller brings spikes from the previous layer and sends them to SNPEs. As a spike is received, an SNPE uses the index of the spiking neuron to iterate through its outgoing synapses. The SNPE calculates the index of the postsynaptic neuron for each synapse. Next, the membrane potential and the weight of the corresponding synapse are fetched for each postsynaptic neuron. Then, membrane potential is updated and written back. All the information that is required to perform probabilistic spike propagation is stored in weight memory in each SNPE lane. The register transfer level was used for the P-SNNAP engine design and the Nangate 15-nm technology was used for synthesizing in Synopsys Design Compiler platform. It has been observed that the proposed probabilistic approach causes a logic area and logic power overhead of 12 and 23.5%, respectively, over a version of SNNAP without support for probabilistic spike propagation. In this work, it has been shown that the temporal nature of SNNs allows the network to regain any accuracy loss caused by this approach. Evaluating alternative synaptic propagation mechanisms and employing larger networks to test the scalability of the proposed accelerator turn out to be further explored.

Memristor-Based Stochastic Neurons for Probabilistic Computing

Stochastic firing mechanism in biological neurons (rather than giving out spikes once reaching a fixed threshold voltage) provides dynamic excitation behavior and allows the brain to perform probabilistic inference in the face of uncertainty. However, due to the complexity of the stochastic firing process, fabrication of stochastic neurons with bio-realistic dynamics to probabilistic scenarios is challenging and needs further study. In Wang et al. (2021), a stochastic neuron has been fabricated based on CuS/GeSe threshold switching memristor (TSM) and applied to implement Bayesian computing in a PSNN that can quantify uncertainty with incomplete or inaccurate data. The experimental results have indicated that compared to VO_2 and $GeTe_6$, which are typical metal-insulator transition (MIT) and ovonic threshold switching (OTS), CuS/GeSe as a conductive-bridge TSM shows the most appropriate randomness of threshold switching as desired by the stochastic firing of neurons. The proposed physical modeling and simulation have revealed that this can be attributed to the similarity between the ion motion tuning in conductive-bridge threshold switching and in biological neurons. In particular, the positive feedback process of Cu electromigration enhanced Joule heating and temperature and thereby accelerated thermal diffusion of Cu, substantially facilitating the formation of the conductive bridge and the stochasticity of ion motion, which leads to the desired variation of threshold voltages. The intrinsic random formation of the Cu conductive bridge in the device is utilized to emulate the stochasticity of the opening of ion channels in the biological membrane. Moreover, the random switching parameters of the device fulfill the requirement to achieve the stochastic neurons in a PSNN. Utilizing the stochastic firing properties of the fabricated CuS/GeSe neuron to a probabilistic SNN is shown in **Figure 14D**. This probabilistic SNN is capable of giving superior prediction on a typical probabilistic inference problem, namely, breast cancer diagnosis with high diagnostic accuracy, and improves the fidelity of the judgment compared to deterministic neuron-based SNN. Moreover, the stochastic neurons enable the SNN to estimate the uncertainty of predictions, a feature that will be of great help for achieving a good balance between diagnostic accuracy and medical cost and avoiding the fatal diagnostic misclassification error often encountered by conventional ANNs. The software synapses used in this demonstration can be achieved by non-volatile memristors, indicating the possibility of implementing a fully memristive probabilistic SNN in the near future. Utilizing developed and optimized stochastic neurons and their powerful application in uncertainty quantification problems open up a new horizon of probabilistic computing in neuromorphic computing systems.

Loihi-Based Bayesian Inference Implementation

Through asynchronous computations and event-based communications in a network of neurons, the brain solves simultaneous localization and mapping (SLAM) while it consumes very low energy; as Tang et al. (2019) show, SNNs (which are famous for mimicking this computational paradigm of the brain) can be used to solve SLAM problems on energy-efficient neuromorphic hardware for mobile robots exploring unknown environments. The proposed SNN shown in **Figure 14E** is integrated into Intel's Loihi neuromorphic processor fabricated on 14-nm FinFET technology (Davies et al., 2018). Loihi is a non-Von Neumann hardware mimicking the brain's computing paradigm and is optimized for SNN computations and online learning algorithms (Thakur et al., 2018).

Via multisensory cues (called visual and odometry information) to implement spike-based recursive Bayesian inference, Tang et al. (2019) proposed a model to determine the robot's heading. To perform head direction localization and mapping, the recursive SNN suggests a cue-integration connectome on Loihi. The head direction and border cells in the network provide biologically realistic performance; thus, to implement them, the proposed model utilizes spiking neurons, multi-compartmental dendritic trees, and plastic synapses, each of which is implementable by Loihi. The model has two sensory spike rate encoders and five Bayesian networks. The odometry sensor drives the neural activity of speed cells, which encodes the angular speed and the RGB Depth camera drives the neural activity of sensory neurons, which encodes the distance to the nearest object. The head direction (HD) network defines the heading of the robot *via* receiving the input from the speed cells. The reference frame transformation (RFT) network generates allocentric distance representation *via* the HD network by getting its input from sensory neurons. The RFT network sends the allocentric observations to the distance mapping (DM) network and the DM network develops the map of the robot's surrounding environment. The DM network sends its information to the observation likelihood (OL) network, which calculates the observation likelihood distribution of the robot's heading. The Bayesian inference network through the utilization of the observation likelihood from the OL network and the odometry likelihood from the HD network provides an optimal posterior of the robot's heading and corrects the heading representation within the HD network.

Note that each one of the networks is implemented on Loihi; here, the Bayesian inference network block is explained based on Equation (32).

$$P(s|d,o) \propto P(d|s)P(o|s)P(s) \tag{32}$$

where s, d, and o denote the heading of the robot, the observed distance, and the odometry sensing, respectively. With a flat prior $P(s)$, the posterior distribution over the robot's heading is proportional to the product of $P(d|\ s)$ and $P(o|\ s)$, the two likelihood functions.

Having known that multiplying two Gaussian distributions generates another Gaussian distribution, Tang et al. (2019) have employed likelihood distributions represented by the OL network and the HD network to predict the posterior distribution. Dendritic trees have been used for implementation; specifically, each Bayesian neuron has two dendritic compartments connected with its corresponding OL neuron and HD cell.

Results of Loihi-based SNN architecture implementation show that it consumes 100 times less energy than conventional GMapping (a common algorithm for SLAM solving) running on a CPU. This provides a motivation to use Loihi as a hardware implementation platform for Bayesian inference. The FinFET technology used in the Loihi architecture is a promising technology in terms of energy and speed over conventional CMOS technology (Bagheriye et al., 2016), while the use of emerging nonvolatile technologies attracts a lot of attention to developing ultra-low energy computing platforms for SNN-based Bayesian inference systems (like crossbar arrays discussed in *Section "Crossbar Arrays for Bayesian Networks Implementation"*). However, the fabrication of robust nonvolatile devices and large-scale crossbar arrays probably require a lot more insights before they can outperform already highly developed technology and this approach is worth exploring.

DISCUSSION

In this paper, we have attempted to review and summarize the recent hardware developments for Bayesian inference. The review is centered on different possible hardware implementations considering algorithmic aspects. Different approaches and their principles have been discussed with extensive references quoted. We review the pros and cons of the approaches reported in the literature. Specifically, there are a number of challenges to be further studied before valid and robust models can be applied to practical systems. We summarize them as follows.

- In asynchronous implementation of Bayesian networks with spintronic devices, updating the network as well as dealing with variations in the thermal barriers or interconnect delays necessitates further study.
- In abstraction layer-based implementations based on the number of linearly independent equations, the appropriate number of auxiliary variables is needed; it would be challenging for a large Bayesian network and would add extra area and energy overhead, which requires further investigation.
- More complex inputs and plasticity mechanisms are needed to support a versatile STDP pulsing scheme *via* using memristors with more than two stable states as synapses to have biologically plausible Bayesian inference in SNNs. Other arbitrary patterns of the input signal for memristor synapses in SNNs is required to depict the clear picture of the Gaussian likelihood distributions that have a capability of performing inference over arbitrary real-valued input states. In memristor synapses, in SNN, the switching probability considered as the learning rate during online learning must be controllable since, for complex datasets, small learning rates, i.e., small switching probabilities, are required. Small switching probabilities need careful remedies in hardware integration by using control peripherals.
- In analog neuromorphic substrates like the BrainScaleS platform, due to limited software flexibility, system assembly, and substrate yield, the maximum connectivity between different locations is strongly limited; hence, post-production, assembly, and the mapping and routing software needed careful consideration to enhance on-wafer connectivity and thereby automatically increase the size of emulable networks, as the architecture of the SSNs. Moreover, approximation of the target distributions is hindered due to the limited synaptic weight resolution and the imperfect symmetry in the weight matrix (due to analog variability of the synaptic circuits). As a result of the successor system, a new generation of scalable platforms is needed to be designed with a higher weight resolution.
- Providing accurate digital encoding where Bayesian network representation is mapped directly (without any abstraction layer) to S-MTJ resistance with equivalent digital voltage representation using arithmetic composers is promising, whereas PSL needs an abstraction level to map Bayesian networks in hardware. To this end, using accurate encoding is required to achieve the required resolution.
- For the structure learning process of Bayesian learning, hardware acceleration *via* FPGA like system implementation is promising, since the runtime for Bayesian network inference has been highly reduced. This property attracts more attention to structure learning acceleration and could be a promising field to be studied utilizing emerging nonvolatile devices.
- In digital encoding of probabilities, the small margin input voltage is highly problematic when it generates the output probability. DACs with high precision are needed for precise mapping from digital probabilities to voltages. In addition, tackling the nonlinear relationship between probabilities and voltages is difficult and a slight noise or process variation may translate a probability to a wrong voltage value. The relation between probability and voltage is not very smooth as a result of the stability of the SBG, which needs improvement. Although the scale of hardware can be reduced, the reduction of the scale of the Bayesian inference system is also worth exploring. In addition, the resolution is limited since every storage method adds a resolution limitation; to this end, utilizing nonvolatile nanomagnets is promising to overcome the power consumption as well as increasing weight resolution by multi-state memristors as synapses.
- The C-element (a standard cell-based implementation) outputs a stochastic bitstream, which is probabilistic and converging more slowly toward the exact Bayesian inference. In this case, if the switching rate of the output was low, the longer "domains" of consecutive "0"s and "1"s are needed and it leads to a more imprecise bitstream and adds time, energy, and area overhead. On the other hand, mixed-signal implementations needs to pay attention to noise and variation sources as well as examining the multiple independent sources of evidence for embedded decision circuits that require circuit design remedies.

- Constructing circuits for approximate inference in hierarchical Bayesian models is a challenging research field that can be *via* merging stochastic samplers with stack-structured memories and content-addressable memories.
- The brain-inspired hardware implementation of algorithms like NB algorithm provides insights for techniques like mean-field approximation, which will help to find an optimal balance between structure and independence, using hardware feasibility considerations and independence assumptions as mutually constraining objectives, which can be a promising research field.
- To provide high-speed stochastic simulations (in which samples of random variables in a Bayesian network are drawn to determine the posterior probabilities), a variety of algorithms with higher sampling efficiency in Bayesian graphs are required since they still fall short of the escalating pace and scale of Bayesian network-based decision engines in many IoT and CPS.
- While providing conditional probability between two variables *via* utilizing two MTJs (or other nonvolatile devices) in a common substrate, the spacing between the MTJs needs to be carefully defined to have significant dipole coupling between the two soft layers. Moreover, in the presence of thermal noise at room temperature, the "flipping" is stochastic, which needs to be controlled with peripheral circuit elements and accurate timing.
- With a higher degree of process variability, prediction error for the probability of a variable is high. Tolerance to process variability needs to be increased by circuit innovations as well as post-fabrication calibration. Adapting ultra-energy-efficient nanomagnetic devices to stochastic/probabilistic computing, neuromorphic, belief networks (non-Boolean computing and information processing) has resulted in rapid strides in new computing paradigms, especially Bayesian networks that may experience revolutionary advances.
- In autonomous systems like self-driving cars, decision-making is based on uncertainty; hence, employing AI platforms is crucial. The standard supervised backpropagation-based learning techniques do not represent uncertainty in the modeling process to solve this issue; Bayesian deep learning plethora is required where a probabilistic framework following the classic rules of probability, i.e., Bayes' theorem, has been utilized to train the DNNs.
- In a standard deep learning architecture during the inference, the dot-product operation between the synaptic weights and inputs involves the compute energy along with memory access and memory leakage components. In a Bayesian deep network, each synaptic weight uses double memory storage since it is represented by two parameters (mean and variance of the probability distribution). Moreover, the dot-product operation does not occur directly between the inputs and these parameters since for each inference operation the synaptic weights are repeatedly updated depending on sampled values from the Gaussian probability distribution. Hence, direct utilization of crossbar-based "In-Memory" computing platforms utilizing non-volatile memory technologies for mitigating the memory access, leakage, and memory fetch bottlenecks is not feasible; thus, a significant rethinking is necessary.
- Despite the specialized custom hardware and brain-inspired possibility of SNNs due to their event-based computing feature, their training for recognition problems has been mostly limited to single-layered networks. On the other hand, Bayesian techniques are more computationally expensive, thereby limiting their training and deployment in resource-constrained environments. Also, the standard von-Neumann bottleneck in current deep learning networks (where memory access and memory leakage can account for a significant portion of the total energy consumption profile) motivates further research in hardware implementation of multi-layer probabilistic SNN and is a promising research field.
- In Bayesian neural networks, Gaussian random number generation operation is a hardware expensive task for CMOS-based designs since a large number of registers, linear feedback circuits, etc. are required. To overcome this issue, non-idealities and stochasticity prevalent in RRAM, spintronic, and other nonvolatile technologies could be extensively exploited to this end.
- Stochasticity provides computational features like regularization and Monte Carlo sampling in a DNN where such normalization features reduce internal covariate shift obtaining an alternative process for divisive normalization in bio-inspired neural networks. Hence, employing the inherent weight normalization feature exhibited by a stochastic neural network using nonvolatile devices is a promising field where it is an online alternative for used batch normalization and dropout techniques. Saturation at the boundaries of fixed range weight formats as well as spurious fluctuations affecting the rows of the weight matrix have been mitigated.
- Despite the widespread applications and simplicity of PNNs, their hardware implementation is relatively underrepresented. This is due to the fact that multicomponent digital CMOS circuits that cause severe area and energy inefficiency are required for hardware implementation of probability functions associated with the PNNs, such as the Gaussian. Hence, utilizing emerging nonvolatile technologies to make use of their biological plasticity features of conductance level changes as well as their energy efficiency would be a promising solution that needs to be extensively explored.
- Uncertainty serves as an intrinsic part of neural computation through which probabilistic computing empowers the brain to analyze sensory stimuli, produce adequate motor control, and make reasonable inferences. On the other hand, quantifying uncertainty is especially crucial for error-critical applications like medical diagnostics, which require probabilistic SNN-based neuromorphic computing systems. The recent literature of electronic neurons for SNN implantation is mostly

focused on deterministic neural units or emulating the complex biological neuronal functions, ignoring the demand of building intrinsically stochastic neurons. Hence, developing probabilistic spiking neurons with low area and power consumption is highly required.
- To foster the neuromorphic computing systems, not only is the mature device fabrication process required but also hardware friendly algorithms are inevitable, To this end, one promising approach that needs further exploration is utilizing evolutionary algorithms in a Bayesian computing platform to optimize the rules.

To conclude these observations, we can state that the pace of the development of efficient hardware implementation of Bayesian networks has been very quick in recent years, but there is still a long way to go to overcome the challenges outlined above. To summarize, comparing the discussed implementations shows that probabilistic hardware-based implementation of Bayesian networks, with nonvolatile devices, needs more attention to solving the scaling issues in Bayesian network hardware; also, sequential signaling from parent to child nodes, controlling the stochastic switching variation due to thermal noise and process variation, defining an abstraction layer, utilizing axillary nodes, employing complex input pattern for memristor synapse, and multi-state memristors for WTA mechanism are required. For NSMs for approximate Bayesian inference, providing technologically mature nonvolatile devices to solve the scaling issue in crossbar arrays on one hand and adding noise to provide uncertainty on the other hand are challenging tasks. Utilizing nonvolatile memory elements for Bayesian network implementation *via* digital encoding needs a high-resolution encoding mechanism to provide readily highly scaled FPGA-like architectures not only for inference but also for learning Bayesian network structure. To this end, utilizing multi-state memristors rather than two-state spintronic-based devices would provide higher resolution with a lower area overhead.

Bayesian inference hardware implementation employing digital logic gates in state-of-the-art FPGA platforms, defining novel stochastic logic gates, and utilizing standard cells needs to solve the accuracy and resolution issue of digital bitstreams, which needs to compromise speed, power, and area overhead. Crossbar arrays for Bayesian network implementation require some innovation where providing a hierarchy of crossbar arrays for approximate Bayesian inference mechanisms like mean-field approximation, taking inspiration from naïve Bayesian classifiers, is promising. Crossbar arrays require solving scaling issues while they act as Bayesian reasoning machines. Utilizing crossbar arrays in PNNs for dot-product operation needs serious rethinking while utilizing approximate inference rules. Moreover, utilizing platforms like BrainScaleS or Loihi is another option for Bayesian inference while the resolution and scaling, as well as energy consumption, need to be considered since these platforms are utilizing mixed-signal CMOS and FinFET technologies, respectively, rather than energy-efficient nonvolatile technologies.

CONCLUSION

A Bayesian network provides a simple way of applying Bayes theorem to complex problems and Bayesian inference is crucial for statistical machine learning, causal discovery, automatic speech recognition, email spam filtering, and clinical decision support systems, to name just a few applications in AI. However, Bayesian inference is an NP-hard problem even when only an approximate solution is sought, implying that this computational problem scales badly, which hinders further progress in AI. Interestingly, many neuroscientists are convinced that our brains employ similar processes to combine prior knowledge with newly arriving information in an approximately optimal Bayesian fashion. For example, in visual perception, the brain establishes this integration literally in the blink of an eye. However, the brain's energy consumption is orders of magnitudes less than what is required for state-of-the-art AI applications. Bayesian network implementations in conventional processor architectures are problematic due to several issues: (i) software solutions involve multiple layers of abstraction to support a non-deterministic framework such as Bayesian networks; (ii) the inherently separated memory and computation in the von Neumann processor architecture introduces bottlenecks in accessing data; and (iii) the non-volatility requirements in cognitive applications are challenging to meet the efficiency. Moreover, as mentioned, the computational complexity of belief updating is an important issue in Bayesian inference. To enhance the computation speed of Bayesian updating, several techniques such as conjugate priors, variational Bayes, or approximate Bayesian computations have been employed, whereas these are software-based, and their efficacy is less than hardware-based accelerators. Hence, the practical use of Bayesian inference has been hindered in many real-world applications (such as large-scale networks or embedded systems) where computational cost is an important performance factor. This review paper has discussed several implementations of Bayesian inference as well as the implantation of several approximate inference algorithms and different architectures, from FPGA-like to brain-inspired ones (crossbar arrays). FPGA-like architectures are not efficient enough in terms of area and energy overhead when compared to brain-inspired architectures. Crossbar arrays, a typical brain-inspired computing paradigm, lead to efficient computation when the network structure is limited to, e.g., naive Bayes classifiers or tree-like structures. Using insights into Bayesian approximation techniques to find an optimal balance between structure and independence and using hardware feasibility considerations and independence assumptions as mutually constraining objectives are open windows for future efforts to achieve an efficient computing paradigm.

AUTHOR CONTRIBUTIONS

LB and JK conceived the study and the discussions around material of the manuscript. LB wrote the first version of the manuscript then it has been edited by LB and JK. Both authors contributed to the article and approved the submitted version.

REFERENCES

Akhmetov, Y., and Pappachen, J. A. (2019). "Probabilistic neural network with memristive crossbar circuits," in *Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS) (Sapporo)* (Piscataway, NJ: IEEE), 49–52. doi: 10.1109/ISCAS.2019.8702153

Atulasimha, J., and Bandyopadhyay, S. (2013). "Hybrid spintronics and straintronics: a super energy-efficient computing paradigm based on interacting multiferroic nanomagnets," in *Spintronics in Nanoscale Devices*, ed. E. R. Hedin (Boca Raton, FL: CRC), 121–154.

Bagheriye, L., Saeidi, R., and Toofan, S. (2016). "Low power and roboust FinFET SRAM cell using independent gate control," in *Proceedings of the 2016 IEEE International Symposium on Circuits and Systems (ISCAS)* (Montreal: IEEE), 49–52. doi: 10.1109/ISCAS.2016.7527167

Bagheriye, L., Toofan, S., Saeidi, R., and Moradi, F. (2018). "A novel sensing circuit with large sensing margin for embedded spin-transfer torque MRAMs," in *Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS) (Florence, Italy)* (Piscataway, NJ: IEEE). doi: 10.1109/ISCAS.2018.8351577

Bashizade, R., Zhang, X., Mukherjee, S., and Lebeck, A. R. (2021). Accelerating Markov random field inference with uncertainty quantification. *arXiv* [preprint] arXiv:2108.00570

Baum, L. E., and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Ann. Math. Stat.* 37, 1554–1563.

Bessière, P., Laugier, C., and Siegwart, R. (2008). *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*. Berlin: Springer-Verlag.

Bill, J., and Legenstein, R. (2014). A compound memristive synapse model for statistical learning through STDP in spiking neural networks. *Front. Neurosci.* 8:412. doi: 10.3389/fnins.2014.00412

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Kim, S., Sidler, S., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys. X* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Chen, C.-Y., Chang, C.-W., and Chen, Z.-C. (2019). "An evolutionary computation approach for approximate computing of PNN hardware circuits," in *Proceedings of the 2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS). (Taipei, Taiwan)* (Piscataway, NJ: IEEE). doi: 10.1109/ISPACS48206.2019.8986351

Chickering, D. M. (1996). "Learning Bayesian networks is NP-complete," in *Learning from data. Lecture Notes in Statistics*, eds D. Fisher and H. J. Lenz (New York, NY: Springer), 121–130.

Choi, J., and Rutenbar, R. A. (2013). "Video-rate stereo matching using Markov random field TRW-S inference on a Hybrid CPU+FPGA computing platform," in *Proceedings of the 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays: ACM 978-1-4503-1887-7/13/02*, Monterey, CA, 63–71.

Choi, J., and Rutenbar, R. A. (2016). Video-rate stereo matching using Markov random field TRW-S inference on a hybrid CPU+FPGA computing platform. *IEEE Trans. Circuits Syst. Video Technol.* 26, 385–398. doi: 10.1109/TCSVT.2015.2397198

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.* 42, 393–405.

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic many core processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Debashis, P., Ostwal, V., Faria, R., and Datta, S. (2020). Hardware implementation of Bayesian network building blocks with stochastic spintronic devices. *Nature* 10:16002. doi: 10.1038/s41598-020-72842-6

Druzdzel, M. J., and van der Gaag, L. C. (1995). "Elicitation of probabilities for belief networks: combining qualitative and quantitative information," in *Proceedings of the 11th conference on Uncertainty in AI (UAI 1995)*, eds B. Philippe, and S. Hanks (Burlington, MA: Morgan Kaufmann), 141–148.

Dutta, S., Detorakis, G., Khanna, A., Grisafe, B., Neftci, E., and Datta, S. (2021). Neural sampling machine with stochastic synapse allows brain-like learning and inference. *arxiv* [preprint] arxiv:2102.10477

Faria, R., Camsari, K. Y., and Datta, S. (2018). Implementing Bayesian networks with embedded stochastic MRAM. *AIP Adv.* 8:045101. doi: 10.1063/1.5021332

Faria, R., Kaiser, J., Camsari, K. Y., and Datta, S. (2021). Hardware design for autonomous bayesian networks. *Front. Comput. Neurosci.* 15:584797. doi: 10.3389/fncom.2021.584797

Friedman, S., Calvet, J. E., Bessière, L., Droulez, P., and Querlioz, D. (2016). Bayesian inference with muller C-elements. *IEEE Trans. Circuits Syst. I Regul. Pap.* 63, 895–904. doi: 10.1109/TCSI.2016.2546064

Gómez Hidalgo, J. M., Bringas, G. C., Sánz, E. P., and García, F. C. (2006). "Content based SMS spam filtering," in *Proceedings of the 2006 ACM symposium on Document Engineering*, Amsterdam, 107–114.

Gordon, N. J. (1993). Noval approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proc. Radar Signal Process.* 140, 107–113.

Guo, S., Yu, Z., Deng, F., Hu, X., and Chen, F. (2019). Hierarchical Bayesian inference and learning in spiking neural networks. *IEEE Trans. Cybern.* 49, 133–145. doi: 10.1109/TCYB.2017.2768554

Hastings, W. K. (1970). Monte carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 97–109.

Heckerman, D. (2020). A tutorial on learning with Bayesian networks. *arXiv* [preprint] arXiv:2002.00269v2

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: the combination of knowledge and statistical data. *Mach. Learn.* 20, 197–243.

Heckerman, D., Meek, C., and Cooper, G. (1999). A Bayesian approach to causal discovery. *Comput. Causation Discov.* 19, 141–166.

Hsieh, H.-Y., Li, P.-Y., and Tang, K.-T. (2017). "An Analog Probabilistic Spiking Neural Network with On-Chip Learning," in *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, eds D. Liu, S. Xie, Y. Li, D. Zhao, and E. S. El-Alfy (Cham: Springer). doi: 10.1007/978-3-319-70136-3_82

Hsieh, H.-Y., Li, P.-Y., Yang, C.-H., and Tang, K.-T. (2018). "A high learning capability probabilistic spiking neural network chip," in *Proceedings of the 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Hsinchu. doi: 10.1109/VLSI-DAT.2018.8373241

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge: Cambridge University Press.

Ji, Z., Xia, Q., and Meng, G. (2015). "A review of parameter learning methods in Bayesian network," in *Proceedings of the International Conference on Intelligent Computing* (Cham: Springer), 3–12.

Jia, X., Yang, J., Dai, P., Liu, R., Chen, Y., and Zhao, W. (2020). SPINBIS: spintronics-based Bayesian inference system with stochastic computing. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* 39, 789–802. doi: 10.1109/TCAD.2019.2897631

Jia, X., Yang, J., Wang, Z., Chen, Y., Li, H., and Zhao, W. (2018). "Spintronics based stochastic computing for effcient Bayesian inference system," in *Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju, 580–585. doi: 10.1109/ASPDAC.2018.8297385

Khasanvis, S., Li, M., Rahman, M., Biswas, A. K., Salehi-Fashami, M., Atulasimha, J., et al. (2015a). Architecting for causal intelligence at nanoscale. *Computer* 48, 54–64. doi: 10.1109/MC.2015.367

Khasanvis, S., Li, M., Rahman, M., Salehi-Fashami, M., Biswas, A. K., Atulasimha, J., et al. (2015b). "Physically equivalent magneto-electric nanoarchitecture for probabilistic reasoning," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'15)*, Boston, MA. doi: 10.1109/NANOARCH.2015.7180581

Ko, G. G., and Rutenbar, R. A. (2017). "A case study of machine learning hardware: real-time source separation using Markov Random Fields via sampling-based inference," in *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA. doi: 10.1109/ICASSP.2017.7952602

Korb, K. B., and Nicholson, A. E. (2010). *Bayesian Artificial Intelligence*. Boca Raton, FL: CRC Press.

Kulkarni, S., Bhat, S., and Moritz, C. S. (2017b). "Structure Discovery for Gene Expression Networks with Emerging Stochastic Hardware," in *Proceedings of the 2016 IEEE International Conference on Rebooting Computing (ICRC)*, Washington, DC, 147–155. doi: 10.1109/ICRC.2016.7738680

Kulkarni, S., Bhat, S., Khasanvis, S., and Moritz, C. A. (2017a). "Magneto-electric approximate computational circuits for Bayesian inference," in *Proceedings*

of the 2017 IEEE International Conference on Rebooting Computing (ICRC), Washington, DC. doi: 10.1109/ICRC.2017.8123678

Kungl, F., Schmitt, S., Klähn, J., Müller, P., Baumbach, A., Dold, D., et al. (2019). Accelerated physical emulation of Bayesian inference in spiking neural networks. *Front. Neurosci.* 13:1201. doi: 10.3389/fnins.2019.01201

Kwisthout, J., Bodlaender, H. L., and van der Gaag, L. C. (2010). "The necessity of bounded treewidth for efficient inference in Bayesian networks," in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, eds H. Coelho, R. Studer, and M. Wooldridge (Amsterdam: IOS Press), 237–242.

Lim, J., Lee, J., and Shin, C. (2021). Probabilistic artificial neural network for line-edge-roughness-induced random variation in FinFET. *IEEE Access* 9, 86581–86589. doi: 10.1109/ACCESS.2021.3088461

Malhotra, A., Lu, S., Yang, K., and Sengupta, A. (2020). Exploiting oxide based resistive RAM variability for bayesian neural network hardware design. *IEEE Trans. Nanotechnol.* 19, 328–331. doi: 10.1109/TNANO.2020.2982819

Mansinghka, V. K., Jonas, E. M., and Tenenbaum, J. B. (2008). *Stochastic Digital Circuits for Probabilistic Inference:* Technical Report MITCSAIL-TR 2069. Cambridge, MA: Massachusetts Institute of Technology.

Maron, M. E., and Kuhns, J. L. (1960). On relevance, probabilistic indexing, and information retrieval. *J. Assoc. Comput. Mach.* 7, 216–244.

Marsaglia, G. (2003). Xor shift RNGs. *J. Stat. Softw.* 8, 1–6.

Murphy, K., (1998). *A Brief Introduction to Graphical Models and Bayesian Networks*. Available online at: https://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html (accessed May 10, 2001).

Nallathambi, A., Sen, S., Raghunathan, A., and Chandrachoodan, N. (2021). Probabilistic spike propagation for efficient hardware implementation of spiking neural networks. *Front. Neurosci.* 15:694402. doi: 10.3389/fnins.2021.694402

Nasrin, S., Drobitch, J., Shukla, P., Tulabandhula, T., Bandyopadhyay, S., and Trivedi, A. R. (2020). Bayesian reasoning machine on a magneto-tunneling junction network. *Nanotechnology* 31:484001. doi: 10.1088/1361-6528/abae97

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Palo Alto, CA: Morgan Kaufmann. doi: 10.1016/C2009-0-27609-4

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Sebastian, A., Pannone, A., Radhakrishnan, S. S., and Das, S. (2019). Gaussian synapses for probabilistic neural networks. *Nat. Commun.* 10:4199. doi: 10.1038/s41467-019-12035-6

Seiler, C., Bchler, P., Nolte, L., Paulsen, R., and Reyes, M. (2009). "Hierarchical Markov random fields applied to model soft tissue deformations on graphics hardware," in *Recent Advances in the 3D Physiological Human*, eds J. J. Zhang, N. Magnenat-Thalmann, and D. D. Feng (London: Springer), 133–148. doi: 10.1007/978-1-84882-565-9_9

Sengupta, A., Parsa, M., Han, B., and Kaushik Roy, K. (2016). Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. *IEEE Trans. Electron Devices* 63, 2963–2970. doi: 10.1109/TED.2016.2568762

Serb, A., Manino, E., Messaris, I., Thanh, L. T., and Prodromakis, T. (2017). "Hardware-Level Bayesian Inference," in *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA.

Sesen, M. B., Nicholson, A. E., Banares-Alcantara, R., Kadir, T., and Brady, M. (2013). Bayesian networks for clinical decision support in lung cancer care. *PLoS One* 8:e82349. doi: 10.1371/journal.pone.0082349

Shachter, R., and Peot, M. (1990). "Simulation approaches to general probabilistic inference on belief networks," in *Proceedings of the sixth conference on Uncertainty in Artificial Intelligence*, eds M. Henrion, R. D. Shachter, J. F. Lemmer, and L. N. Kanal (Amsterdam: North-Holland), Vol. 5, 221–231.

Shamsi, J., Mohammadi, K., and Shokouhi, S. B. (2018). A hardware architecture for columnar-organized memory based on CMOS neuron and memristor crossbar arrays. *IEEE Trans. Very Large Scale Integr. Syst.* 26, 2795–2805. doi: 10.1109/TVLSI.2018.2815025

Shim, Y., Chen, S., Sengupta, A., and Roy, K. (2017). Stochastic spin-orbit torque devices as elements for Bayesian inference. *Nature* 7:14101. doi: 10.1038/s41598-017-14240-z

Specht, D. F. (1990). Probabilistic neural networks. *Neural Netw.* 3, 109–118.

Synopsys (2012). *Digital Standard Cell Library:SAED_EDK90_CORE Databook*. Yerevan: Synopsys Armenia Educational Department

Tang, G., Shah, A., and Michmizos, K. P. (2019). "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional SLAM," in *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau. doi: 10.1109/IROS40897.2019.8967864

Thakur, C. S., Afshar, S., Wang, R. M., Hamilton, T. J., Tapson, J., and Schaik, A. V. (2016). Bayesian estimation and inference using stochastic electronics. *Front. Neurosci.* 10:104. doi: 10.3389/fnins.2016.00104

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018). Large-Scale neuromorphic spiking array processors: a quest to mimic the brain the brain. *Front. Neurosci.* 12:891. doi: 10.3389/fnins.2018.00891

Theodoridis, S. (2015). *Machine Learning: A Bayesian and Optimization Perspective*. Cambridge, MA: Academic press.

Tipping, M. E. (2003). "Bayesian inference: an introduction to principles and practice in machine learning," in *Summer School on Machine Learning*, eds O. Bousquet, U. von Luxburg, and G. Rätsch (Berlin: Springer), 41–62.

Tziantzioulis, G., Gok, A. M., Faisal, S. M., Hardavellas, N., Memik, S., and Parthasarathy, S. (2015). "b-HiVE: a bit-level history-based error model with value correlation for voltage- scaled integer and floating point units," in *Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA. doi: 10.1145/2744769.2744805

Wang, J. J., Yu, Q., Hu, S. G., Liu, Y., Guo, R., Chen, T. P., et al. (2019). Winner-takes-all mechanism realized by memristive neural network. *Appl. Phys. Lett.* 115:243701. doi: 10.1063/1.5120973

Wang, K., Hu, Q., Gao, B., Lin, Q., Zhuge, F.-W., Zhang, D.-Y., et al. (2021). Threshold switching memristor-based stochastic neurons for probabilistic computing. *Mater. Horizons* 8, 619–629. doi: 10.1039/d0mh01759k

Weijia, Z., Ling, G. W., and Seng, Y. K. (2007). "PCMOS-based Hardware Implementation of Bayesian Network," in *Proceedings of the 2007 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, Tainan, 337–340. doi: 10.1109/EDSSC.2007.4450131

Wu, B., Feng, D., Tong, W., Liu, J., Wang, C., Zhao, W., et al. (2019). "ReRAM crossbar-based analog computing architecture for naive bayesian engine," in *Proceedings of the 2019 IEEE 37th International Conference on Computer Design (ICCD)*, Abu Dhabi, 147–155. doi: 10.1109/ICCD46524.2019.00026

Yang, K., Malhotra, A., Lu, S., and Sengupta, A. (2020). All-Spin Bayesian neural networks. *IEEE Trans. Electron Devices* 67, 1340–1347. doi: 10.1109/TED.2020.2968223

Yu, Z., Chen, F., and Liu, J. K. (2020). Sampling-Tree model: efficient implementation of distributed bayesian inference in neural networks. *IEEE Trans. Cogn. Dev. Syst.* 12, 497–510. doi: 10.1109/TCDS.2019.2927808

Yu, Z., Guo, S., Deng, F., Yan, Q., Huang, K., Liu, J. K., et al. (2018). Emergent inference of hidden markov models in spiking neural networks through winner-take-all. *IEEE Trans. Cybern.* 50, 1347–1354. doi: 10.1109/TCYB.2018.2871144

Zweig, G., and Russell, S. (1998). "Speech recognition with dynamic Bayesian networks," in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence* (Menlo Park, CA: American Association for Artificial Intelligence), 173–180.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 09 December 2021
doi: 10.3389/fnins.2021.750458

Mapping the BCPNN Learning Rule to a Memristor Model

Deyu Wang[1†], Jiawei Xu[1†], Dimitrios Stathis[2], Lianhao Zhang[3], Feng Li[1], Anders Lansner[2,4], Ahmed Hemani[2*], Yu Yang[2], Pawel Herman[2] and Zhuo Zou[1*]*

[1] State Key Laboratory of ASIC and System, School of Information Science and Technology, Fudan University, Shanghai, China, [2] School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, [3] Department of Electrical Engineering, Technical University of Denmark, Kongens Lyngby, Denmark, [4] Department of Mathematics, Stockholm University, Stockholm, Sweden

The Bayesian Confidence Propagation Neural Network (BCPNN) has been implemented in a way that allows mapping to neural and synaptic processes in the human cortex0and0has been used extensively in detailed spiking models of cortical associative memory function and recently also for machine learning applications. In conventional digital implementations of BCPNN, the von Neumann bottleneck is a major challenge with synaptic storage and access to it as the dominant cost. The memristor is a non-volatile device ideal for artificial synapses that fuses computation and storage and thus fundamentally overcomes the von Neumann bottleneck.0While the implementation of other neural networks like Spiking Neural Network (SNN) and even Convolutional Neural Network (CNN) on memristor has been studied, the implementation of BCPNN has not. In this paper, the BCPNN learning rule is mapped to a memristor model and implemented with a memristor-based architecture. The implementation of the BCPNN learning rule is a mixed-signal design with the main computation and storage happening in the analog domain.0In particular, the nonlinear dopant drift phenomenon of the memristor is exploited to simulate the exponential decay of the synaptic state variables in the BCPNN learning rule.0The consistency between the memristor-based solution and the BCPNN learning rule is simulated and verified in Matlab, with a correlation coefficient as high as 0.99. The analog circuit is designed and implemented in the SPICE simulation environment, demonstrating a good emulation effect for the BCPNN learning rule with a correlation coefficient as high as 0.98. This work focuses on demonstrating the feasibility of mapping the BCPNN learning rule to in-circuit computation in memristor. The feasibility of the memristor-based implementation is evaluated and validated in the paper, to pave the way for a more efficient BCPNN implementation, toward a real-time brain emulation engine.

Keywords: Bayesian Confidence Propagation Neural Network (BCPNN), learning rule, memristor, nonlinear dopant drift phenomenon, synaptic state update, spiking neural networks, analog neuromorphic hardware

OPEN ACCESS

Edited by:
Irem Boybat,
IBM Research-Zurich, Switzerland

Reviewed by:
Christopher Bennett,
Sandia National Laboratories, United States
Wei Wang,
Technion Israel Institute of Technology, Israel

***Correspondence:**
Anders Lansner
ala@kth.se
Ahmed Hemani
hemani@kth.se
Zhuo Zou
zhuo@fudan.edu.cn

[†] These authors have contributed equally to this work and share first authorship

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *30 July 2021*
Accepted: *11 November 2021*
Published: *09 December 2021*

Citation:
Wang D, Xu J, Stathis D, Zhang L, Li F, Lansner A, Hemani A, Yang Y, Herman P and Zou Z (2021) Mapping the BCPNN Learning Rule to a Memristor Model. Front. Neurosci. 15:750458. doi: 10.3389/fnins.2021.750458

1. INTRODUCTION

In the last decade, Artificial Neural Networks (ANNs) have made rapid and significant progress in real-world applications, demonstrating outstanding performance in a wide range of pattern recognition problems such as speech recognition (Hinton et al., 2012), image classification (Ciregan et al., 2012), and natural language processing (Yin et al., 2017). Despite the great success

and popularity of ANNs in data-driven computational paradigms, they have some limitations. First of all, most of the existing ANNs adopt supervised learning, requiring a large amount of labeled training data, which is different from the unsupervised and reward modulated learning mechanisms attributed to biological brains. Secondly, the most prominent learning algorithm used by ANNs is error back-propagation, which requires a high level of accuracy and is neither robust nor biologically plausible. Thirdly, the current mainstream ANN models do not account for the functionality underlying human cognition and inspiring artificial intelligence, like e.g., associative memory, temporal association, and reward-based trial-and-error learning. Unlike classical ANNs with non-spiking units, Spiking Neural Networks (SNNs) use the same event-based communication mechanism as the human brain where neurons communicate with spikes.

The Bayesian Confidence Propagation Neural Network (BCPNN) was originally derived from principles of Bayesian inference (Lansner and Ekeberg, 1989; Lansner and Holst, 1996) and was further developed into an architecture inspired by the modularity of the mammalian cortex with hypercolumn units (HCUs) and minicolumn units (MCUs). Later implementation within the framework of SNNs allowed mapping to neural and synaptic processes in the human cortex (Tully et al., 2014). Compared with other SNN models, BCPNN provides a compact and practical solution for the implementation of large-scale neural networks due to its modular, coarse-grained, and hierarchical architecture. Importantly, both reduced non-spiking and biologically detailed spiking realizations of BCPNN perform similar functions. They have been extensively used to model brain-like cognitive capabilities such as associative memory (Johansson and Lansner, 2007; Lundqvist et al., 2011), episodic memory (Chrysanthidis et al., 2021), and working memory (Fiebig and Lansner, 2017; Fiebig et al., 2020), which play a key role in human intelligence. In a broader perspective, we suggest that these advancements in simulating different aspects of human cognitive function within a system framework of brain-like BCPNN constitute a promising direction in the development of artificial general intelligence (AGI).

Furthermore, the local associative nature of the Bayesian-Hebbian BCPNN learning rule has also been leveraged in cortex-inspired neural networks built for pattern recognition problems in the machine learning domain. In particular, these recent developments were facilitated by the addition of a novel brain-like structural plasticity algorithm to build a hidden layer using the original synaptic trace variables of BCPNN in an unsupervised manner (Ravichandran et al., 2020, 2021a). Classification performance on the MNIST and Fashion-MNIST benchmark problems—98.6% and 88.9% on test sets, respectively (Ravichandran et al., 2021a)—is competitive with e.g., single-layer multi-layer perceptron (MLP) with backprop, restricted Boltzmann machine (RBM), and overcomplete autoencoder. The aforementioned unsupervised nature of the structural plasticity lends itself to the efficient use of unlabeled training examples, which has been exploited to perform semi-supervised learning with competitive results on MNIST for only 10–1,000 labeled training samples (Ravichandran et al., 2021b).

At present, BCPNN is usually implemented in high-performance computers, such as clusters (Johansson and Lansner, 2007), GPUs (Yang et al., 2020; Podobas et al., 2021), and ASICs (Stathis et al., 2020). However, these systems do not fully leverage the scalability of the modular BCPNN with its local learning since they are all based on the von Neumann architecture that separates computation and storage, which puts a high demand on computing and memory access. We observe that the ASIC implementation with its full customized architecture with the 3D-RAM, achieved three orders better efficiency compared to GPUs, but it is still many orders less efficient compared to a biological brain.

Besides overcoming the von Neumann bottleneck, the non-linearity of the memristor naturally mimics the behavior of synapses. This paper shows how these properties of memristors can be leveraged to implement the BCPNN learning rule. The long-term goal of this research is to realize a large-scale memristor-based BCPNN network that is 10–100x more efficient than ASICs. However, the research presented in this paper focuses on demonstrating the feasibility of mapping the BCPNN learning rule to an in-circuit memristor-based computation. Follow-up work to this paper will focus on addressing the non-idealities of memristors and the energy efficiency analysis.

The contributions of this work are as follows:

- The non-linearity of the memristor is exploited to emulate the synaptic traces in the BCPNN learning rule. On this basis, a memristor-based architecture for the BCPNN learning rule is proposed.
- The memristor-based design for the BCPNN learning rule is simulated and verified in Matlab. The consistency between the memristor-based solution and the reference model is validated, and the correlation coefficient is as high as 0.99.
- The analog circuit for the BCPNN learning rule is designed and implemented in the SPICE simulation environment. The SPICE simulation results demonstrate a good emulation effect for the BCPNN learning rule, and the correlation coefficient is as high as 0.98.

The rest of this paper is organized as follows: Section 2 introduces the background knowledge and details about BCPNN and the memristor. Section 3 shows the similarity between the BCPNN traces and the memristor non-linearity and demonstrates how to map the BCPNN learning rule to in-circuit memristor-based computation. Section 4 proposes the memristor-based architecture for the BCPNN learning rule and explains the corresponding analog circuit design. Section 5 presents the results of Matlab and SPICE-level simulations. Section 6 summarizes the paper and further discusses several aspects of this work. Finally, section 7 presents the prospects for future work.

2. PRELIMINARIES

2.1. BCPNN

2.1.1. BCPNN Overview

The BCPNN features a modular structure in terms of HCUs and MCUs, based on a generalization of the structure of the mammalian cortex, first described by Hubel and Wiesel

(Hubel and Wiesel, 1977). In models of the mammalian cortex, an HCU module has a diameter on the order of 500 μm and comprises about 100 MCUs with 50 μm diameter. Each MCU is composed of about 100 neurons, mainly excitatory pyramidal cells and one or two local inhibitory double bouquet cells (DeFelipe et al., 2006). Activity within an HCU is regulated by lateral inhibition mediated via inhibitory basket cells. In the abstract models, it takes the form of softmax that normalizes the total HCU activity (sum of the corresponding MCU activities) to 1. The number of HCUs in the human cortex has been estimated at around two million.

The BCPNN network can be represented with a $H \times M$ configuration, which means it is composed of H HCUs, and each HCU contains M MCUs. Generally, H is much larger than M. The number of HCUs H can be increased without an upper limit, while the number of MCUs M has an upper limit of about 100 based on biological data. Therefore, when it comes to the configuration of large networks, the number of H tends to be quite high. In a small network, each MCU can be fully connected to its local HCU and other HCUs, as shown in **Figure 1A**. Such full connectivity can not be employed in large networks due to the extreme cost of computation and storage. Instead, a cortex-inspired sparse patchy connection is adopted (Meli and Lansner, 2013), which greatly reduces the number of connections and yet maintains proper function.

Figure 1B presents the structure of the HCU, which is composed of 4 parts: 1) the presynaptic vector, used to store presynaptic traces Z_i, E_i and P_i; 2) the postsynaptic vector, used to store postsynaptic traces Z_j, E_j, P_j and the bias β_j; 3) the synaptic matrix, used to store synaptic traces E_{ij}, P_{ij} and the weight w_{ij}. 4) a certain number of MCUs, which integrate the incoming spiking activities and fire in a soft winner-take-all manner.

At a higher level, HCUs function like independent network modules between which spikes are transmitted. The HCU size depends on the number of incoming connections and MCUs. The biologically constrained maximum number of incoming connections and MCUs is 10,000 and 100, respectively. Consequently, in a max-size HCU, a synaptic matrix with a size of $10,000 \times 100$ is used, thus representing a million plastic synapses.

2.1.2. BCPNN Learning Rule

The BCPNN learning rule was derived from Bayes' rule while making some independent assumptions between neural activities and by transformation to log-space to achieve a proper neural activation function (Lansner and Ekeberg, 1989; Lansner and Holst, 1996; Sandberg et al., 2002). Thus, rather than being purely phenomenological as the commonly used Spike Timing Dependent Plasticity (STDP) learning rule, it was derived from the probabilistic inference. The BCPNN learning rule is in essence another kind of Hebbian learning rule in which synaptic updates are driven by co-activation between the pre- and post-synaptic neural units. It generates positive weights if the activity between neurons is positively correlated, zero weights if they are uncorrelated, and negative weights if they are anti-correlated. Besides, it has an intrinsic bias for each neural unit which reflects the prior activation and also is observed experimentally (Tully et al., 2014). The BCPNN learning rule equations estimate the activation and co-activation of network units utilizing a cascade of three exponential running averages, as shown in **Figure 2A**.

First, the incoming spikes drive pre- and post-synaptic Z-traces:

$$\frac{dZ_i}{dt} = \frac{S_i - Z_i}{\tau_{zi}} \quad \frac{dZ_j}{dt} = \frac{S_j - Z_j}{\tau_{zj}} \tag{1}$$

Here, i denotes pre- and j denotes post-synaptic variables and S represents incoming and generated spiking activity. These Z-traces in turn drive the E-traces and P-traces following the same kind of dynamics with different time constants:

$$\frac{dE_i}{dt} = \frac{Z_i - E_i}{\tau_e} \quad \frac{dE_j}{dt} = \frac{Z_j - E_j}{\tau_e} \quad \frac{dE_{ij}}{dt} = \frac{Z_i Z_j - E_{ij}}{\tau_e} \tag{2}$$

$$\frac{dP_i}{dt} = \frac{(E_i - P_i)}{\tau_p}\kappa \quad \frac{dP_j}{dt} = \frac{(E_j - P_j)}{\tau_p}\kappa \quad \frac{dP_{ij}}{dt} = \frac{(E_{ij} - P_{ij})}{\tau_p}\kappa \tag{3}$$

The learning rate κ in the dynamics of P traces modulates the learning. The E-traces form a synaptic tag important for delayed reinforcement learning. In many cases, the E-traces can be omitted and the P-traces can be updated according to equation (4) with an added parameter κ. The simplified BCPNN learning rule without E trace is shown in **Figure 2B**.

$$\frac{dP_i}{dt} = \frac{(Z_i - P_i)}{\tau_p}\kappa \quad \frac{dP_j}{dt} = \frac{(Z_j - P_j)}{\tau_p}\kappa \quad \frac{dP_{ij}}{dt} = \frac{(Z_i Z_j - P_{ij})}{\tau_p}\kappa \tag{4}$$

Finally, as shown in equation (5), the P-traces are used to update network unit biases, and weights with an additional parameter ε, which originates from a minimum spiking activity assumed for the pre- and postsynaptic units:

$$\beta_j = \log(P_j + \varepsilon) \quad W_{ij} = \log\left(\frac{P_{ij} + \varepsilon^2}{(P_i + \varepsilon)\cdot(P_j + \varepsilon)}\right) \tag{5}$$

2.1.3. BCPNN Application and Implementation

The BCPNN model has been used for neural computation and machine learning applications as well as to model the synaptic plasticity like long-term potentiation (LTP) and long-term depression (LTD) in SNN models of cortical associative memory. In the case of neural computation, BCPNN has been used to model scalable self-organizing associative memory (Johansson and Lansner, 2007). As for the classification of the MNIST machine learning benchmarking, an accuracy of 98.6% can be achieved while maintaining a high neurobiological plausibility (Ravichandran et al., 2020, 2021a). In the latter case, the hidden layer had 200 HCUs, each with 100 MCUs. Recent cortical associative memory models have focused on synaptic working memory using BCPNN as a model for rapid cortical synaptic plasticity (Fiebig and Lansner, 2017; Fiebig et al., 2020). The positive BCPNN weights are used as excitatory connections between pyramidal cells, while the negative ones are disynaptically inhibiting pyramidal cells in distant HCUs via e.g., double bouquet cells. These SNN models are tiny compared to their biological counterparts, typically comprising up to a thousand MCUs partitioned into some 30 HCUs.

The BCPNN model has been implemented in software packages, GPU, and supercomputer clusters. It has also been

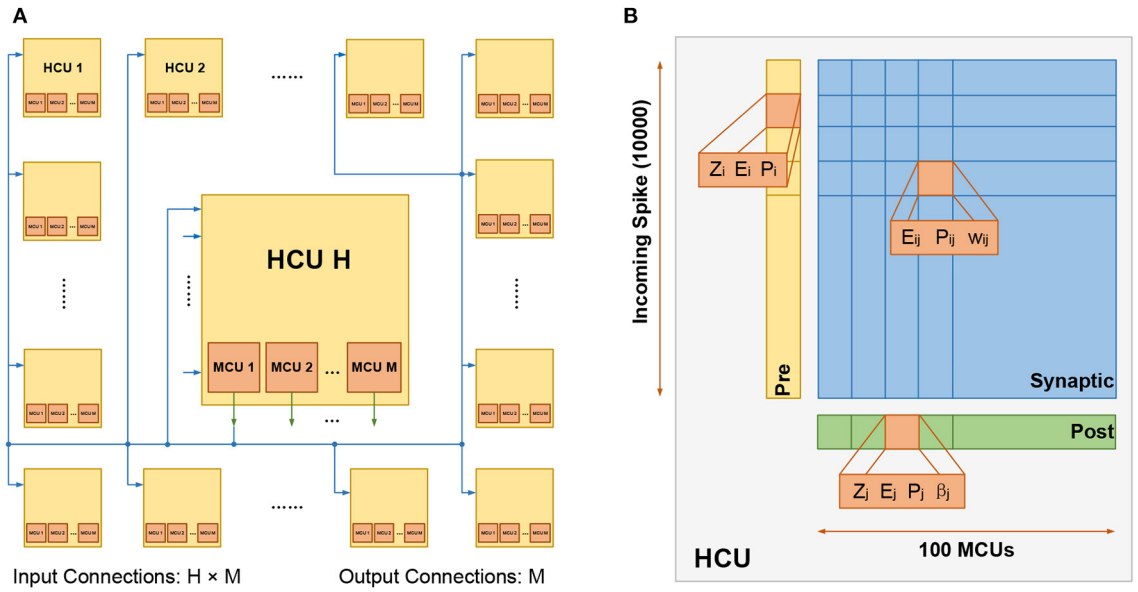


FIGURE 1 | (A) The fully connected HCUs. The number of HCUs is *H*, each HCU contains *M* MCUs. For each HCU, the number of input connections is $H \times M$, and the number of output connections is *M*. **(B)** The structure of HCU. The upper limit of input connections and MCUs is 10,000 and 100, respectively.

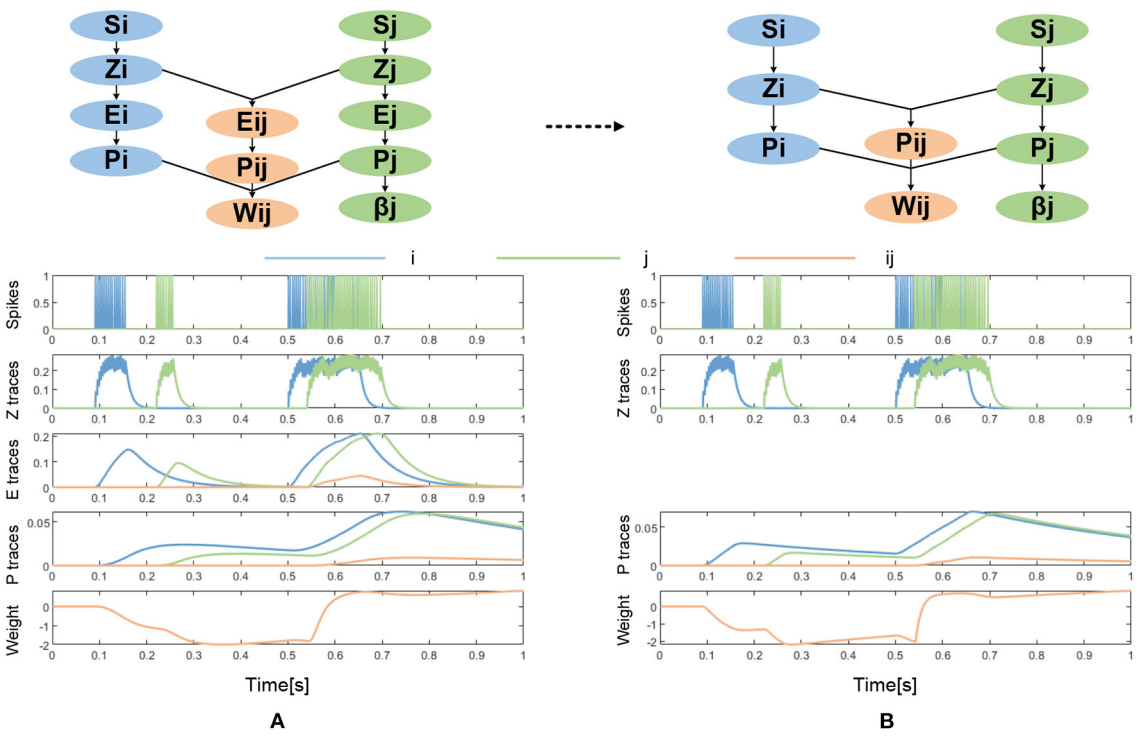


FIGURE 2 | (A) The original BCPNN learning rule (adapted after Tully et al., 2014). **(B)** The simplified BCPNN learning rule without E trace.

implemented as custom hardware with 3D integration of DRAM for the synaptic weights (Farahini et al., 2014; Lansner et al., 2014; Stathis et al., 2020; Yang et al., 2020). The BCPNN learning rule is amenable to low-precision implementation (Vogginger et al., 2015), and the cortical memory models have proven quite robust and tolerant to external as well as to intrinsic noise

and imprecision in weights and unit biases. Therefore, it is a highly scalable, modular, and hardware-friendly neuromorphic architecture with the potential for compact and low-power digital or mixed-signal design.

2.2. The Memristor

The memristor was predicted as a fourth fundamental circuit element following the resistor, capacitor, and inductor by Chua (1971). In 2008, HP Labs demonstrated and fabricated a memristor for the first time (Strukov et al., 2008). The HP Memristor was based on a nanoscale TiO_2 thin film, with a doped region and an undoped region, as shown in **Figure 3A**. The total resistance of the device is determined by the variable resistances of these two regions. When an external bias voltage is applied across the device, the charged dopants will drift, moving the boundary between the two regions. The HP memristor produces rich hysteretic current-voltage behavior, which can be observed in many nanoscale electronic devices. However, in nanoscale devices, a small voltage can yield enormous electric fields, secondarily producing significant non-linearities in ionic transport, which is called the non-linear dopant drift phenomenon. This phenomenon can be represented with a window function model, as shown in **Figure 3B**.

The memristor has many characteristics that can be utilized in a variety of applications. To begin with, as suggested by its name, a memristive device remembers the charge that passes through it rather than storing the charge so that the memristor is nonvolatile. What is more, the memristor device can store multi-valued rather than binary values. The ability to represent multi-bit values stems from the memristor's ability to have multiple intermediate points in its transfer curve. The transfer curve, with its hysteretic behavior and ability to represent multiple values, resembles biological synapses. This is the reason for memristors attracting attention as ideal building blocks for neuromorphic structures. The ability to remember multi-valued quantities in response to voltages applied to its terminals mimics analog computation. This *in-situ* computation has also been exploited to build general-purpose computers (Zidan et al., 2018), content addressable memory (Li et al., 2020a), and to implement neural networks, both spiking and non-spiking, as discussed next.

For both non-spiking artificial neural networks and spiking neural networks, the core operation is to reinforce or weaken the synaptic weights. The algorithms used for deciding the time, magnitude, and sign of reinforcement vary from one algorithm to another. The commonality is in applying appropriate voltages for an appropriate duration to the two terminals of the memristors.

In the ANN space, several memristor-based ANNs have been studied and implemented. A single-layer perceptron (Prezioso et al., 2015) was constructed based on transistor-free metal-oxide memristor crossbars, performing the perfect classification of images. The feasibility of a three-layer fully connected neural network on MNIST and a 13-layer Convolutional Neural Network (CNN) on CIFAR-10 using the flexible memristor are studied and evaluated (Xu et al., 2018). A five-layer memristor-based CNN (Yao et al., 2020) was demonstrated to perform image recognition on MNIST, achieving an accuracy of over 96%. It is worth noting that it is challenging to take *in-situ* training on memristor-based ANNs due to non-ideal device characteristics. Prezioso's work takes *in-situ* learning with a simple learning rule called the Manhattan update rule. In Yao's work, a hybrid-training method is taken to compensate for existing device imperfections.

In-situ computation in memristors has also been widely studied for spiking neural networks. A supervised learning model (Nishitani et al., 2015) that enables error backpropagation for spiking neural network hardware was proposed, and the memristor was employed as an electric synapse to store the analog synaptic weight in the circuit. An all-memristor stochastic SNN architecture (Wijesinghe et al., 2018) was proposed in which the inherent stochasticity of nanoscale devices is utilized to emulate the functionality of a spiking neuron. An area-efficient memristor SNN for hardware implementation (Zhou et al., 2019) was presented based on the modified SpikeProp-like supervised learning algorithm. An STDP-based SNN (Zhao et al., 2020) was proposed to achieve the mechanism of lateral inhibition and homeostasis by memristor-based inhibitory synapses. A novel memristive synapse model based on the HP memristor was proposed, and a spiking neural network hardware fragment was constructed (Huang et al., 2021).

However, the majority of the state-of-the-art memristor-based SNNs are limited in scale and employ simple learning rules such as STDP. Compared with small-scale SNNs using STDP, the BCPNN learning rule is more complex, and its computational structure is modular and cascaded. This paper exploits the non-linearity of the memristor and elaborates how *in-situ* analog computation in the memristor has been utilized to implement the BCPNN learning rule.

3. A MEMRISTOR-BASED BCPNN LEARNING RULE

3.1. BCPNN Model

The BCPNN learning rule has been depicted with ordinary differential equations, representing the update process of Z, E, P traces. To facilitate the hardware implementation, the ordinary differential equations (1,2,3) are further transformed to equations (6,7,8), respectively, with Euler's method, as shown below:

$$\begin{aligned} Z_i(t) &= Z_i(t-1) \times (1 - kz_i) + S_i(t-1) \times kz_i \\ Z_j(t) &= Z_j(t-1) \times (1 - kz_j) + S_j(t-1) \times kz_j \end{aligned} \tag{6}$$

$$\begin{aligned} E_i(t) &= E_i(t-1) \times (1 - ke) + Z_i(t-1) \times ke \\ E_j(t) &= E_j(t-1) \times (1 - ke) + Z_j(t-1) \times ke \\ E_{ij}(t) &= E_{ij}(t-1) \times (1 - ke) + Z_i(t-1) \times Z_j(t-1) \times ke \end{aligned} \tag{7}$$

$$\begin{aligned} P_i(t) &= P_i(t-1) \times (1 - kp) + E_i(t-1) \times kp \\ P_j(t) &= P_j(t-1) \times (1 - kp) + E_j(t-1) \times kp \\ P_{ij}(t) &= P_{ij}(t-1) \times (1 - kp) + E_{ij}(t-1) \times kp \end{aligned} \tag{8}$$

where,

$$kz_i = \frac{dt}{\tau_{z_i}} \; kz_j = \frac{dt}{\tau_{z_j}} \; ke = \frac{dt}{\tau_e} \; kp = \frac{dt}{\tau_p} \cdot \kappa \tag{9}$$

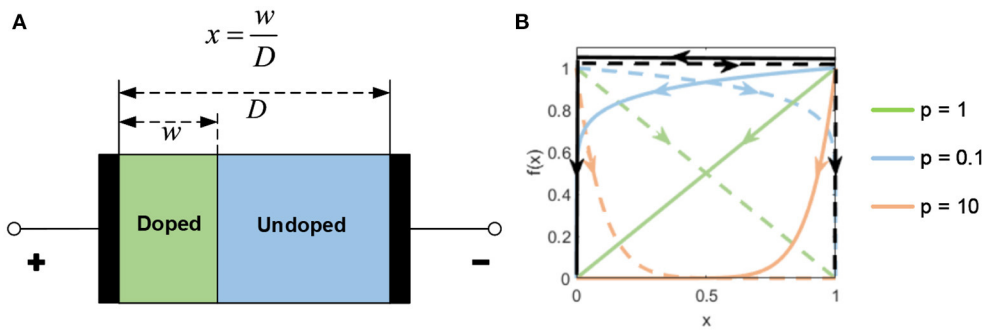


FIGURE 3 | (A) The HP model. **(B)** The window function.

The current values of Z, E, and P traces are all calculated from their previous values. The kz_i, kz_j, ke and kp are all constants. The simplified equation (4) can be transformed to equation (10) in the same manner, as follows:

$$\begin{aligned} P_i(t) &= P_i(t-1) \times (1-kp) + Z_i(t-1) \times kp \\ P_j(t) &= P_j(t-1) \times (1-kp) + Z_j(t-1) \times kp \\ P_{ij}(t) &= P_{ij}(t-1) \times (1-kp) + Z_i(t-1) \times Z_j(t-1) \times kp \end{aligned} \tag{10}$$

It should be noted that we do not consider the E trace in this work to facilitate hardware implementation. Therefore, we adopt simplified equation (10), whose update process and curve of traces can be seen in **Figure 2B**.

3.2. The Memristor Model

In 2008, HP Labs proposed the linear model of a memristor (Strukov et al., 2008). Following the HP model, a variety of memristor models have been devised, such as the non-linear ion drift model (Yang et al., 2008), Simmons Tunnel Barrier Model (Pickett et al., 2009), the TEAM model (Kvatinsky et al., 2013) and the VTEAM model (Kvatinsky et al., 2015). To emulate the non-linear dopant drift phenomenon, the window function is introduced as an essential part of a memristor model, and dozens of window functions have been proposed so far. However, most window functions are facing one or more of the following problems: the boundary effect, the boundary lock, and inflexibility (Xu et al., 2021). Joglekar's window function (Joglekar and Wolf, 2009) considers the boundary effect but suffers from the boundary lock problem. Biolek's window function (Biolek et al., 2009) takes the current direction into account to solve the boundary lock issue, but its parameter setting is not flexible enough. Recently, Li's window function (Li et al., 2020b) is proposed to consider all three issues. However, Li's window function is complex, and its expression is associated with six controlling parameters, which may increase the effort required for simulation. The window function that we proposed in Xu et al. (2021) is introduced to address this problem, which is both flexible and concise.

The VTEAM model is adopted for this work because of the following advantages: 1) the VTEAM model has a good fitting effect for the nonlinear bipolar physical memristor devices that we are concerned with (Johnson et al., 2010; Chanthbouala et al., 2012; Li et al., 2018); 2) this memristor model is voltage-controlled, and the threshold voltage phenomenon has been observed in many physical devices; 3) the VTEAM model is compatible with many window functions, which demonstrates great flexibility to simulate the non-linear dopant drift phenomenon. Besides, the window function that we proposed in Xu et al. (2021) is used in this work due to its flexibility and simplicity.

The VTEAM model is shown as follows:

$$\frac{dw(t)}{dt} = \begin{cases} k_{\text{off}} \cdot \left(\frac{v(t)}{v_{\text{off}}} - 1\right)^{\alpha_{\text{off}}} \cdot f(x(t)), & 0 < v_{\text{off}} < v \\ 0, & v_{\text{on}} < v < v_{\text{off}} \\ k_{\text{on}} \cdot \left(\frac{v(t)}{v_{\text{on}}} - 1\right)^{\alpha_{\text{on}}} \cdot f(x(t)), & v < v_{\text{on}} < 0 \end{cases} \tag{11}$$

$$x(t) = \frac{w(t)}{W} \tag{12}$$

$$R(t) = R_{\text{on}} + (R_{\text{off}} - R_{\text{on}}) \cdot x(t) \tag{13}$$

$$v(t) = R(t) \cdot i(t) \tag{14}$$

where $w(t)$ is an internal state variable in $[0, W]$, W is the maximum value of $w(t)$, $x(t)$ is an internal state variable in $[0, 1]$, $f(x)$ is the proposed window function, $v(t)$ is the voltage across the memristor, $i(t)$ is the current passing through the memristor, $R(t)$ is the resistance of the memristor, and t is the time. The parameters v_{on} and v_{off} are threshold voltages, R_{on} and R_{off} are the corresponding resistances of the memristor when $w(t)$ is 0 and W, respectively. The parameters k_{on}, k_{off}, α_{on} and α_{off} are constants.

The proposed window function is provided as below:

$$\begin{aligned} f(x) &= j[\text{sgn}(-i) \cdot (x-1) + \text{stp}(-i)]^p \\ \text{sgn}(i) &= \begin{cases} 1, & i \geq 0 \\ -1, & i < 0 \end{cases} \quad \text{stp}(i) = \begin{cases} 1, & i \geq 0 \\ 0, & i < 0 \end{cases} \end{aligned} \tag{15}$$

where i is the memristor current, and j and p are two tuning parameters. The memristor current i is positive when the internal

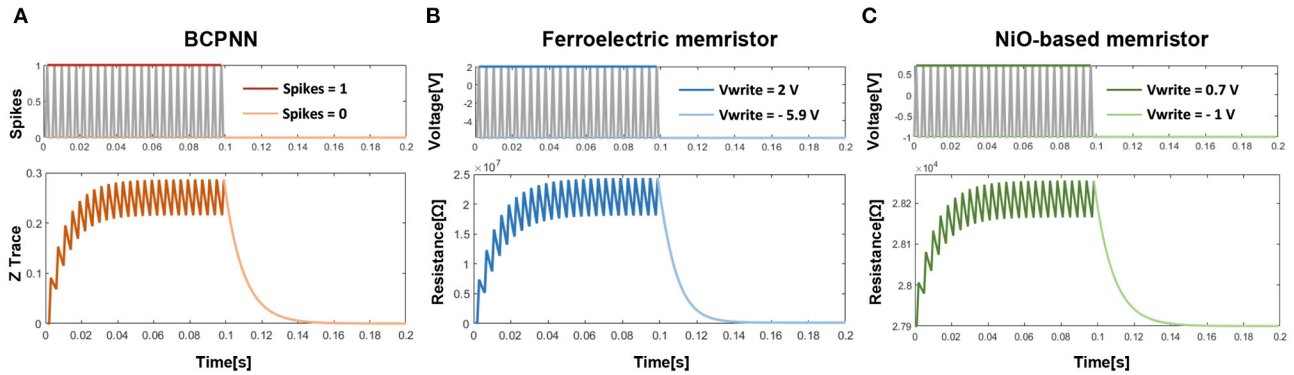


FIGURE 4 | Similarity between the BCPNN trace and the memristor nonlinearity: **(A)** The curve of Z trace. **(B)** The curve of resistance of the Ferroelectrc memristor. **(C)** The curve of resistance of the NiO-based memristor.

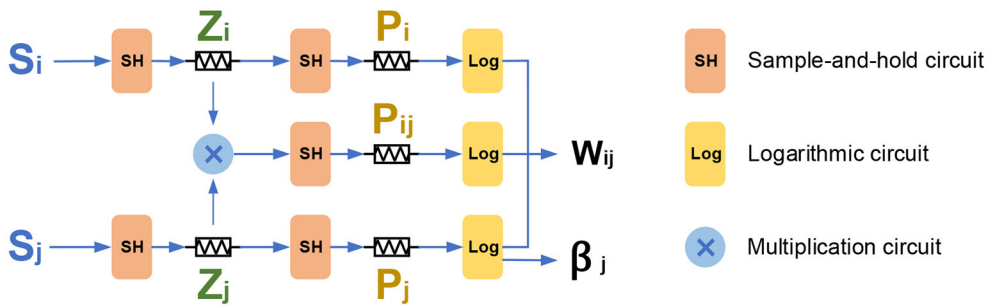


FIGURE 5 | The basic memristor-based architecture for the BCPNN learning rule.

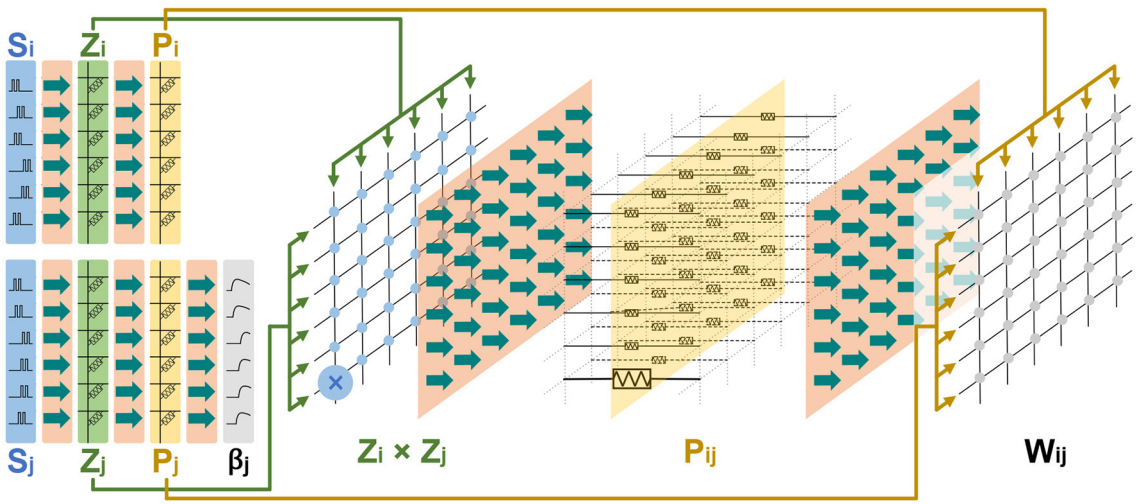


FIGURE 6 | The memristor-based architecture for an HCU with a configuration of 6 × 6.

state x is moving toward 1. The parameter j determines the magnitude, and the parameter p controls the decrease rate of the window function when approaching the boundaries. When p approaches 0, the non-linearity is weakened.

3.3. Similarity Between BCPNN Synaptic Traces and the Memristor Non-linearity

To explore the similarity of the BCPNN traces and the memristor non-linearity, the curve of the BCPNN trace (take

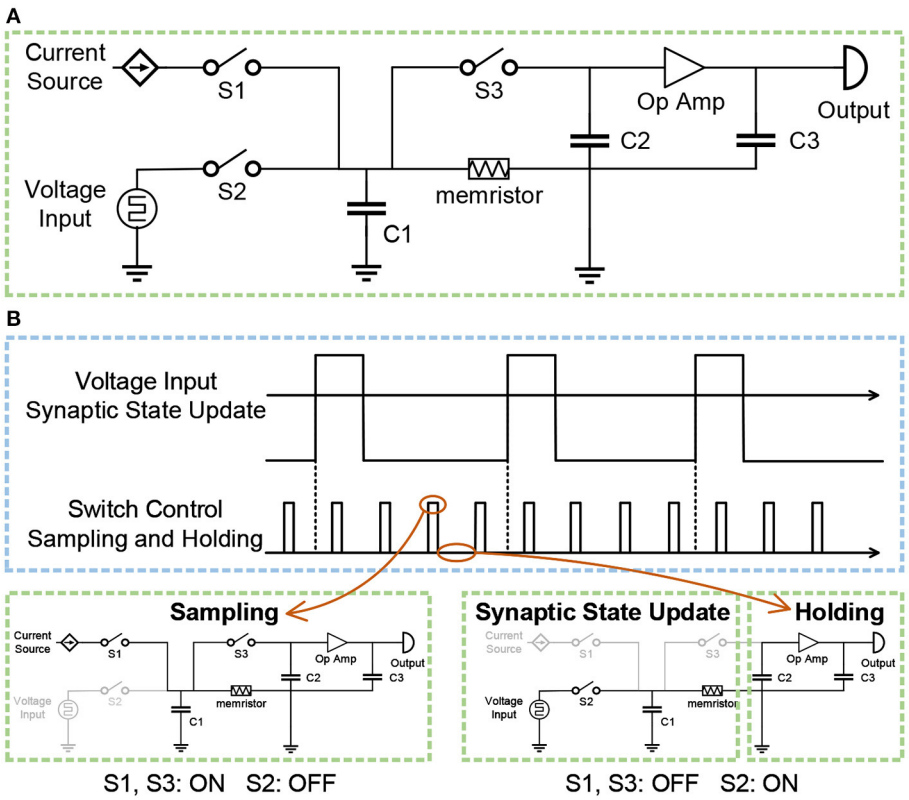


FIGURE 7 | (A) The diagram of the sample-and-hold circuit. **(B)** The switch between the sampling state and the holding state, controlled by switches S1, S2, and S3.

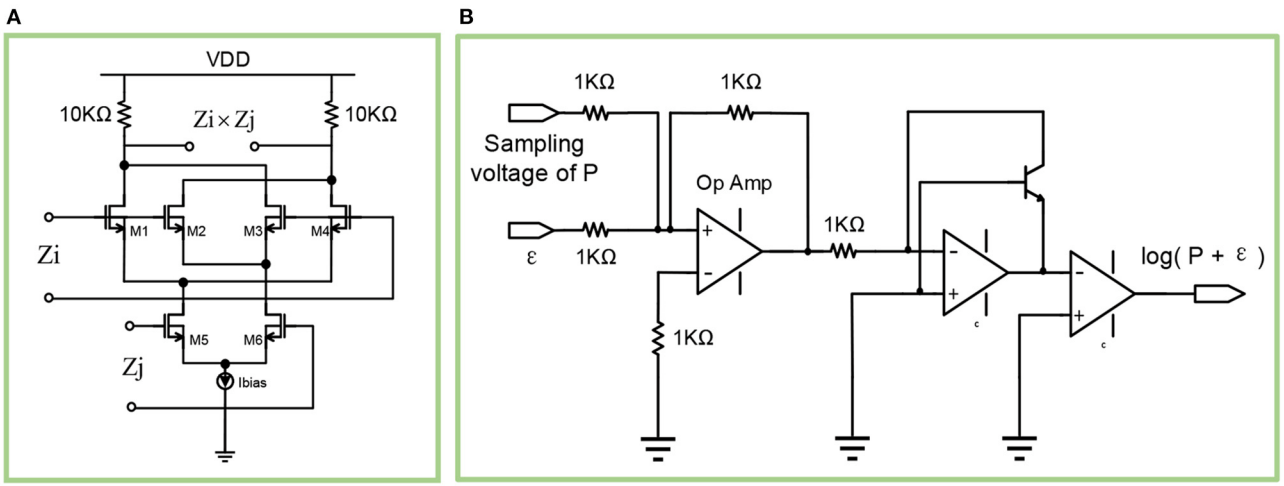


FIGURE 8 | (A) The diagram of the multiplication circuit. **(B)** The diagram of the logarithmic circuit.

Z trace as an example) and the curves of the resistances of two physical memristor devices are depicted in **Figure 4**. As shown in **Figure 4A**, the Z trace of BCPNN increases when there is a spike and decreases when there is no spike. While **Figures 4B,C** show that the resistances of the ferroelectric memristor (Chanthbouala et al., 2012) and the NiO-based memristor (Li et al., 2018) both increase when a positive voltage is applied and decrease when a negative voltage is applied. Therefore, a similarity can be found from **Figure 4** that both the BCPNN trace and the resistance of memristor change in a similar non-linear manner.

To further analyze the similarity between the BCPNN traces and the memristor non-linearity, their respective formulas are listed and compared. Take the Z trace of BCPNN as an example, when there is a spike or not, the formula of the Z trace is as follows:

$$\begin{aligned} S = 1 &: \; Z(t) = A \cdot Z(t-1) + B \\ S = 0 &: \; Z(t) = A \cdot Z(t-1) \end{aligned} \tag{16}$$

Correspondingly, when the voltage is positive or negative, the formula for the internal state variable of the memristor is as follows:

$$\begin{aligned} \nu\text{positive} &: \; w(t) = C \cdot w(t-1) + D \\ \nu\text{negative} &: \; w(t) = E \cdot w(t-1) \end{aligned} \tag{17}$$

where A, B, C, D and E are all constants expressed as:

$$\begin{cases} A = 1 - kz \\ B = kz \end{cases} \quad \begin{cases} C = 1 - dt \cdot \frac{k_{\text{off}}}{W} \cdot \left(\frac{v(t)}{v_{\text{off}}} - 1\right)^{\alpha_{\text{off}}} \\ \text{D} = dt \cdot k_{\text{off}} \cdot \left(\frac{v(t)}{v_{\text{off}}} - 1\right)^{\alpha_{\text{off}}} \\ E = 1 + dt \cdot \frac{k_{\text{on}}}{W} \cdot \left(\frac{v(t)}{v_{\text{on}}} - 1\right)^{\alpha_{\text{on}}} \end{cases} \tag{18}$$

Comparing formulas (16) and (17), a significant similarity can be observed, which also demonstrates the similarity between BCPNN traces and memristor non-linearity. Consequently, it is inspired that the non-linearity dopant drift phenomenon found in the memristor can be utilized to simulate the traces in the BCPNN learning rule.

4. MEMRISTOR-BASED ARCHITECTURE AND IMPLEMENTATION

4.1. Memristor-Based Architecture

The BCPNN learning rule involves the update of synaptic traces, the bias, and the weight. **Figure 5** presents the basic memristor-based architecture for the BCPNN learning rule. In the basic structure, five memristors are used to mimic the traces Z_i, Z_j, P_i, P_j, and P_{ij} respectively, and a multiplication circuit is used to calculate the product of Z_i and Z_j. Besides, five sample-and-hold circuits are used to provide the converted voltage input for the memristors, and three logarithmic circuits are used to calculate the weight w_{ij} and the bias β_j. The circuit design of the sample-and-hold circuit, logarithmic circuit, and the multiplication circuit will be explained in section 4.2.

As illustrated in **Figure 5**, the incoming presynaptic spike S_i is filtered into the Z_i trace through a sample-and-hold circuit. Then the Z_i trace is further filtered into the P_i trace with the same sample-and-hold circuit. Similarly, the postsynaptic spike S_j is first filtered into the Z_j trace, and then the Z_j trace is filtered into the P_j trace, both via a sample-and-hold circuit. Besides, the Z_i, Z_j traces are multiplied with each other, and then the obtained $Z_i \times Z_j$ is filtered into the P_{ij} through a sample-and-hold circuit. Last but not least, the P_{ij}, together with the P_i and P_j is used to calculate the weight W_{ij} through a logarithmic circuit. The P_j trace is calculated through a logarithmic circuit to obtain the value of bias β_j. It should be noted that although the E trace is removed in this work, it could be added without any issue by adding another level in the cascade if the E trace is needed.

TABLE 1 | Parameters for the Simulations.

Parameters	Value	Parameters	Value
BCPNN Model			
kz_i	1/11	kz_j	1/11
kp	1/500	ε	0.01
Memristor Model			
p	1	j	1
α_{off}	1	α_{on}	1
v_{off}	0.02 V	v_{on}	–0.02 V
R_{off}	200 kΩ	R_{on}	2 kΩ
k_{off}	21 nm/s	k_{on}	–28 nm/s
W	1 nm	w_{init}	0 nm
dt	1 ms		

What's more, the basic memristor-based architecture described above can be reused and scaled to build a memristor-based HCU that includes more synaptic traces. As a typical case for demonstration, **Figure 6** presents the memristor-based architecture for an HCU with a 6 × 6 configuration. The HCU contains a presynaptic vector of length 6, a postsynaptic vector of length 6, and a synaptic matrix of size 6 × 6.

It should be noted that the intention of **Figure 6** is to illustrate the scalability of the basic architecture in **Figure 5**. In this work, we focus on simulating and implementing the basic memristor-based architecture for the BCPNN learning rule in **Figure 5**.

4.2. Analog Circuit Implementation

4.2.1. Pre- and Post-synaptic Trace

The spike-based BCPNN is implemented with local synaptic state variables Z_i, Z_j, P_i, P_j and P_{ij}, which keep track of presynaptic, postsynaptic and synaptic activities. The implementation of pre- and post-synaptic trace Z_i, Z_j, P_i, P_j can be divided into two cascaded processing stages. In the first stage, the incoming pre- and post-synaptic spike trains S_i, S_j are low pass filtered into the Z_i, Z_j traces, with time constants τ_{zi} and τ_{zj}. In the second stage, the Z_i, Z_j traces are low pass filtered into the P_i, P_j traces with time constant τ_p. To implement the above two cascaded processing stages, two sample-and-hold circuits are cascaded in the analog circuit implementation. **Figure 7A** presents the diagram of the sample-and-hold circuit. The voltage input is used to represent the incoming spike trains S_i, S_j. The input spike is either 0 or 1, while the voltage input is either excitatory 193.2 mV or inhibitory –149.9 mV. The input of the current source is constant, which is used to transform the resistance of the memristor into a voltage value. Switches S1, S2, S3 are used to control the switch between the sampling state and the holding state. Three capacitors C1, C2, C3, are used to store voltage. Besides, an operational amplifier is utilized to amplify the voltage value.

Figure 7B illustrates the switch between the sampling state and the holding state. When switches S1, S3 are on and switch S2 is off, the circuit is in the sampling state. The constant current

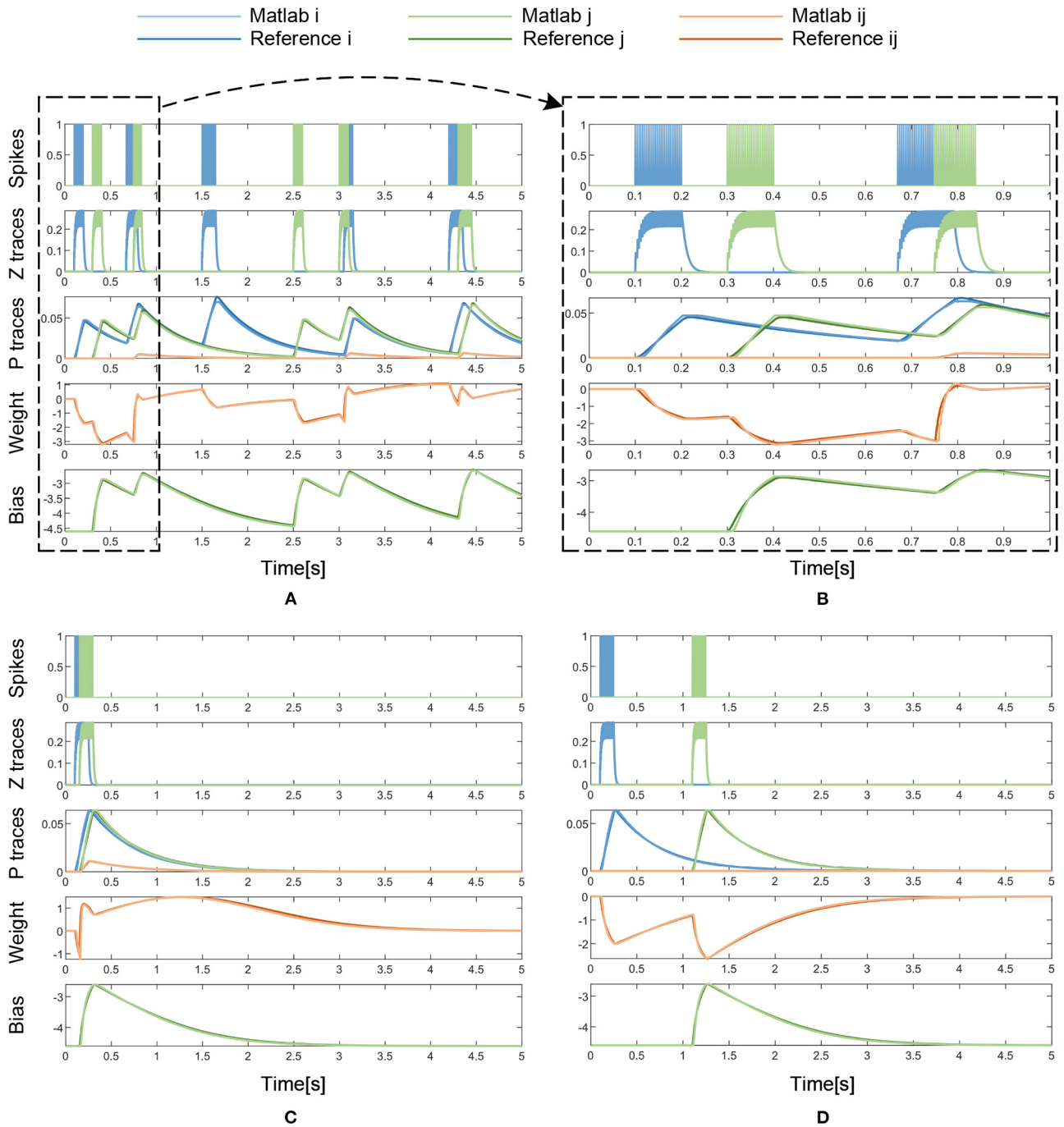


FIGURE 9 | Matlab simulation results with dense or sparse incoming spikes. **(A)** Dense Spikes: 5-s simulation. **(B)** Dense Spikes: 1-s simulation. **(C)** Sparse Spikes: presynaptic spike train S_i overlaps with postsynaptic spike train S_j. **(D)** Sparse Spikes: presynaptic spike train S_i is seperated with postsynaptic spike train S_j.

of the current source passes through the memristor to obtain the voltage, which is stored in capacitor C1. Since switch S3 is on, the voltage stored in capacitor C2 is equal to the voltage stored in capacitor C1. Therefore, the resistance of the memristor is converted into the corresponding voltage value, and the voltage value is stored in the capacitor C2, thus completing a sampling process. When switch S2 is on and switches S1, S3 are off, the left part of the circuit is responsible for the update of synaptic traces, and the right part of the circuit is in the holding state. The voltage source is the input excitation of the memristor, thereby changing the resistance of the memristor. At the same time, the sampled voltage stored in the capacitor C2 is amplified by the operational

amplifier, and the obtained voltage is stored in the capacitor C3 as the input voltage of the next-stage circuit. Similarly, the second stage adopts the same circuit, and the only difference is that the voltage input of the second circuit is the output of the first circuit rather than a voltage source.

4.2.2. Synaptic Trace Pij

The pre- and postsynaptic traces Z_i, Z_j, P_i, P_j can be obtained with the mentioned sample-and-hold circuit. However, to get the value of synaptic trace P_{ij}, an extra multiplier is required to calculate the product of Z_i and Z_j, as illustrated in formula (10).

As shown in **Figure 8A**, the multiplication circuit is based on the classic Gilbert cell. The resistance of the two resistors are both 10kΩ. The aspect ratios W/L for $M1$, $M2$, $M3$, $M4$ are 1μm/0.18μm, while the aspect ratios W/L for $M5$, $M6$ are 2μm/0.18μm. Besides, the bias voltage Vdc for Z_i and Z_j are 1.5 v and 1.3 v respectively.

4.2.3. Weight and Bias Computation

The three P traces P_i, P_j, P_{ij} represent the exponentially weighted moving averages of firing probability for presynaptic spikes, postsynaptic spikes, and spike co-activation respectively, which are used to compute the weight w_{ij} and the bias β_j. The calculation formula (5) for w_{ij} and β_j can be further rewritten as:

$$\begin{aligned} W_{ij} &= \log(P_{ij} + \varepsilon^2) - \log(P_i + \varepsilon) - \log(P_j + \varepsilon) \\ \beta_j &= \log(P_j + \varepsilon) \end{aligned} \tag{19}$$

The key to the calculation of w_{ij} and β_j lies in the logarithmic calculation of the sum of P trace and the constant ε, as shown in formula (19). Therefore, a logarithmic calculation circuit is required. As shown in **Figure 8B**, the sampling voltage of P trace (P_i, P_j, P_{ij}) is added with the constant parameter ε, then the sum is logarithmically calculated through a triode and an operational amplifier. Using such a circuit, the bias β_j can be obtained with an input pair of P_i and ε. Similarly, using three such circuits, whose input pairs are P_{ij} and ε^2, P_i and ε, P_j and ε respectively, the results of the three circuits can be used to get the value of weight w_{ij}.

5. EXPERIMENTAL RESULTS

In this section, we conduct simulations to verify the feasibility of the memristor-based implementation for the BCPNN learning rule at both the algorithm level and the circuit level. From the algorithmic perspective, we conducted simulations in Matlab. From the circuit-level perspective, we conducted SPICE-level simulations. The typical values of the parameters used in the simulations are shown in **Table 1**, including the parameters of the BCPNN model and the memristor model.

5.1. Matlab Simulation Results

To verify the effectiveness of the memristor-based solution for the BCPNN learning rule from an algorithmic perspective, a simulation of the Z traces, P traces, the weight w_{ij}, and the bias β_j is conducted using a model of the memristor device in Matlab. In the simulation, the results of the memristor-based solution are compared with those of the BCPNN reference model. The simulation lasts for 5 s with a simulation step of 1 ms, which is the simulation step in BCPNN.

TABLE 2 | Five-second simulation results with dense spikes in matlab.

Trace	Mean error	Max error	RMSE	Correlation coefficient
Z_i	0.0000	0.0000	0.0000	1.0000
Z_j	0.0000	0.0000	0.0000	1.0000
P_i	0.0015	0.0064	0.0019	0.9961
P_j	0.0013	0.0045	0.0015	0.9973
P_{ij}	0.0001	0.0008	0.0002	0.9984
w_{ij}	0.0418	1.4643	0.0862	0.9972
β_j	0.0408	0.2795	0.0489	0.9979

In **Figure 9** and **Table 2**, the simulation results are visualized and analyzed. **Figure 9A** presents the memristor-based 5-s Matlab simulation results with dense incoming spikes. To take a closer look at the difference between the memristor-based results and the reference model of BCPNN, the period from 0 to 1 s in **Figure 9A** has been enlarged, as shown in **Figure 9B**. With the same incoming pre- and post-synaptic spikes, the Z traces (Z_i, Z_j) of the memristor-based solution are the same as those of the BCPNN model. Therefore, in the Z traces part, the Z_i, Z_j curves of the two models completely coincide. As for the P traces (P_i, P_j, P_{ij}), the weight w_{ij} and the bias β_j, the curves of the two models are not the same but very close. In particular, simulations with sparse spikes are carried out to observe the change of the weight in the long-lasting silent state. When the presynaptic spike train S_i overlaps with the postsynaptic spike train S_j, the weight rises and finally decays to 0 in the long-lasting silent state, as shown in **Figure 9C**. Similarly, when the presynaptic spike train S_i is separated from the postsynaptic spike train S_j, the weight drops and gradually returns to 0 in a long-lasting silent state, as shown in **Figure 9D**. In the analysis of the simulation results, the average error, maximum error, Root Mean Square Error (RMSE), and correlation coefficient are used as the main evaluation metrics, as shown in **Table 2**. Due to the nonlinearity of the memristor, the memristor-based emulation of the BCPNN learning rule is accurate with a correlation coefficient of over 0.99.

5.2. SPICE Simulation Results

To further validate the feasibility of the memristor-based design from a circuit-level perspective, a SPICE-level simulation is conducted for the analog circuit implementation. In the SPICE simulation, the cascade circuit in **Figure 5** is implemented, where 5 sample-and-hold circuit modules, 3 logarithmic circuit modules, and 1 multiplication circuit module described in section 4.2 are used. The parameters for the BCPNN model and the memristor model used in the SPICE simulation are the same as those used in the Matlab simulation, as shown in **Table 1**. It is worth noting that the timestep in the Matlab simulation is 1 ms, while the timestep in the SPICE simulation is 100 ns because of the limitation of the timestep for transistors in the simulation environment.

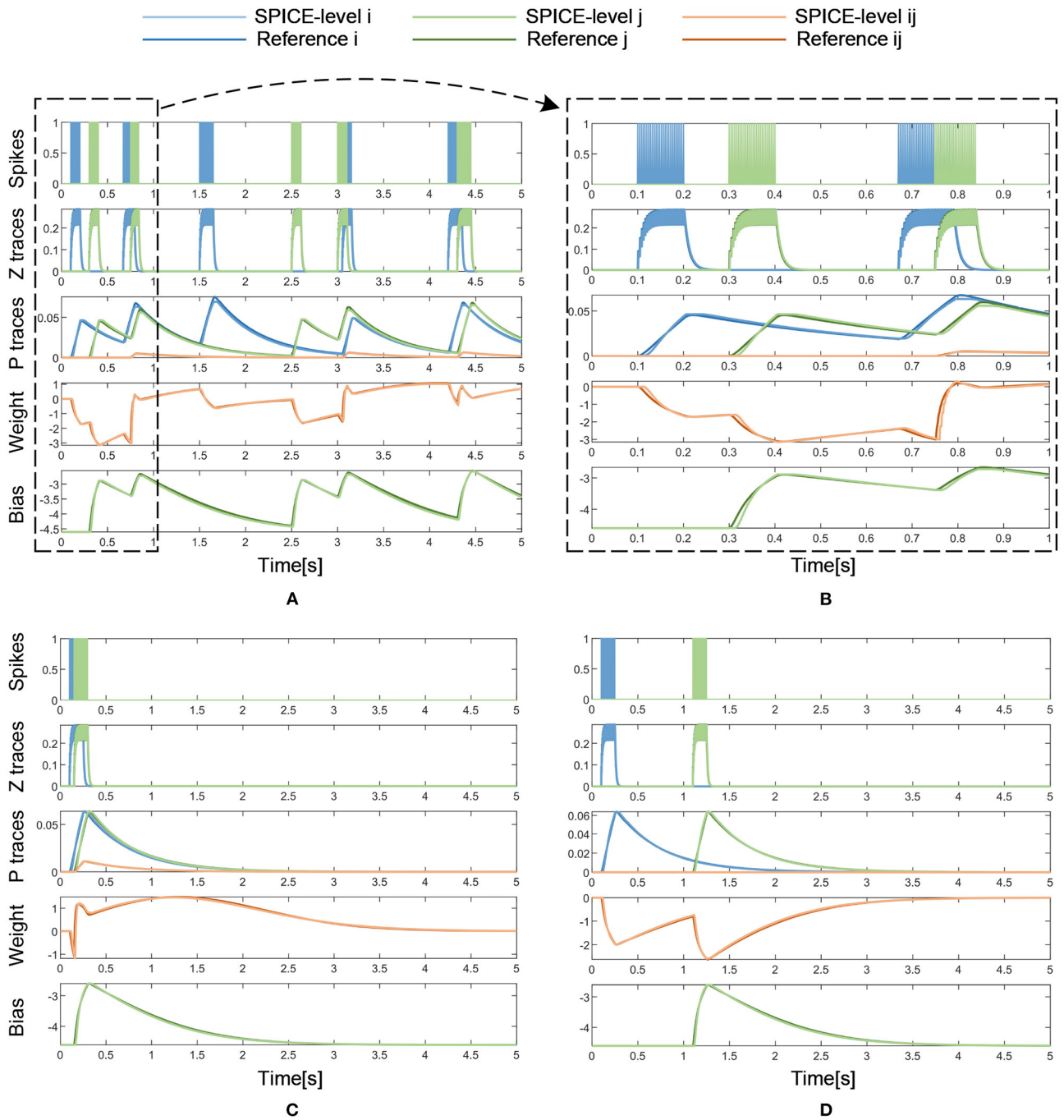


FIGURE 10 | Analog circuit implementation results with dense or sparse incoming spikes. **(A)** Dense Spikes: 5-s simulation. **(B)** Dense Spikes: 1-s simulation. **(C)** Sparse Spikes: presynaptic spike train S_i overlaps with postsynaptic spike train S_j. **(D)** Sparse Spikes: presynaptic spike train S_i is seperated with postsynaptic spike train S_j.

With the same input of pre- and post-synaptic spikes, the results of the SPICE-level simulation are compared with those of the reference model and the error is analyzed. **Figure 10A** presents the SPICE simulation results with the same dense incoming spikes as in the Matlab simulation. Similarly, the period from 0 to 1 s of the simulation results is magnified to show more details of the curves, as shown in **Figure 10B**. Besides, **Figures 10C,D** also demonstrates that the weight increases with a pair of correlated S_i and S_j and decreases with a pair of uncorrelated S_i and S_j. In a long-lasting silent state, the

weight returns to 0 eventually. As shown in **Table 3**, the SPICE simulation results of memristor-based solution demonstrate a fairly high degree of fit with the reference model of BCPNN, and a correlation coefficient of over 0.98 is achieved. All in all, it is validated that the memristor-based solution for BCPNN can achieve a high degree of fit with the reference BCPNN model in the analog circuit implementation.

TABLE 3 | Analog circuit implementation results of 5-s simulation with dense spikes.

Trace	Mean error	Max error	RMSE	Correlation coefficient
Z_i	0.0046	0.0909	0.0147	0.9830
Z_j	0.0041	0.0909	0.0138	0.9830
P_i	0.0014	0.0067	0.0018	0.9965
P_j	0.0012	0.0056	0.0015	0.9974
P_{ij}	0.0001	0.0011	0.0002	0.9981
w_{ij}	0.0432	1.5765	0.1003	0.9957
β_j	0.0483	0.3733	0.0631	0.9971

6. DISCUSSION

In this paper, the BCPNN learning rule is mapped to a memristor model and implemented with a memristor-based architecture. The similarity between the nonlinearity of the memristor and the trace update rule of BCPNN is explored and analyzed. The strong correlation between the simulated memristor-based BCPNN traces and the reference BCPNN traces has been validated in the Matlab simulation with a correlation coefficient over 0.99. Moreover, the analog circuit design of the memristor-based architecture is implemented, and the SPICE-level implementation for the BCPNN learning rule can achieve a decent emulation effect with a correlation coefficient of over 0.98.

6.1. Cumulative Error Analysis

The cumulative error of the memristor-based implementation can be analyzed from three aspects: the BCPNN algorithm, the memristor-based solution for BCPNN, and the analog circuit implementation. Firstly, as described before, BCPNN employs a correlation-based learning rule, which is robust and tolerant to the intrinsic noise and imprecision. BCPNN has proven to be able to function using lower precision (Vogginger et al., 2015). Secondly, as shown in **Table 2**, the memristor-based solution for the BCPNN learning rule presents a good simulation effect with the reference model. Moreover, it can be seen from **Figure 9** that the simulation effect does not deteriorate with the increasing simulation time, which means that there is no significant increase of cumulative error. Thirdly, the same is true for the analog circuit implementation, as can be seen in **Figure 10** and **Table 3**. Therefore, the cumulative error will not affect the stability of the memristor-based implementation for the BCPNN learning rule.

6.2. Setting of the Parameter ε

In the BCPNN model, the setting of the parameter ε has an impact on the performance of BCPNN-based tasks, as shown in **Figure 11**. With ε less than 0.001, good performance was achieved in an associative memory task and a standard machine learning classification benchmark (MNIST, LeCun et al., 1998). With ε equal to 0.01, the associative memory task still maintained good performance, but the performance in the MNIST task dropped a lot. For the experiments in section 5, the parameter ε was set to be 0.01, due to the limitation of the resolution of the logarithmic circuit. Later work will seek a higher-precision analog logarithmic circuit design or adopt digital methods to implement the logarithmic calculation of weight. In this way, the value of ε can be set to be less than 0.001, which can likely meet the requirement of most BCPNN-based tasks.

It should be noted that the intention of **Figure 11** is to analyze the impact of the value of ε (a parameter in the BCPNN learning rule) on the BCPNN performance from the perspective of software simulation. For the details about the working mechanism of the whole BCPNN network and how it implements associative memory tasks and practical recognition functions like MNIST classification, these works can be referred to (Johansson and Lansner, 2007; Meli and Lansner, 2013; Ravichandran et al., 2020, 2021a). The realization of the whole memristor-based BCPNN network and the algorithmic benchmarking is outside the scope of this paper and is what we plan to do in follow-up work.

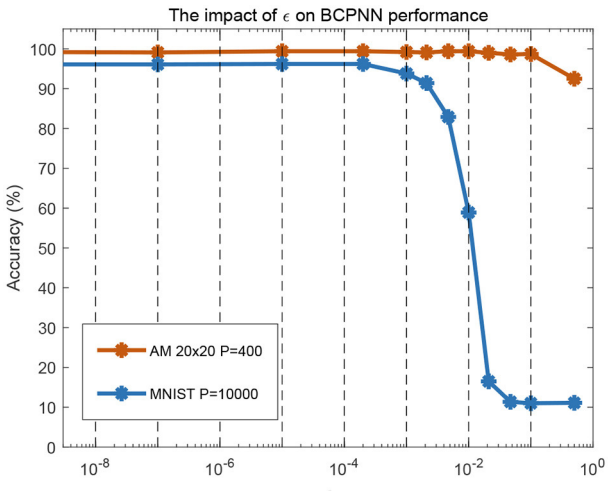


FIGURE 11 | Software simulation of the impact of ε on BCPNN performance in associative memory (AM) storage capacity and handwritten digit recognition (MNIST). For AM, a network configuration of $H \times M = 20 \times 20$ was tested for storage of 400 patterns. For MNIST, all 10,000 test patterns were used.

6.3. Consideration of Device Variation

In this paper, we focus on mapping the BCPNN learning rule to a memristor model and validating the feasibility of the memristor-based implementation at the algorithm and circuit level. However, in reality, memristor-based structures suffer from device variations due to process variation and age degradation. These two factors lead to two different types of variations in the memristors devices (Park et al., 2013; Le et al., 2018). The first is spatial variations, where different devices in the crossbar react differently to the applied voltage, i.e., identical voltage pulse

can drive different devices to different resistances. The second is temporal variations, where the behavior of the same device will change over time. Neural networks can adapt to such variations by taking them into account during the training of the network. This method has been used in deep neural networks (Long et al., 2019) and spiking neural networks (Querlioz et al., 2013). The authors in Querlioz et al. (2013) identify non-supervised learning as one of the fundamental benefits of the STDP learning rule that helps when dealing with device variations. Previous work indicates that the BCPNN learning rule is amenable to low-precision implementation (Vogginger et al., 2015), and the cortical memory models have proven quite robust and tolerant to external as well as to intrinsic noise and imprecision in weights and unit biases. In the follow-up work, we will focus on the non-idealities of memristors and study to what extent BCPNN's robustness can absorb the non-idealities and what other measures could be needed to cope with the non-idealities.

7. FUTURE WORK

As a follow-up to this paper, we plan to rigorously address the issue of nonidealities in memristors. Specifically, its variance in both space and time. We plan to quantify the extent to which BCPNN's robustness can cope with the variances and if that is not sufficient, we will study how the behavior diverges and use these experiments to devise techniques to counter the nonidealities.

Next to addressing nonidealities, the aspect on our priority list is to make the implementation more complete. This would involve implementing the control logic in CMOS, data converters, drive circuits, etc. It is also obvious, a large stack of memristor devices cannot be driven by single drivers. For this reason, we plan to experiment with and find out fragments of memristor fabrics that can be stacked with scalable drive circuits. Besides the above, we might also need to implement compensation logic to deal with nonidealities in the spirit of pre-distortion.

Having a good grip on nonidealities and more complete implementation, we will then be in a position to have a fair comparison of performance and energy efficiency between a memristor-based implementation of BCPNN and pure digital implementations that we have been experimenting with (Stathis et al., 2020; Yang et al., 2020). Besides providing realistic comparison, such an experiment will also provide us with inputs to create a more optimized implementation.

Designing memristor-based systems, at present, is a circuit-level effort. This is cumbersome and not accessible to everyone. We plan to develop, a Lego-inspired design flow called SiLago (Hemani et al., 2017), to enable automation of memristor-based designs from higher abstractions. Some work toward building such infrastructure has happened for CMOS-based conventional digital designs (Gonzalez et al., 2021; Hemani et al., 2021). We plan to enhance this for the memristors.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

The initial idea proposed in the manuscript came from DS, AL, and AH. DW performed experiments and was responsible for the manuscript writing. JX proposed the methodology and guided the overall experimental design. LZ and FL contributed in the SPICE and Matlab simulations. ZZ provided supervision on DW, JX, and FL's work. AL, AH, DS, YY, PH, and ZZ helped with the refinement of this work and the revision of the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported in part by the National Natural Science Foundation of China under Grant 61876039 and 62011530132 (NSFC-STINT project), and Shanghai Municipal Science and Technology Major Project No. 2021SHZDZX0103 and No. 2018SHZDZX01, and in part by the Shanghai Platform for Neuromorphic and AI Chip under Grant 17DZ2260900. In part, this work was financed by the mobility grant from STINT Sweden Dnr: CH2019-8357.

REFERENCES

Biolek, Z., Biolek, D., and Biolkova, V. (2009). SPICE model of memristor with nonlinear dopant drift. *Radioengineering* 18, 201–214. doi: 10.1049/el.2010.0358

Chanthbouala, A., Garcia, V., Cherifi, R. O., Bouzehouane, K., Fusil, S., Moya, X., et al. (2012). A ferroelectric memristor. *Nat. Mater.* 11, 860–864. doi: 10.1038/nmat3415

Chrysanthidis, N., Fiebig, F., Lansner, A., and Herman, P. (2021). Traces of semantization-from episodic to semantic memory in a spiking cortical network model. *bioRxiv*. doi: 10.1101/2021.07.18.452769

Chua, L. (1971). Memristor-The missing circuit element. *IEEE Trans. Circ. Theory* 18, 507–519. doi: 10.1109/TCT.1971.1083337

Ciregan, D., Meier, U., and Schmidhuber, J. (2012). "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition* (Providence, RI: IEEE), 3642–3649.

DeFelipe, J., Ballesteros-Yá nez, I., Inda, M. C., and Mu noz, A. (2006). Double-bouquet cells in the monkey and human cerebral cortex with special reference to areas 17 and 18. *Prog. Brain Res.* 154, 15–32. doi: 10.1016/S0079-6123(06)54002-6

Farahini, N., Hemani, A., Lansner, A., Clermidy, F., and Svensson, C. (2014). "A scalable custom simulation machine for the Bayesian confidence propagation neural network model of the brain," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Singapore: IEEE), 578–585.

Fiebig, F., Herman, P., and Lansner, A. (2020). An indexing theory for working memory based on fast hebbian plasticity. *eNeuro* 7, 1–22. doi: 10.1523/ENEURO.0374-19.2020

Fiebig, F., and Lansner, A. (2017). A spiking working memory model based on Hebbian short-term potentiation. *J. Neurosci.* 37, 83–96. doi: 10.1523/JNEUROSCI.1989-16.2016

Gonzalez, J. A., Hemani, A., and Stathis, D. (2021). "Synthesis of predictable global NoC by abutment in synchoros VLSI design," in *Proceedings 15th*

IEEE/ACM International Symposium on Networks-on-Chip – NOCS 2021 (Virtual Conference).

Hemani, A., Jafri, S. M. A. H., and Masoumian, S. (2017). "Synchoricity and NOCs could make billion gate custom hardware centric SOCs affordable," in *Proceedings2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)* (Seoul), 1–10.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-,r., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process Mag.* 29, 82–97. doi: 10.1109/MSP.2012.2205597

Huang, Y., Liu, J., Harkin, J., McDaid, L., and Luo, Y. (2021). An memristor-based synapse implementation using BCM learning rule. *Neurocomputing* 423, 336–342. doi: 10.1016/j.neucom.2020.10.106

Hubel, D. H., and Wiesel, T. N. (1977). The functional architecture of the macaque visual cortex. *Ferrier Lect.* 198, 1–59. doi: 10.1098/rspb.1977.0085

Joglekar, Y. N., and Wolf, S. J. (2009). The elusive memristor: properties of basic electrical circuits. *Eur. J. Phys.* 30, 661–675. doi: 10.1088/0143-0807/30/4/001

Johansson, C., and Lansner, A. (2007). Towards cortex sized artificial neural systems. *Neural Netw.* 20, 48–61. doi: 10.1016/j.neunet.2006.05.029

Johnson, S., Sundararajan, A., Hunley, D., and Strachan, D. (2010). Memristive switching of single-component metallic nanowires. *Nanotechnology* 21, 125204. doi: 10.1088/0957-4484/21/12/125204

Kvatinsky, S., Friedman, E. G., Kolodny, A., and Weiser, U. C. (2013). TEAM: ThrEshold adaptive memristor model. *IEEE Trans. Circ. Syst. I Regul. Pap.* 60, 211–221. doi: 10.1109/TCSI.2012.2215714

Kvatinsky, S., Ramadan, M., Friedman, E. G., and Kolodny, A. (2015). VTEAM: a general model for voltage-controlled memristors. *IEEE Trans. Circ. Syst. II Express Briefs* 62, 786–790. doi: 10.1109/TCSII.2015.2433536

Lansner, A., and Ekeberg, Ö. (1989). A one-layer feedback artificial neural network with a Bayesian learning rule. *Int. J. Neural Syst.* 1, 77–87. doi: 10.1142/S0129065789000499

Lansner, A., Hemani, A., and Farahini, N. (2014). "Spiking brain models: computation, memory and communication constraints for custom hardware implementation," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Singapore), 556–562.

Lansner, A., and Holst, A. (1996). A higher order Bayesian neural network with spiking units. *Int. J. Neural Syst.* 7, 115–128. doi: 10.1142/S0129065796000816

Le, B. Q., Grossi, A., Vianello, E., Wu, T., Lama, G., Beigne, E., et al. (2018). Resistive RAM with multiple bits per cell: array-level demonstration of 3 bits per cell. *IEEE Trans. Electron. Devices* 66, 641–646. doi: 10.1109/TED.2018.2879788

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Li, C., Graves, C. E., Sheng, X., Miller, D., Foltin, M., Pedretti, G., et al. (2020a). Analog content-addressable memories with memristors. *Nat. Commun.* 11, 1–8. doi: 10.1038/s41467-020-15254-4

Li, J., Dong, Z., Luo, L., Duan, S., and Wang, L. (2020b). A novel versatile window function for memristor model with application in spiking neural network. *Neurocomputing* 405, 239–246. doi: 10.1016/j.neucom.2020.04.111

Li, Y., Chu, J., Duan, W., Cai, G., Fan, X., Wang, X., et al. (2018). Analog and digital bipolar resistive switching in solution-combustion-processed nio memristor. *ACS Appl. Mater. Interfaces* 10, 24598–24606. doi: 10.1021/acsami.8b05749

Long, Y., She, X., and Mukhopadhyay, S. (2019). "Design of reliable DNN accelerator with un-reliable ReRAM," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)* (Florence: IEEE), 1769–1774.

Lundqvist, M., Herman, P., and Lansner, A. (2011). Theta and gamma power increases and alpha/beta power decreases with memory load in an attractor network model. *J. Cogn. Neurosci.* 23, 3008–3020. doi: 10.1162/jocn_a_00029

Meli, C., and Lansner, A. (2013). A modular attractor associative memory with patchy connectivity and weight pruning. *Network* 24, 129–150. doi: 10.3109/0954898X.2013.859323

Nishitani, Y., Kaneko, Y., and Ueda, M. (2015). Supervised learning using spike-timing-dependent plasticity of memristive synapses. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2999–3008. doi: 10.1109/TNNLS.2015.2399491

Park, J.-K., Kim, S.-Y., Baek, J.-M., Seo, D.-J., Chun, J.-H., and Kwon, K.-W. (2013). "Analysis of resistance variations and variance-aware read circuit for cross-point ReRAM," in *2013 5th IEEE International Memory Workshop* (Monterey, CA: IEEE), 112–115.

Pickett, M. D., Strukov, D. B., Borghetti, J. L., Yang, J. J., Snider, G. S., Stewart, D. R., et al. (2009). Switching dynamics in titanium dioxide memristive devices. *J. Appl. Phys.* 106, 074508. doi: 10.1063/1.3236506

Podobas, A., Svedin, M., Chien, S. W., Peng, I. B., Ravichandran, N. B., Herman, P., et al. (2021). "Streambrain: an hpc framework for brain-like neural networks on cpus, gpus and fpgas," in *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 1–6.

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 61–64. doi: 10.1038/nature14441

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans. Nanotechnol.* 12, 288–295. doi: 10.1109/TNANO.2013.2250995

Ravichandran, N. B., Lansner, A., and Herman, P. (2020). "Learning representations in bayesian confidence propagation neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)* (Glasgow: IEEE), 1–7.

Ravichandran, N. B., Lansner, A., and Herman, P. (2021a). "Brain-like approaches to unsupervised learning of hidden representations-a comparative study," in *International Conference on Artificial Neural Networks* (Bratislava: Springer), 162–173.

Ravichandran, N. B., Lansner, A., and Herman, P. (2021b). Semi-supervised learning with bayesian confidence propagation neural network. *arXiv [Preprint] arXiv:2106.15546*. doi: 10.14428/esann/2021.ES2021-156

Sandberg, A., Lansner, A., Petersson, K., and Ekeberg. (2002). A Bayesian attractor network with incremental learning. *Network* 13, 179–194. doi: 10.1080/net.13.2.179.194

Stathis, D., Chaourani, P., Jafri, S. M. A. H, and Hemani, A. (2021). "Clock tree generation by abutment in synchoros VLSI design," in *Proceedings 2021 Nordic Circuits and Systems Conference (NorCAS)* (Oslo).

Stathis, D., Sudarshan, C., Yang, Y., Jung, M., Weis, C., Hemani, A., et al. (2020). eBrainII: a 3 kW realtime custom 3D DRAM integrated ASIC implementation of a biologically plausible model of a human scale cortex. *J. Signal Process Syst.* 92, 1323–1343. doi: 10.1007/s11265-020-01562-x

Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *Nature* 453, 80–83. doi: 10.1038/nature06932

Tully, P. J., Hennig, M. H., and Lansner, A. (2014). Synaptic and nonsynaptic plasticity approximating probabilistic inference. *Front. Synaptic. Neurosci.* 6:8. doi: 10.3389/fnsyn.2014.00008

Vogginger, B., Schüffny, R., Lansner, A., Cederström, L., Partzsch, J., and Höppner, S. (2015). Reducing the computational footprint for real-time BCPNN learning. *Front. Neurosci.* 9:2. doi: 10.3389/fnins.2015.00002

Wijesinghe, P., Ankit, A., Sengupta, A., and Roy, K. (2018). An all-memristor deep spiking neural computing system: a step toward realizing the low-power stochastic brain. *IEEE Trans. Emerg. Top. Comput. Intell.* 2, 345–358. doi: 10.1109/TETCI.2018.2829924

Xu, J., Huan, Y., Yang, K., Zhan, Y., Zou, Z., and Zheng, L.-R. (2018). Optimized near-zero quantization method for flexible memristor based neural network. *IEEE Access* 6:29320–29331. doi: 10.1109/ACCESS.2018.2839106

Xu, J., Wang, D., Li, F., Zhang, L., Stathis, D., Yang, Y., et al. (2021). "A memristor model with concise window function for spiking brain-inspired computation," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Washington DC: IEEE), 1–4.

Yang, J. J., Pickett, M. D., Li, X., Ohlberg, D. A., Stewart, D. R., and Williams, R. S. (2008). Memristive switching mechanism for metal/oxide/metal nanodevices. *Nat. Nanotechnol.* 3, 429–433. doi: 10.1038/nnano.2008.160

Yang, Y., Stathis, D., Jord ao, R., Hemani, A., and Lansner, A. (2020). Optimizing BCPNN learning rule for memory access. *Front. Neurosci.* 14:878. doi: 10.3389/fnins.2020.00878

Yao, P., Wu, H., Gao, B., Tang, J., Zhang, Q., Zhang, W., et al. (2020). Fully hardware-implemented memristor convolutional neural network. *Nature* 577, 641–646. doi: 10.1038/s41586-020-1942-4

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *arXiv [Preprint] arXiv:1702.01923*.

Zhao, Z., Qu, L., Wang, L., Deng, Q., Li, N., Kang, Z., et al. (2020). A memristor-based spiking neural network with high scalability and learning efficiency. *IEEE Trans. Circ. Syst. II Express Briefs* 67, 931–935. doi: 10.1109/TCSII.2020.2980054

Zhou, E., Fang, L., Liu, R., and Tang, Z. (2019). Area-efficient memristor spiking neural networks and supervised learning method. *Sci. China Inf. Sci*. 62, 1–3. doi: 10.1007/s11432-018-9607-8

Zidan, M. A., Jeong, Y., Lee, J., Chen, B., Huang, S., Kushner, M. J., et al. (2018). A general memristor-based partial differential equation solver. *Nat. Electron*. 1, 411–420. doi: 10.1038/s41928-018-0100-6

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Neuroscience

ORIGINAL RESEARCH
published: 11 April 2022
doi: 10.3389/fnins.2022.775457

MONETA: A Processing-In-Memory-Based Hardware Platform for the Hybrid Convolutional Spiking Neural Network With Online Learning

Daehyun Kim, Biswadeep Chakraborty, Xueyuan She, Edward Lee, Beomseok Kang and Saibal Mukhopadhyay*

Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, United States

We present a processing-in-memory (PIM)-based hardware platform, referred to as MONETA, for on-chip acceleration of inference and learning in hybrid convolutional spiking neural network. MONETA uses 8T static random-access memory (SRAM)-based PIM cores for vector matrix multiplication (VMM) augmented with spike-time-dependent-plasticity (STDP) based weight update. The spiking neural network (SNN)-focused data flow is presented to minimize data movement in MONETA while ensuring learning accuracy. MONETA supports on-line and on-chip training on PIM architecture. The STDP-trained convolutional neural network within SNN (ConvSNN) with the proposed data flow, 4-bit input precision, and 8-bit weight precision shows only 1.63% lower accuracy in CIFAR-10 compared to the STDP accuracy implemented by the software. Further, the proposed architecture is used to accelerate a hybrid SNN architecture that couples off-chip supervised (back propagation through time) and on-chip unsupervised (STDP) training. We also evaluate the hybrid network architecture with the proposed data flow. The accuracy of this hybrid network is 10.84% higher than STDP trained accuracy result and 1.4% higher compared to the backpropagated training-based ConvSNN result with the CIFAR-10 dataset. Physical design of MONETA in 65 nm complementary metal-oxide-semiconductor (CMOS) shows 18.69 tera operation per second (TOPS)/W, 7.25 TOPS/W and 10.41 TOPS/W power efficiencies for the inference mode, learning mode, and hybrid learning mode, respectively.

Keywords: spiking neural network (SNN), processing-in-memory (PIM), convolutional spiking neural network, on-line learning, on-chip learning, spike-time-dependent plasticity (STDP), AI accelerator, hybrid network

OPEN ACCESS

Edited by:
Irem Boybat,
IBM Research, Switzerland

Reviewed by:
Abhronil Sengupta,
The Pennsylvania State University (PSU), United States
Deliang Fan,
Arizona State University, United States

****Correspondence:***
Daehyun Kim
daehyun.kim@gatech.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *14 September 2021*
Accepted: *07 March 2022*
Published: *11 April 2022*

Citation:
Kim D, Chakraborty B, She X, Lee E, Kang B and Mukhopadhyay S (2022) MONETA: A Processing-In-Memory-Based Hardware Platform for the Hybrid Convolutional Spiking Neural Network With Online Learning. Front. Neurosci. 16:775457. doi: 10.3389/fnins.2022.775457

1. INTRODUCTION

Spiking neural network (SNN) (Maass, 1997; Gerstner and Kistler, 2002) with spike-time-dependent-plasticity (STDP) based unsupervised learning provides a bio-inspired and energy-efficient alternative to deep learning (Kim et al., 2020; Panda et al., 2020). There is a growing interest in developing specialized hardware accelerators for SNN (Akopyan et al., 2015; Buhler et al., 2017; Davies et al., 2018; Chen et al., 2019; Park et al., 2019; Chuang et al., 2020). However, majority of the prior accelerators focused on fully connected SNN and shallow networks.

Deep Convolutional Neural Network (CNN) architectures incorporated within SNN, hereafter referred to as ConvSNN, can improve the accuracy of SNNs for complex problems (Cao et al., 2015; Tavanaei et al., 2016; Kheradpisheh et al., 2018; Lee et al., 2019). As the complexity of ConvSNN increases, deep ConvSNN requires more synaptic weights and generates larger input/output feature maps, all of which can increase data movement. Processing-in-memory (PIM) has emerged as a key approach to reduce data movement and enhance the energy efficiency of CNNs (Chi et al., 2016; Shafiee et al., 2016; Imani et al., 2019; Long et al., 2020; Sze et al., 2020). However, to the best of our knowledge, there has been no prior work on PIM based accelerator for ConvSNN with on-chip learning.

This article for the first time presents a PIM, hereafter referred to as MONETA, to accelerate ConvSNN with on-chip STDP learning. The overall architecture of MONETA includes SRAM-based PIM cores for computing synapse responses, all-digital modules for computing membrane potentials of neurons, and centrally manage but locally apply STDP-based weight update. The SRAM-based PIM cores augment the sequential access PIM used in DNN acceleration, such as the ones presented by Long et al. (2020), with STDP-based weight update modules for parallel updates of synaptic weights (Kim et al., 2020). The novelty of MONETA lies in the optimized data flow for improving resource efficiency while implementing inference and learning in PIM-based ConvSNN.

In traditional CNN, the output feature map (OFM) tensor of a layer is obtained from the total input feature map (TIFM) tensor and filter weights (**Figure 1A**). In ConvSNN, we first generate a tensor for the membrane potential of all neurons (TV_{mem}), followed by output spikes (OFMs) (**Figure 1B**). However, as input pixels are encoded as spike trains, multiple time steps (spike cycles) are necessary to process one image using ConvSNN. Hence, the TIFM for each layer must be processed multiple times to generate the TV_{mem} in each spike cycle, leading to a large on-chip buffer for TV_{mem} tensor, and significant off-chip (from DRAM) and on-chip (from TV_{mem} buffer) data movement. Although, Narayanan et al. have analyzed the temporal aspects of SNN for logic-based engines (Narayanan et al., 2020), they did not optimize data flow simultaneously considering data movement and learning accuracy in ConvSNN.

We propose a novel data flow for the PIM-based processing of the TIFM. We read an input feature map (IFM) from the TIFM tensor, process the IFM using PIM, and generate the V_{mem} for output neurons. The sequential processing of an IFM overall spike cycles eliminates repeated reading of TIFM from DRAM and on-chip storage of TV_{mem}. However, the sequential processing of IFMs introduces a bias in the STDP learning as IFMs processed earlier more strongly influence filter weights than the ones processed later. We propose a central STDP controller to ensure each filter is updated based on the IFM that results in the maximum V_{mem} of the firing neuron, rather than the IFMs that were processed earlier in sequence. In summary, our approach minimizes the data movement during inference, while ensuring the accuracy of the STDP learning process.

The accuracy of the accelerator is estimated considering MNIST, CIFAR-10, and CIFAR-100 datasets. With CIFAR-10 dataset, the accuracy with the weights trained by the standard

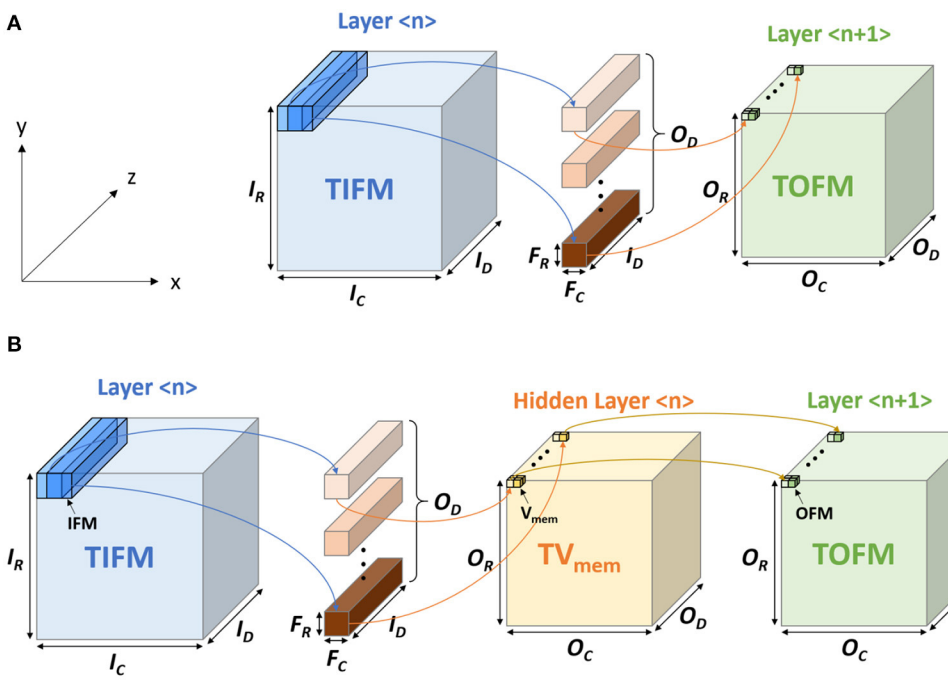


FIGURE 1 | The computational model in **(A)** convolutional neural network (CNN) and the **(B)** convolutional neural network within spiking neural network (ConvSNN).

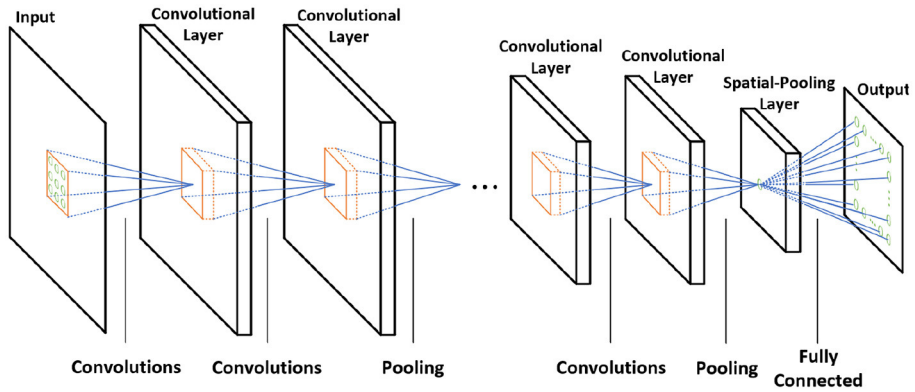


FIGURE 2 | The architecture of the CNN.

STDP model is 67.88%. When we apply our modified STDP model, the accuracy is 66.25%, which is 1.63% lower than standard STDP model-based result. The experiment result demonstrates that on-chip and on-line STDP learning can be achieved with insignificant accuracy loss. The average power efficiencies of 18.69 TOPS/W and 7.25 TOPS/W are observed for inference and learning, respectively.

Along with a fully-STDP trained ConvSNN, the proposed architecture is also used to accelerate inference and on-line learning of a hybrid ConvSNN architecture that couples supervised (off-chip) trained and STDP (on-chip) learned layers. Previously, the concept of hybridization combining supervised training and STDP has been first introduced for a DNN (She et al., 2021). After that, Chakraborty et al. has shown the same concept of hybridization on SNN (Chakraborty et al., 2021). In this article, we show the hardware platform to accelerate the ConvSNN using the same concept of hybridization.

In addition to homogeneous networks, MONETA also supports hybrid ConvSNN. Half of the layers can be on-line trained using the STDP algorithm and the other half of the layers are based on the externally programmed fixed weights. These fixed weights are off-chip trained by supervised learning. STDP uses unsupervised local learning to extract low-level features under spatial correlation. On the other hand, surrogate-gradient based backpropagation (BP) in ConvSNN enables global learning between low-level pixel-to-pixel interactions (Wu et al., 2018). It thus aids in high-level detection and classification similar to a SGD trained CNN model. By integrating global features using supervised training and local features using STDP learning, the hybrid network is also much more robust to local uncorrelated perturbations in pixels while extracting the correct feature representation from the overall image. Consequently, hybridization of surrogate-gradient and STDP enables robust image classification improving the accuracy of the baseline backpropagated ConvSNN model.

Based on the hybrid network simulation, we achieve 1.40% higher accuracy (77.83%) in MONETA than the accuracy based on the supervised learning (76.43%) with the CIFAR-10 dataset. In addition, the average power efficiency for the hybrid on-line learning mode is 10.41 TOPS/W. This power efficiency is larger than on-line learning mode, but smaller than inference mode because half of the layer use inference mode and the other half of the layers use learning mode.

2. BACKGROUND

2.1. ConvSNN and Unsupervised Learning Using STDP

The spiking CNN uses the same structure as a traditional CNN (**Figure 2**). However, the input is a binary spike where the magnitude of the input. For example, the value of an image pixel is encoded in the frequency of the spikes. A spiking neuron computes the membrane potential V_{mem} using the spike levels multiplied by synaptic weights following the leaky integrate and fire (LIF) dynamics (**Figure 3**). An output spike is generated (neuron firing) when V_{mem} is higher than a threshold V_{th} and resets V_{mem} to V_{reset}.

When the neuron fires (i.e., generates an output spike), the synaptic weights connected to the spiked neuron are updated following a stochastic STDP model (**Figure 3B**) (She et al., 2019). The firing of the neuron inhibits the firing of other neurons. There are two types of inhibitions, which are cross-depth inhibition and lateral inhibition. **Figure 4A** shows the cross-depth inhibition. In the case of the cross-depth inhibition, the firing of a neuron inhibits the firing of all other neurons located at the same (x, y) coordinates of all depths (across "z"-axis) in TV_{mem}. The cross-depth inhibition can be easily implemented within the single PIM array and the neuron set (**Figure 4B**). In the case of lateral inhibition, the firing of the neuron inhibits all the neurons located at the same z coordinates (**Figure 4C**).

2.2. CNN Mapping for PIM Architecture

Figure 1A shows the basic terminologies for the CNN hardware (Chen et al., 2017). In the layer <n>, the size of TIFM is $I_R \times I_C \times I_D$, the size of the filter is $F_R \times F_C \times I_D$, and the size of total output feature maps (TOFM) is $O_R \times O_C \times O_D$. The number of filters (depth of filters) are the same as the TOFM's

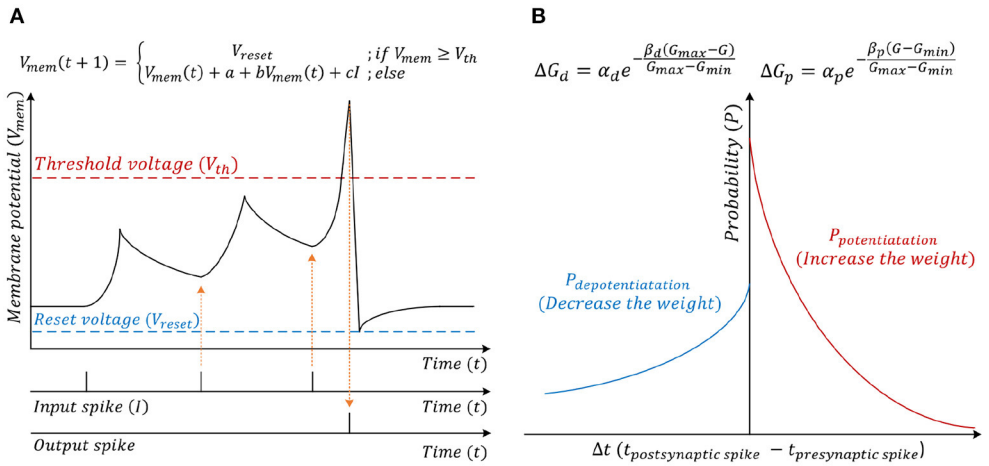


FIGURE 3 | (A) Leaky integrate and fire (LIF) neuron computational model. **(B)** Stochastic spike-time-dependent-plasticity (STDP) model.

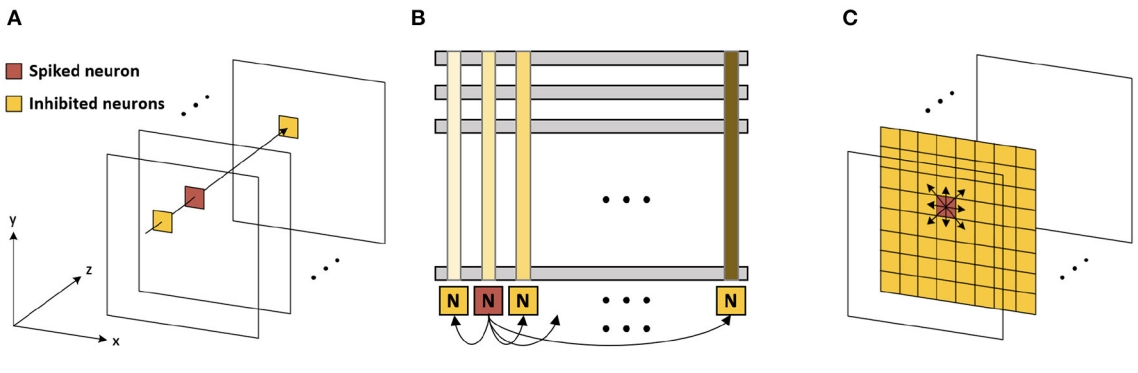


FIGURE 4 | (A) Cross-depth inhibition **(B)** cross-depth inhibition on the memory array **(C)** lateral inhibition.

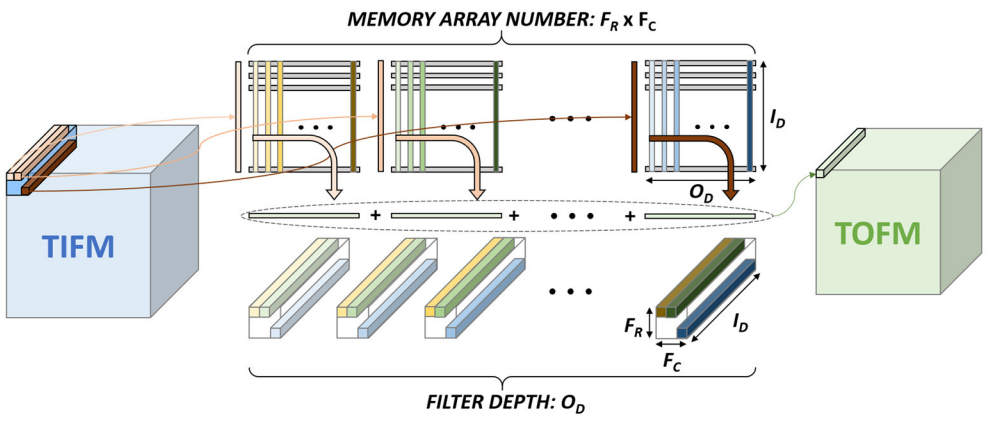


FIGURE 5 | CNN mapping on the memory for processing-in-memory (PIM) architecture.

depth (O_D). The IFM, whose size is $F_R \times F_C \times I_D$, is multiplied by each filter and generates the OFM, which size is $1 \times 1 \times 1$. The stride is called as *S*. **Figure 5** shows the CNN mapping method on the memory for the PIM architecture (Peng et al., 2021). Filter weights are divided by the x and y-axis, whose size is $1 \times 1 \times I_D$ and distributed on the different memory arrays. Also, each filter

is placed on the different columns. To calculate the OFM, IFM is divided and sent to the memory sub-arrays. The multiplication between the synapse matrix and input vector is computed in each array, and outputs are summed to compute the OFM.

2.3. Prior SNN Accelerator Hardware

Various types of SNN based accelerators have been introduced in recent years. Buhler et al. (2017) made the analog neuron-based accelerator for the compact and energy-efficient design. However, they use the spiking locally competitive algorithm for an accelerator. Chen et al. (2019) showed the large-scale neuromorphic processor with 4,096-neuron and 1M-synapse. Their design uses binary activation, but the hardware is not optimized for the ConvSNN (Chen et al., 2019). Park et al. (2019) showed the ConvSNN based accelerator. However, they only used the stochastic gradient descent algorithm for the learning to improve the accuracy. In addition, the ConvSNN accelerator is introduced by Chuang et al. using a 2D systolic array with efficient data re-use, but their design does not include the on-chip training (Chuang et al., 2020).

Our design accelerates the ConvSNN using PIM architecture with on-chip STDP learning. The ConvSNN requires more complicated hardware design than multilayer perceptron-based SNN, but it has higher accuracy and lower memory usage for the weights on the complex image datasets such as CIFAR-10. The PIM architecture does not require the VMM calculation module, as we calculate the VMM in the SRAM array. In this sense, the PIM architecture can reduce the data transmission, as it does not require transmitting the weights to another module. In addition, the STDP learning rule benefits efficient learning for large-scale models or on-line learning as it enables unsupervised local learning. We propose a modified STDP algorithm to efficiently accelerate the PIM architecture.

2.4. Hybrid Spiking Neural Network

The SNN training methodologies can be broadly classified into three types: (1) conversion from artificial-to-spiking models (Diehl et al., 2015; Sengupta et al., 2019), (2) surrogate gradient descent based backpropagation with spikes (Lee et al., 2018; Wu et al., 2018; Neftci et al., 2019), and (3) unsupervised STDP based learning (Diehl and Cook, 2015; Srinivasan et al., 2018). Each technique has its own set of advantages and disadvantages. ANN-to-SNN conversion yields state-of-the-art accuracies, even for complex datasets like ImageNet (Deng et al., 2009) and can be used to convert complex architectures, like VGGNet (Simonyan and Zisserman, 2014), ResNet (He et al., 2016), RetinaNet (Miquel et al., 2021), the latency incurred to process the rate-coded image is very high (Pfeiffer and Pfeil, 2018; Lee et al., 2020). Surrogate gradient-based methods address the latency concerns but lag behind conversion in terms of accuracy for larger and complex tasks. The unsupervised STDP training also suffers from accuracy deficiencies. As pointed out by Panda et al. (2020), the accuracy loss due to vanishing spike propagation and input pixel-to-spike coding are innate properties of SNN design that can be addressed to a certain extent, but, cannot be completely eliminated. In order to achieve competitive accuracy as that of an ANN, previous works have taken a hybrid approach with a partly-artificial-and-partly-spiking neural architecture (Panda et al., 2020; She et al., 2020). As discussed by Ledinauskas et al. (2020), SNNs obtained by conversion must use only rate encoding, due to which the expressive capacity might be reduced. Another drawback of such conversion using rate-based encoding is that one needs to use forward propagation time steps in the order of thousands during the inference procedure for SNN. This drawback severely limits the computation speed and energy efficiency benefit of SNNs. Large spikes are necessary to reduce the uncertainty of spiking frequency values. Also, several ANN architectures are limited before conversion (e.g., batch normalization cannot be used) (Diehl et al., 2015; Sengupta et al., 2019). This limits ANN performance and the upper bound of SNN performance. Due to these limitations, we use a surrogate gradient-based method to train SNNs directly instead of converting ANN parameters to SNN.

Hence, following the work done by Chakraborty et al. (2021), we use a hybrid network consisting of surrogate-gradient based backpropagated ConvSNN modules along with the unsupervised STDP trained ConvSNN module. **Figure 6** shows the architectural block diagrams of the different types of neural networks. **Figures 6A,B** show the homogeneous network architecture that uses STDP and the backpropagation, respectively. **Figure 6C** shows the hybrid network architecture whose weights are from the different training algorithms. The hybrid network consists of spiking layers placed in parallel to form different spiking convolution modules. The first spiking convolution module and half of the third spiking convolutional module (shown in blue in **Figures 6A,C**) are the backpropagated spiking modules. The second spiking convolutional module and the other half of the third spiking convolutional module (shown in orange in **Figures 6B,C**) are trained with the unsupervised STDP algorithm. The STDP-spiking convolution module is placed in parallel to the backpropagated module to enable robust extraction of local and low-level features. Further, to ensure that the low-level feature extraction also considers global learning, which is the hallmark of gradient back-propagation, several backpropagated ConvSNN layers of a similar size in parallel with the STDP ConvSNN module are used. The output feature map of the two parallel modules is maintained to have the same height and width and concatenated along the depth to be used as input tensor to the final ConvSNN layers. This ConvSNN module is responsible for higher level feature detection as well as the final classification. The main CNN module can be designed based on existing deep learning models. The concatenation of features from backpropagation-based ConvSNN and STDP-based ConvSNN modules help integrate global and local learning.

In addition, there exist other types of hybridization in the prior works. Lee et al. (2018) show the STDP-based unsupervised pre-training followed by supervised fine-tuning to improve the accuracy. Other works also show ANN-SNN hybridization that uses both ANN and SNN (Deng et al., 2020; Singh et al., 2020; Wang et al., 2021). On the other hand, hybridization in this article means, using only SNN with the different types of weight training algorithms (pre-trained backpropagation and STDP-based on-line leaning).

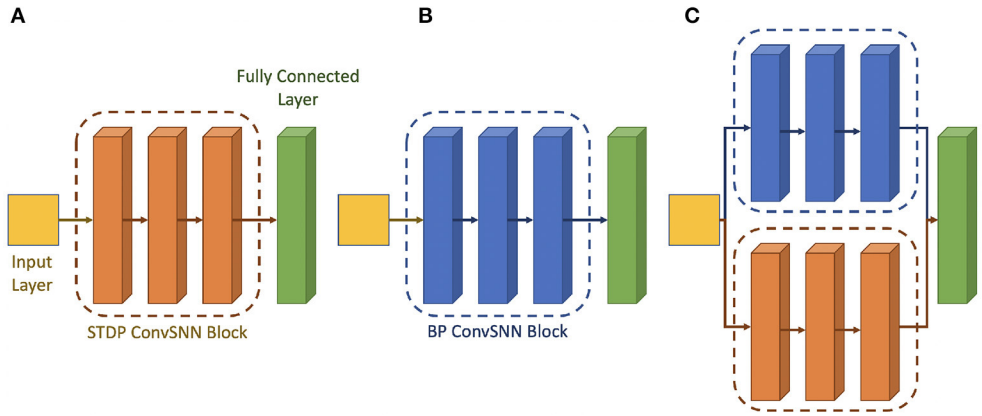


FIGURE 6 | Architectural block diagram of the **(A)** STDP only network **(B)** backpropagation only network **(C)** hybrid Network.

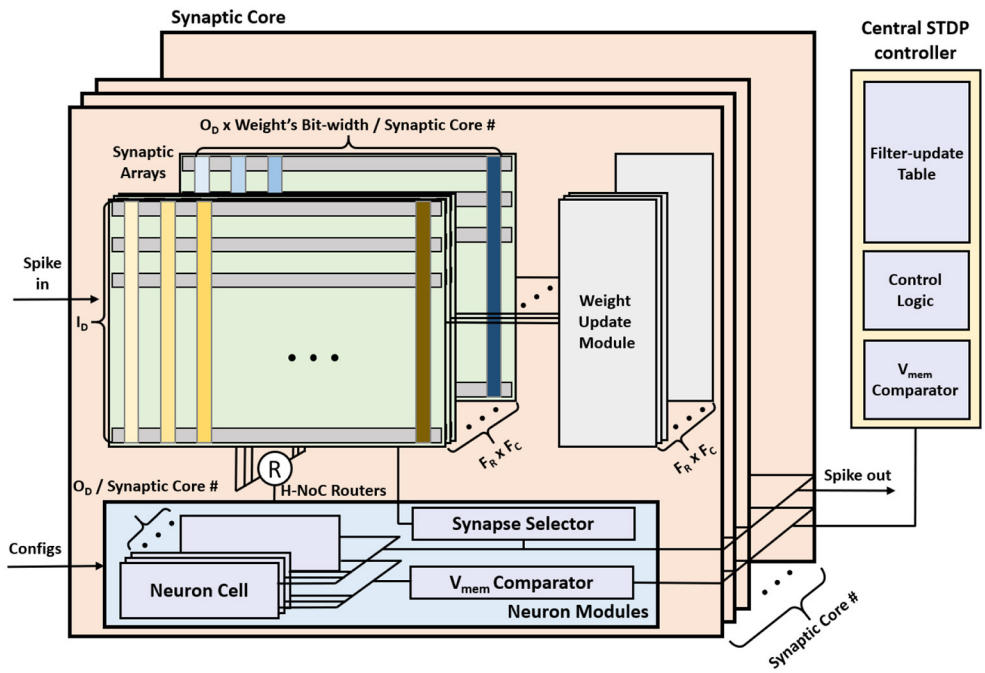


FIGURE 7 | The MONETA system architecture overview.

3. HARDWARE ARCHITECTURE

The overall MONETA architecture consists of synaptic cores, neuron modules, and a central STDP controller (**Figure 7**). The synaptic core calculates the V_{mem} for each filter based on the IFMs and the weights. The synaptic array inside the synaptic core functions as a digital PIM core and calculates the vector matrix multiplication (VMM) of IFMs and synaptic weights. The results generated by the synaptic array are accumulated in the neuron module. The neuron module generates the output spikes based on the accumulated V_{mem} using the LIF model. The central STDP controller has a filter-update table and the training control module to control the synaptic core and the STDP-based weight update.

The STDP learning is performed using distributed weight update modules embedded in each synaptic core and a central STDP controller (**Figure 7**). The weight update module reads, computes the update, and writes back the synaptic weights using stochastic STDP (Kim et al., 2020). The central STDP controller manages the filter-update table and the learning process.

There are two phases in our design, the inference phase and weight update phase. In the inference mode, only the inference phase exists. In the learning mode, both the inference phase and the weight update phase exist. More precisely, during the

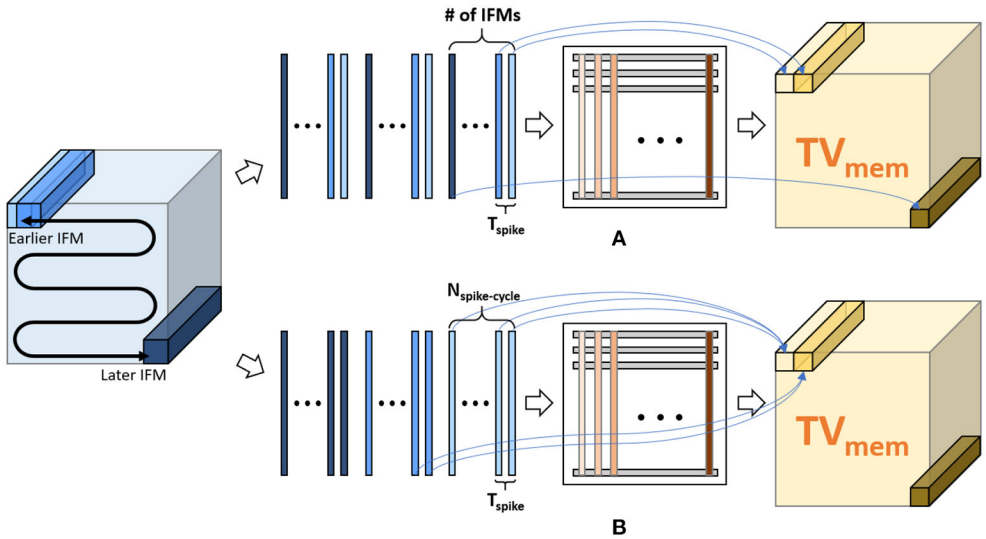


FIGURE 8 | Inference approaches: **(A)** sequential processing of spike cycles: serially process all input feature maps (IFMs) in a total IFM (TIFM) for a given spike cycle and generate the entire TV_{mem}. **(B)** sequential processing of IFMs: an IFM is serially processed for $N_{\text{spike-cycle}}(= \frac{T_{\text{total}}}{T_{\text{spike}}})$ spike cycles followed by processing the next IFM.

inference phase in the learning mode, the central STDP controller collects the data in the filter update table while other modules do the same function with inference mode. After finishing the inference function for the scheduled cycles, MONETA starts the weight update phase and updates the weights.

3.1. Proposed SNN Inference Methodology

An SNN receives the input as spikes. Based on each pixel's brightness, the range of the spike frequency is $f_{\text{spike-min}} \sim f_{\text{spike-max}}$. Assume, $T_{\text{spike}}(= \frac{1}{f_{\text{spike-max}}})$ is a unit time-step, and T_{total} is a total exposure time for an input image. Therefore, ConvSNN (**Figure 1B**) receives all the IFMs, including the input image, for $N_{\text{spike-cycle}}(= \frac{T_{\text{total}}}{T_{\text{spike}}})$ of spike cycles, computes the V_{mem} for all neurons, i.e., entire TV_{mem} tensor in each cycle based on the LIF neuron policy (**Figure 3A**). All the V_{mem} values in the TV_{mem} tensor that are higher than the threshold generate an output spike in the OFM.

3.1.1. Sequential Processing of Spike Cycles

Ideally, at spike cycle i, we need to generate a TV_{mem} tensor which is used along with the TIFM tensor to compute TV_{mem} for cycle $i+1$. Hence, in each spike cycle, we must compute all V_{mem} values in the TV_{mem} tensor by processing the all the IFMs in the TIFM tensor (**Figure 8A**). As all the IFMs are multiplied by same weight matrix, parallel processing of all the IFMs will require duplication of weight memory by $\frac{I_R}{S} \times \frac{I_C}{S}$ (where S is the stride), which is infeasible to store on-chip. Hence, we must process each IFMs serially in each accelerator clock cycle ($f_{\text{CLK}} = 1$ GHz in our design), as shown in **Figure 8A**). Hence, for each spike cycle, we can serially read all IFMs, multiply each IFM to all the filters in one accelerator clock f_{CLK}, and serially compute all the elements of the TV_{mem} tensor. This is similar to operating a normal CNN. However, in ConvSNN, we must process the same TIFM tensor repeatedly for $N_{\text{spike-cycle}}$ spike cycles in ConvSNN, such an approach requires either reading the same data (IFMs) from the off-chip memory repeatedly in every spike cycle resulting in a significant ($\frac{T_{\text{total}}}{T_{\text{unit}}} \times$) increase in data movement or store the TIFM tensor on-chip requiring large buffer. Moreover, we also need a global buffer to store the TV_{mem} tensor generated over the entire spike cycle. While processing spike cycle $i+1$, the global TV_{mem} buffer generated in the spike cycle i must be read by individual PIM blocks to generate the TV_{mem} tensor for the $i+1$ spike cycle. We will also need a global on-chip buffer of size $O_R \times O_C \times O_D$ to store the TV_{mem} tensor increasing on-chip data movement between the PIM cores.

3.1.2. Sequential Processing of IFMs

We propose to re-order the IFM processing as shown in **Figure 8B**. We first read one IFM and serially compute all V_{mem} values generated by that IFM over all $N_{\text{spike-cycle}}$ spike cycles. Note, all these V_{mem} values can now be computed in $N_{\text{spike-cycle}}$ of accelerator clock cycle. Moreover, as the IFM remains constant, the V_{mem} values for successive spike cycles can be locally accumulated within the PIM block eliminating the need for global TV_{mem} buffer and associated data movement. Moreover, serial processing of all spike cycles for a given IFM eliminates the need for repeated reading of the entire TIFM tensor thereby reducing off-chip data movement.

3.2. Hardware Support for Inference

3.2.1. Synaptic Core

The synaptic core is used for distributed computation of V_{mem} and generates the output spike. The synaptic core uses synaptic arrays (weight storage), routers, and neuron modules for the inference. Since the weight matrix is distributed across multiple synaptic cores, each synaptic core has a subarray of dimension

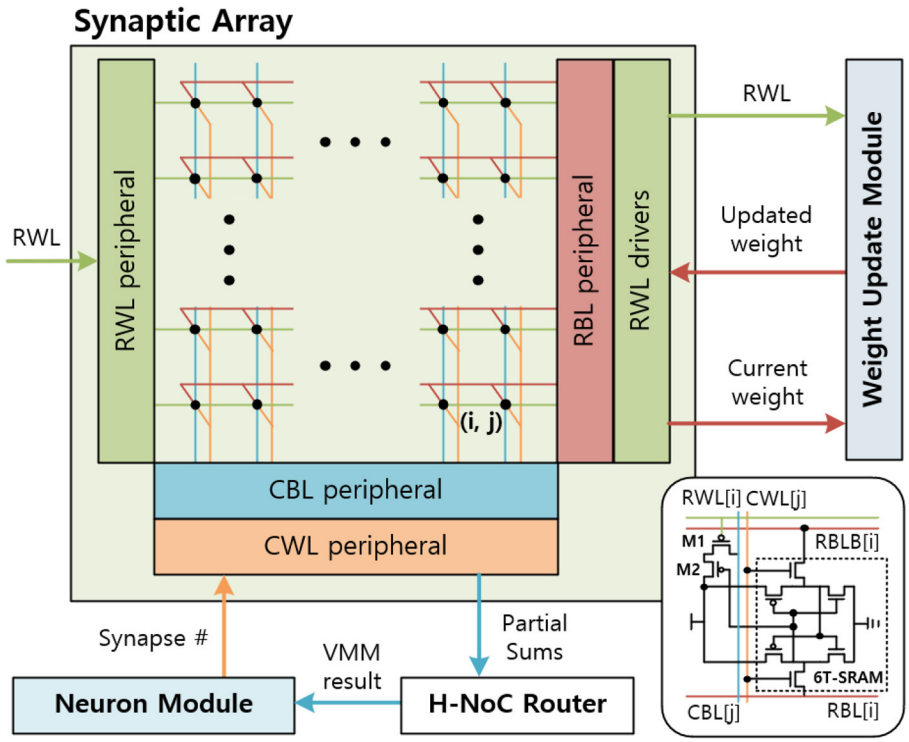


FIGURE 9 | The synaptic array architecture.

$I_D \times \frac{O_D \times \text{weight's bit-width}}{\#\text{ of synaptic core}}$, and calculates the matrix multiplication results for $\frac{O_D}{\text{synaptic core \#}}$ filters.

3.2.2. Synaptic Array

The synaptic array multiplies the IFMs and synaptic weights to generate the partial sum of the VMM result. A sequential (row-by-row) read access-based PIM design is considered for synaptic arrays to multiply IFMs and weights. Then, the hierarchical network-on-chip (H-NoC) router adds partial sums and sends the VMM result to the neuron module. The synaptic array is implemented by SRAM array, peripherals, and drivers (**Figure 9**). Synaptic weights are 8 bits and consist of 8 consecutive SRAM cells in a row. The most left SRAM cell represents the sign bit.

The synaptic array receives the input spikes of the IFM on the row-wise word-line (RWL) port. The input spikes are sent in row-by-row order, so the RWL peripheral uses a counter-based decoder to send input spikes to an 8T-SRAM array sequentially. When the result of sense amplifier for CBL is 1, the CBL peripheral sends the partial sum, 1, to the H-NoC router and pre-discharges the CBL. The H-NoC connects the synaptic arrays, accumulates the partial sums, and sends the VMM result to the neuron module (Long et al., 2019).

3.2.3. Neuron Module

The neuron module receives the VMM result from the synaptic arrays, calculates the V_{mem}, and generates the output spike. **Figure 10** shows the neuron module architecture. The neuron module consists of $\frac{O_D}{\text{synapse core \#}}$ neuron cells, V_{mem} comparator, and synapse update selector. The neuron cell updates the V_{mem} based on the LIF neuron dynamics and generates the output spike when the V_{mem} over the V_{th}. V_{mem} comparator and Synapse update selector are disabled during inference. These modules are discussed in section 3.3

To generate the output spike, the neuron cell receives the VMM result and updates the V_{mem} based on the LIF neuron dynamics. Inside the V_{mem} calculation module (**Figure 10**), VMM result and the current V_{mem} are added when the compute enable signal is enabled. This V_{mem} accumulation takes I_D cycles, as the whole VMM computation requires I_D cycles with row-by-row access on synaptic array. Then, the leakage calculator calculates the leakage based on the current V_{mem} and subtracts the leakage to the result of the V_{mem} accumulator to generate the updated V_{mem}. In the end, if the updated V_{mem} is larger than V_{th}, the neuron cell will generate the output spike and reset V_{mem} as 0. Updated V_{mem} is stored in the V_{mem} register inside the neuron cell to be used in the next time step.

3.3. Proposed PIM-Friendly STDP Learning Methodology

We argue that the proposed approach of sequential processing of IFMs can lead to bias in STDP learning. In ConvSNN with cross-depth inhibition, each depth controls the weight update for a filter tensor. Consider a neuron at location x_k, y_k, z_k fires, then it will inhibit the firing of all other neurons

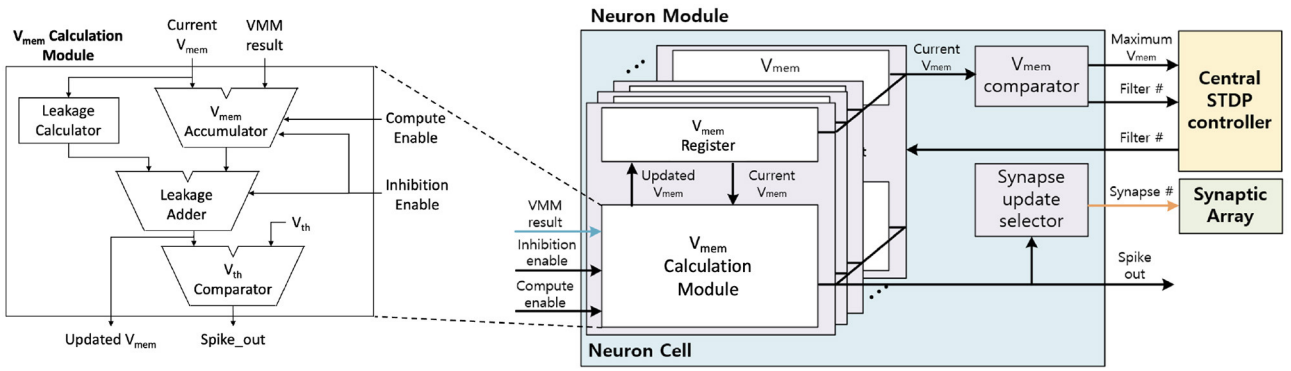


FIGURE 10 | The LIF Neuron Module architecture.

across the depth, i.e., all neurons at x_k, y_k but all locations across the z-axis. In an ideal case, V_{mem} values of all the neurons in the same depth of the TV_{mem} tensor are calculated simultaneously. Hence, for a given depth, the neuron with the maximum V_{mem} considering the entire TIFM will fire and control the weight update process for the associated filter. However, when IFMs are processed sequentially, the STDP based updates of filter weights are controlled by the order in which IFMs are processed. For example, considering the order shown in **Figure 8B**, the IFM in the earliest position (top-left segment in the TIFM tensor) can cause firing at a given depth change with the associated filter weights. The V_{mem} computation for the later IFMs will be performed with the already changed filter weights and hence will have less impact on overall learning. This leads to undesired sequential bias in the STDP learning.

We address this problem by ensuring that at a particular depth the neuron which has the maximum V_{mem} considering all IFMs control STDP-based update of the corresponding filter weight (shown in **Figure 1B**). This is achieved by maintaining a central filter-update table where for each filter we store a running value of the maximum V_{mem} and corresponding IFM number (**Figure 11A**). While processing the "ith" IFM over all spike cycles, we compute V_{mem}, fire a neuron (as required), reset V_{mem} for all other cross-depth neurons but do not initiate weight update. Instead, we estimate the V_{mem} values for all the neurons at all depths due to the "ith" IFM. If at a given depth, the V_{mem} generated by "ith" IFM is higher than the maximum V_{mem} value stored in the table for the corresponding filter, we update the central table to indicate "ith" IFM results in the maximum V_{mem} for this filter. The table generation is finished after processing all the IFMs. Once completed, we show all the IFMs one more time and update the filter weights based on the filter-update table (**Figure 11B**). The overhead is cost of processing TIFMs two times, one for generating the filter-update table and the second for updating the weights (**Figure 3B**). Therefore, our PIM-friendly STDP learning can train the weights based on the STDP algorithm without considerable IFM movements and the bias occurring from the sequential IFM processing.

3.4. Hardware Support for Learning

3.4.1. Synaptic Core

In the learning mode, the synaptic core uses synaptic arrays (weights storage), routers, neuron modules, and weight update modules. The weight update module is power gated in the inference mode but is used in the learning mode. Synaptic core and neuron modules calculate the V_{mem} and generate the output spike. When the output spike is generated, weights are updated with the control from the central STDP controller.

3.4.2. Synaptic Array

The SRAM array in the synaptic arrays is implemented by an 8T-SRAM array. 8T-SRAM allows transposable read and write thereby allowing parallelism in weight update (Seo et al., 2011; Kim et al., 2020). **Figure 9** shows the synaptic array and its connections with other modules. The 8T-SRAM includes the 6T-SRAM, the PMOS M1, and the PMOS M2. The 6T-SRAM stores the synapse weight, and the PMOS M1 and the PMOS M2 connect RWL, synapse weight, and column-wise bitline (CBL) for the matrix multiplication. When the RWL sends the spike and the synapse weight bit is 1, CBL is charged.

During the weight update phase, the syna does the same inference function until the neuron module generates the output spike. When the output spike is generated, the synaptic array receives the synapse number from the neuron module and decodes it to generate the column-wise wordline (CWLs) to read the SRAM data stored in the 6T-SRAM cell, included in the 8T-SRAM. Total 8 CWLs are generated sequentially for each clock to read the 8-bit synapse weight information. The CWL is connected to the 6T-SRAM cells' CWL vertically and reads the data by RBL and RBLB horizontally. The RBL peripheral reads the synapse weight data for each clock and sends it to the weight update module. After the weight update module calculates the synaptic weights, RBL peripheral receives the updated synapse weights and writes them back to the 6T-SRAM cells.

3.4.3. Neuron Module

In the learning mode, V_{mem} comparator and the synapse update selector are additionally used. During the inference phase in the learning mode, the neuron module compares the V_{mem} at the

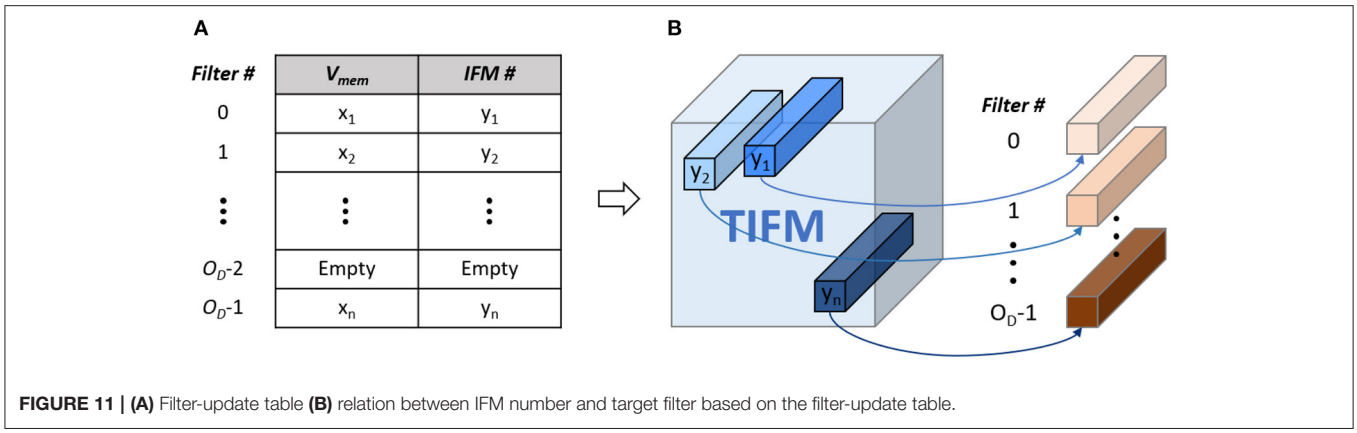


FIGURE 11 | (A) Filter-update table **(B)** relation between IFM number and target filter based on the filter-update table.

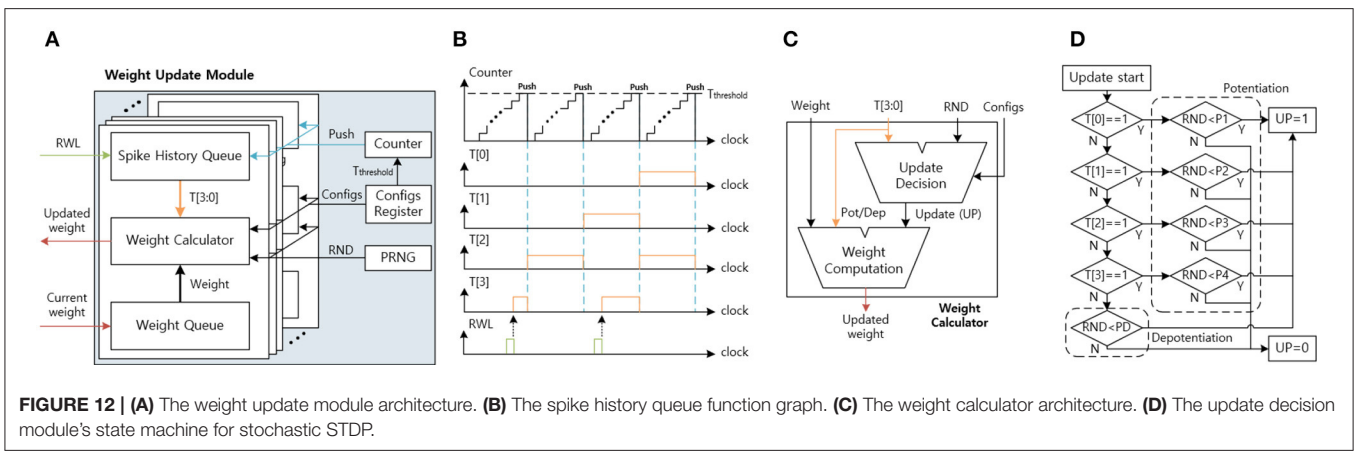


FIGURE 12 | (A) The weight update module architecture. **(B)** The spike history queue function graph. **(C)** The weight calculator architecture. **(D)** The update decision module's state machine for stochastic STDP.

V_{mem} comparator and sends the maximum V_{mem} and the filter number to the central STDP controller for each IFM. In the weight update phase, the neuron module receives the active filter number from the central STDP controller, and only the selected filter calculates the V_{mem}. The selected filter calculates the V_{mem} in the neuron cell and generates the output spike when the V_{mem} is over the threshold voltage. When the neuron cell generates the output spike, the neuron cell resets the V_{mem} to 0 and holds the V_{mem} calculation while the synapse array updates the weight. The synapse update selector receives the output spike, generates the synapse number, and sends this number to the synapse array.

3.4.4. Weight Update Module

Figure 12A shows the architecture of the weight update module. The weight update module calculates the updated weights based on the current weights and the timing information using the stochastic STDP rule. The timing information is used to check the probability of potentiation or depotentiation (**Figure 3B**). The configuration register (configs register) stores the programmable configurations for the timing queue control and the stochastic STDP rule. The pseudo-random number generator (PRNG) generates the random number (RND), which decides whether to update or not to update weights and is implemented by linear-feedback shift registers (LFSRs). The counter is used to push the spike history queue.

The spike history queue receives the RWL, delivered from the synaptic array, and stores the input spike history in the spike history queue (**Figure 12B**). The T[3] is connected to the RWL and is set to 1 when the RWL is 1. When the counter reaches the threshold time ($T_{\text{threshold}}$), reset the T[3] to 0 and push the queue from T[3:1] to T[2:0]. As a result, the spike history queue stores the input spike history, and by changing the $T_{\text{threshold}}$ in the configs register, we can control the spike history period.

The weight queue receives the current weight from the synaptic array one bit per clock until it receives all 8-bit of the synapse weight. After the weight queue receives the current weight, the weight calculator calculates the updated weight based on the spike history and the current weight (**Figure 12C**). The weight update calculator includes the update decision module and the weight computation module. The update decision module determines whether to update the weight or not according to the stochastic STDP rule. When the update (UP) signal is 1 and the T[3:0] is all 0, the weight computation module decreases the weight. When the UP signal is 1 and the T[3:0] has at least 1, the weight computation module increases the weight. At the end, when the UP signal is 0, the weight computation module does not change the weight. After the updated weight calculation, the weight update module sends the updated weight one bit per clock to the synaptic array.

As shown in **Figure 12D**, the update decision module's state-machine describes the stochastic STDP. The update decision module receives the input spike history, the RND, and the configurations (configs). The configurations include the potentiation thresholds (P1, P2, P3, and P4) and the depotentiation threshold (PD). The spike history determines which potentiation/depotentiation threshold will be used. The UP is set to 1 when the RND is smaller than the selected threshold. When the RND is equal to or larger than the threshold, UP is set to 0.

3.4.5. Central STDP Controller

The central STDP controller includes SRAM which stores the filter update table, the V_{mem} comparator to find the maximum V_{mem} of MONETA, and the control logic to control MONETA. The central STDP controller controls the design and determines the weights to update during the learning mode. **Figure 13** shows the architecture of the central STDP controller.

During the inference phase in the learning mode, MONETA fills the filter update table. To generate the data for the filter update table, the synaptic core receives the IFM and calculates the membrane potential. During this process, the neuron modules calculate the V_{mem} and send the maximum V_{mem} and the corresponding filter number to the central STDP controller. The central STDP controller compares the current IFM's maximum V_{mem} and the previous IFM's V_{mem} which is stored in the filter update table. If the current IFM's maximum V_{mem} is larger, the filter update table will store the current IFM and the new maximum V_{mem} in the filter update table. When the weight update phase starts, the central STDP controller reads the IFM for the maximum V_{mem} from off-chip memory for each filter. The IFM is applied to the target synaptic core to re-compute the corresponding V_{mem}, generate the output spikes, and update the weights using the weight update modules. Once all the filters are updated for a TIFM, the filter update table is reset.

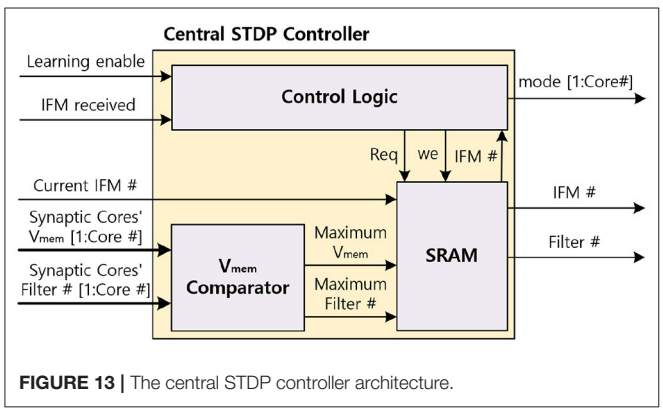


FIGURE 13 | The central STDP controller architecture.

Figure 14 indicates the data flow timing diagram of the central STDP controller. The central STDP controller receives the mode signal from the user to determine the mode of the synaptic core. The central STDP controller controls the function of synaptic cores by sending the phase and mode signals to the synaptic cores. During the inference phase in the learning mode, the central STDP controller receives the maximum V_{mem} of the synaptic cores, filter number from the synaptic cores, and the current IFM number from the off-chip memory to generate the filter update table. In the weight update phase, the control logic request the IFM number to the filter update table in the SRAM with the request (Req) signal. The filter update table sends the updating target IFM number to the off-chip memory and the updating target filter number to the updating target synaptic core. The received IFM signal transmitted from the off-chip memory is used to determine whether the IFM is delivered from off-chip memory during the weight update phase. This is because the central STDP controller sends non-sequential IFM requests to the off-chip memory, so the MONETA needs to be in the idle state until the IFM is transmitted. The control logic also receives

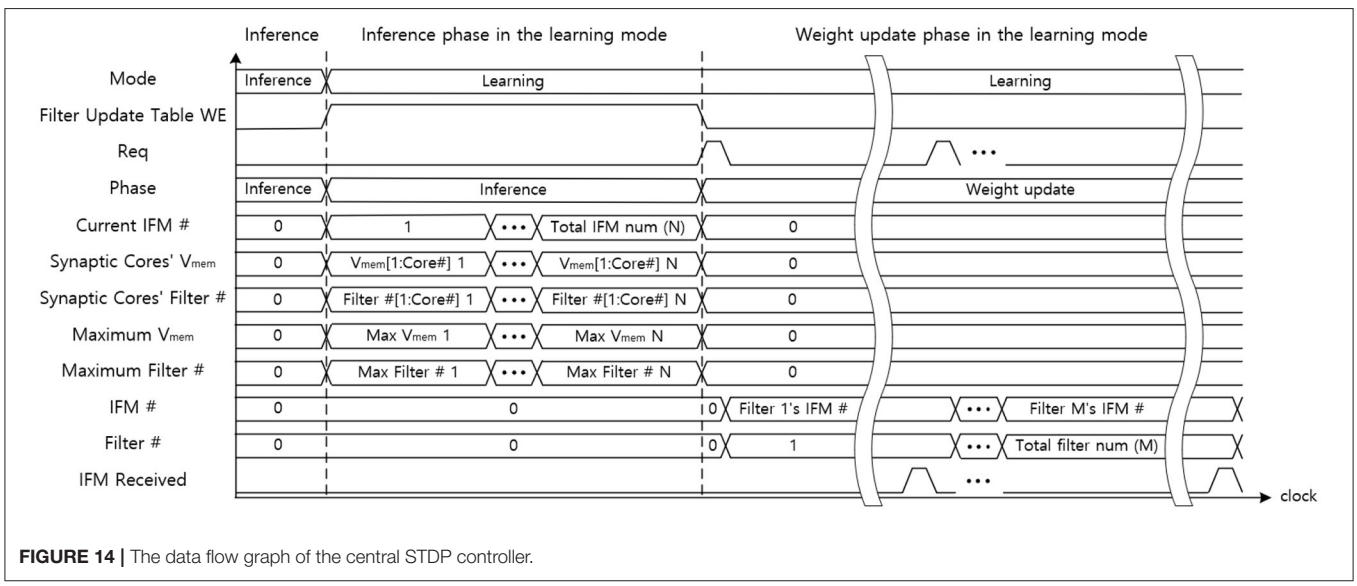


FIGURE 14 | The data flow graph of the central STDP controller.

the filter number. After that, the control logic generates the mode signal to the target synaptic core. Only the updating target synaptic core is enabled to update weights by mode signal and other synaptic cores are in the idle state as they do not need to update the synapse weights. This process is continued for all the filters in the filter update table.

3.5. Hybrid Network With Coupled Supervised and Unsupervised Learning

As we discussed in section 2.4, hybrid networks can help us improve the accuracy of the STDP network. Thus, in this article, we used a hybrid supervised-unsupervised learning methodology similar to the works done by Chakraborty et al. (2021). Supervised learning is the surrogate gradient-based training of the SNNs (Wu et al., 2018; Neftci et al., 2019). The supervised learning-based weights are trained at the off-chip, then are loaded on the synaptic array. These supervised learning-based layers are set as the inference mode (marked blue in **Figure 6C**). The unsupervised learning algorithm is our modified STDP-based learning. These layers are set as the learning mode and the weights are trained on-chip. Therefore, because our design supports on-line learning, half convolutional layers have supervised learning-based fixed weights and the other half of convolutional layers have unsupervised on-line learning-based flexible weights (marked orange in **Figure 6C**).

4. SIMULATION RESULTS

4.1. Configurations of Simulated ConvSNN

As discussed before, we use both homogeneous and hybrid networks. Hybrid networks with supervised training can help us improve the accuracy of the STDP network. Thus, we simulate the hybrid supervised-unsupervised learning methodology similar to the works done by Chakraborty et al. (2021).

Configurations: We define the type of networks to compare our hybrid spiking neural network for image classification on the MNIST and CIFAR-10 dataset as follows:

- **Standard STDP model (Type 1):** we use the 4-layer ConvSNN model trained using the standard STDP model (Bi and Poo, 1998)
- **PIM-friendly STDP model (Type 2):** we use the 4-layer ConvSNN model and train it using the modified STDP rule explained in section 3.3
- **Fully Backpropagated ConvSNN model (Type 3):** for this model, we use another backpropagated ConvSNN block instead of the STDP ConvSNN block (orange block in **Figure 6**). This makes the entire model to be trained with a surrogate gradient without any unsupervised STDP block.
- **Hybrid model with standard STDP model (Type 4):** for this model, we use the hybrid network as shown in **Figure 6C**. However, we use the standard STDP learning rule for the STDP ConvSNN block (orange block).
- **Hybrid model with PIM-friendly STDP model (Type 5):** this is the proposed model using hybridization of STDP-based ConvSNN and backpropagated-based ConvSNN blocks. The STDP learning rule used to train the STDP block is the modified STDP rule as discussed in section 3.3

Types 1–3 are based on the homogeneous network architecture. The weights of architectures in **Types 1–3** are trained by single training algorithm. **Types 4-5** are based on the hybrid network architecture we discussed in section 2.4. Half of the weights in **Types 4-5** are trained by backpropagation algorithm and the other half of the weights are trained by different STDP algorithms for each network type.

4.2. Hardware Architectures for Simulation

Table 1 shows the simulated ConvSNN network architecture with four convolutional (CONV) and one fully-connected (FC) layer. We use 8-bit precision for the weights and 4-bit precision for the input spikes. The total on-chip memory used for synaptic cores is determined by the filter size of the **CONV4** layer in the homogeneous network architecture. Therefore, we need two MONETA chips for the **CONV4** layer in the hybrid network. We divide the total capacity into 8 synaptic cores where each core has nine 128 × 128 synaptic arrays. We consider on-chip STDP is performed using a layer-by-layer fashion because OFMs for one layer are used to train the next layer. Note the memory capacity is sufficient to simultaneously map **CONV1**, **CONV2**, and **CONV3** on the chip during inference. We consider that the **FC** layer exists off-chip and connected with MONETA.

The hardware architecture of MONETA with 8 synaptic cores and one central STDP controller is implemented in 65 nm CMOS (**Figure 15**). We used the Virtuoso for the full-custom layout of 128 × 128 SRAM sub-arrays and Innovus for the auto place and route (PNR) of other logic blocks. Each synapse core and the central STDP controller have 1.394 and 0.025 mm^2 areas. The throughput and power of the design are estimated from the layout and after parasitic extraction.

4.3. Accuracy Analysis

The ConvSNN shown in **Table 1** is simulated using ParallelSpikeSim, an open source GPU accelerated SNN simulator (She et al., 2019). The MNIST and CIFAR-10 datasets are used for accuracy evaluation. All synapses are designed with 8-bit weights. The unsupervised-learning based CONV layers are trained with STDP for unsupervised clustering of inputs. The supervised-learning based CONV layers are trained with the BPTT algorithm. The final FC layer is trained using Stochastic Gradient Descent (SGD) to label the clusters with appropriate classes. Input spike frequency is converted from image pixels intensity to the range of 10–50 Hz. We assumed $\frac{T_{\text{total}}}{T_{\text{unit}}} = 100$, i.e., each image is shown to the network for 100 time steps. Each layer learns the entire training set for 5 epochs.

Table 2 shows the accuracies of each ConvSNN configuration. **Type 2** is based on the data flow of MONETA (**Figures 8B**, **11**). When we compare the accuracies with CIFAR-10, the accuracy of **Type 2** shows only 1.63 (%) lower accuracy than a fully parallel (as shown in **Figure 8A**) software implementation of the network using 8-bit precision (**Type 1**). As mentioned before, the fully parallel implementation incurs $\frac{T_{\text{total}}}{T_{\text{unit}}}\times$ (=100×) higher data movement than our design. The

TABLE 1 | Architecture and parameters of the tested convolutional neural network within spiking neural network (ConvSNN) networks.

Layers	Homogeneous network architecture			Hybrid network architecture					
				BP ConvSNN block			STDP ConvSNN block		
	F_R	F_C	F_D	F_R	F_C	F_D	F_R	F_C	F_D
CONV1	3	3	64	3	3	64	3	3	64
CONV2	3	3	64	3	3	64	3	3	64
CONV3	3	3	128	3	3	128	3	3	128
CONV4	3	3	128	3	3	128	3	3	128
FC	F_R	F_C	F_D	F_R		F_C		F_D	
	1	1	512	1		1		1,024	

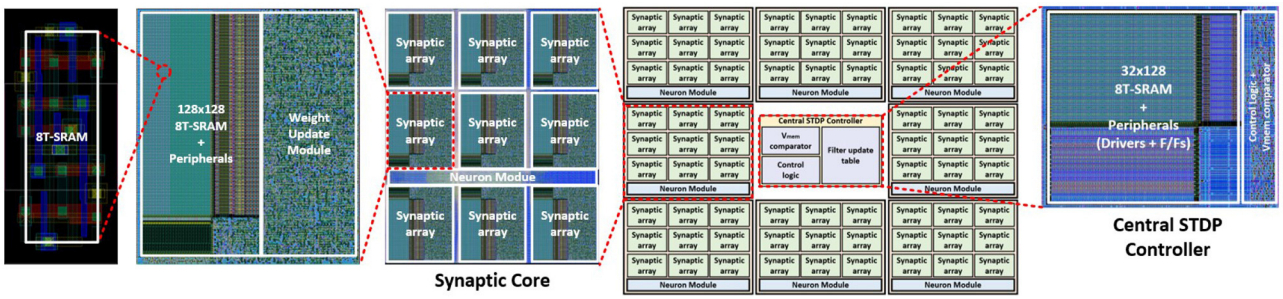


FIGURE 15 | The overview of the physical design of MONETA.

TABLE 2 | Simulated network types and the results.

Type	Learning algorithm	Required parameters (Kb)	Inference throughput (TOPS)	Inference Energy efficiency (TOPS/W)	On-line learning throughput (TOPS)	Learning energy efficiency (TOPS/W)	Accuracy (CIFAR-100)	Accuracy (CIFAR-10)	Accuracy (MNIST)
1	Standard STDP	1,152	2.304	18.69	N/A	N/A	54.25	67.88	90.89
2	PIM-friendly STDP				2.2	7.25	52.19	66.25	90.13
3	Backpropagation (BP)	2,304	4.608		N/A	N/A	62.12	76.43	92.55
4	BP + Standard STDP				N/A	N/A	63.86	78.94	93.16
5	BP + PIM-friendly STDP				4.4	10.41	62.31	77.83	92.07

accuracy of the ConvSNN accelerated using MONETA (**Type 2**) is 10.18% lower than a spiking neural network of the same layer configurations trained using backpropagation (**Type 3**). In the end, the accuracy of the hybrid network shows improved accuracy than supervised learning. The result of the hybrid network using backpropagated-based ConvSNN and the PIM-friendly STDP learning (**Type 5**) shows 1.4% higher accuracy than the fully backpropagated ConvSNN model (**Type 3**). This accuracy from **Type 5** is only 1.11% lower than the hybrid network applying the standard STDP model (**Type 4**).

We note that the accuracy of backpropagation-based trained SNN demonstrated in this article is lower than the state-of-the-art, for example, 99.59% accuracy was observed on MNIST dataset (Lee et al., 2020). This is primarily because, we have reduced the simulation time necessary for training the network with back propagation. For example, instead of 100 epochs of training as performed in the Lee et al. (2020) we only trained the network for 20 epochs. Further, we did not apply pre-processing and the fine- tuning methodologies that are normally applied in BP SOTA, as these techniques are applied off-chip and are not related to the PIM-based hardware implementation of ConvSNN.

4.4. Throughput Analysis

Table 2 shows the peak throughput of MONETA estimated as Tera Operation per Second (TOPS). The throughput for each synaptic array is determined by the number of parallel multiplies (# of weights stored in a row) and accumulate. The total throughput of all synaptic arrays is given by # of weights × # of synaptic arrays × frequency. H-NoC sums partial outputs from synaptic arrays resulting in a throughput of # of weights × # of synaptic array × frequency. The neuron modules compute membrane potential

neurons at a throughput of # of weights × frequency. The total throughput is obtained considering the parallel operation of all synaptic cores. Our design has # of weights per word line = 16, # of synaptic arrays = 9, # of synaptic cores = 8, and frequency = 1GHz. Hence, the total throughput of one MONETA chip is 2.304 TOPS in the inference mode. On the other hand, in the case of the on-line learning mode, the throughput is reduced based on the time used for the training. Because the weight update takes 17 cycles, the throughput becomes $2.304 \times \frac{1}{1+17\times(\text{output spike rate})}$. For example, learning mode throughput in **CONV4** layer is 2.2 TOPS with an output spike rate of 0.0028.

In MONETA, each time-step is represented as 1 clock cycle (1 GHz), and 100 time-steps are used for each image. The total clock cycles required to operate on one image in each layer is $100 \times \frac{I_R}{S} \times \frac{I_C}{S} \times I_D$ (S is a stride), where I_D represents a number of rows in the synaptic array. We compute the image processing rate (fps) of **CONV1-4** is 13.02, 2.44, 9.77, and 19.53 K, respectively, at 1 GHz.

4.5. Area and Power Analysis

The power of the MONETA design is 123.28 mW, 303.52 mW for the inference mode and the learning mode, respectively, at 1 GHz with 1 V supply on the one chip. This power is calculated based on **CONV4** which is the maximum power of MONETA. In addition, the power is computed considering **CONV4**'s input and output spike activity ratio (0.0092 and 0.0028). Note, **CONV4** of the hybrid network requires two MONETA chips because of its parameters, so the total power is two times the homogeneous networks (246.56 mW and 607.04 mW for the inference mode and the learning mode, respectively).

Figure 16A shows the power breakdown of the synaptic core's inference mode. The weight update module is idle during the inference mode by clock gating. The 8T-SRAM array consumes the 21.86 pJ for matrix multiplication calculation, 4.48 pJ for transposable weight read, and 12.34 pJ for transposable weight write. The SRAM-based computation naturally transforms sparsity in neuron firing (i.e., zero values in IFM) to power saving during inference. If an input spike is absent in a cycle, the SRAM power for that cycle is zero as word lines are not activated.

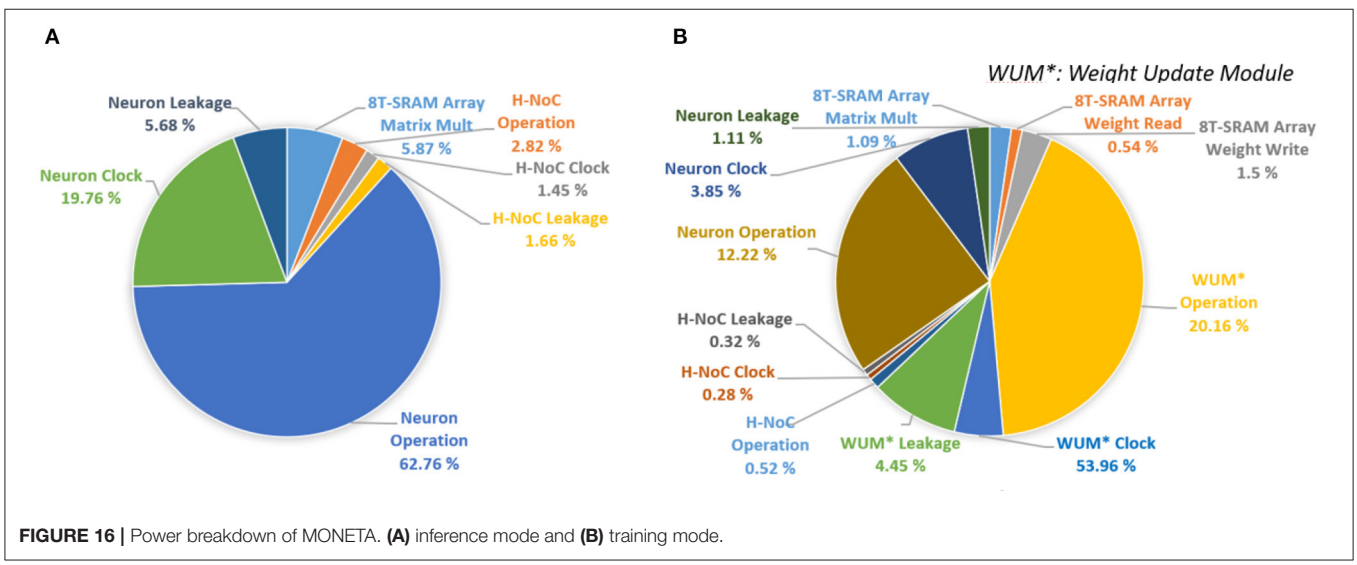


FIGURE 16 | Power breakdown of MONETA. **(A)** inference mode and **(B)** training mode.

TABLE 3 | Comparison with other works.

Reference	This work				DAC'20	ISSCC'19	JSSC'19	VLSI'17
	Homogeneous		Hybrid					
	Inference	Learning	Inference	Learning				
Technology (nm)	65				90	65	10	40
Algorithm	ConvSNN				ConvSNN	SNN	SNN	SNN
On-chip Training	Yes				No	Yes	Yes	Yes
Voltage (V)	1.0				1.0	0.8	0.9	0.9
Frequency (MHz)	1,000				100	20	506	250
Synapse Bits	8				1	7	7	4
Area (mm^2)	11.155		22.23		2.07	10.08	1.72	1.31
TOPS/mm^2	0.207	0.197	0.207	0.197	0.312	0.008	0.015	0.227
Power (mW)	123.28	303.52	246.56	423.83	45.71	23.6	208.3	87
TOPS/W	18.69	7.25	18.69	10.41	14.1	3.42	0.12	3.43

Moreover, as we use single ended sensing in 8T-SRAM, there is no bit-line discharge when the values of the corresponding bit are "0." Hence, the SRAM contributes very little power to the overall operation. The power is dominated by the V_{mem} calculation in the neuron module. This is because there exists an inherent leakage component in the membrane potential computation ($a + bV_{\text{mem}}$ in LIF dynamics in **Figure 3**) that causes the membrane potential to reduce when there are no input spikes. Hence, the neuron module needs to perform the leakage computation in each clock. However, the power in the synaptic array and the H-NoC reduces significantly due to low spiking activity (=0.0092). **Figure 16B** shows the power distribution of the synaptic cores in the training mode. It shows the much higher power consumption compared to the inference mode, mainly because of the complex weight update module (Kim et al., 2020).

The central STDP controller consists of 32 × 128 SRAM, the control logic, and the V_{mem} Comparator. The read energy is 1.14 pJ, and the write energy is 3.09 pJ. The bus-width is 32 bits. It also operates at 1 GHz by following the Synapse Core's clock frequency. Overall, the central STDP controller has a much smaller area (0.025 mm^2) and power (0.141 mW during inference and 0.154 mW during learning).

4.6. Comparison With Prior Works

Table 3 shows the comparison of MONETA with a set of recent SNN accelerators (Buhler et al., 2017; Chen et al., 2019; Park et al., 2019; Chuang et al., 2020). Note that all designs use different SNN architectures for evaluation, and most of the prior designs considered MNIST as the dataset while our work is evaluated on MNIST CIFAR-10 and CIFAR-100. Our design supports STDP learning (fully for the homogeneous network and partially for the hybrid network) and inference.

Our throughput is higher than prior works mainly due to highly parallel in-memory computation, as well as higher frequency (1 GHz) of operation. The PIM architecture eliminates the arithmetic computation units used in prior designs leading to a much higher operating speed at similar voltage. Thanks to the PIM architecture, MONETA shows higher compute density (TOPS/mm^2) compared to the prior works using similar bit-precision. We observe similar area efficiency compared to 4-bit precision-based SNN in 40 nm CMOS, even though our design is realized in 65 nm CMOS. However, compared to the binary SNN design we observe 33% lower area efficiency (note, the binary SNN was implemented in 90 nm CMOS). Further, we observe a higher power efficiency compared to other designs during inference and learning. This is mainly because the PIM-based operation naturally translates the sparsity in neuron firing to power reduction as discussed before.

5. CONCLUSION

This article presents a PIM-based hybrid ConvSNN acceleration platform with an on-chip STDP based weight update. We present an optimized data flow for sequential processing of input feature maps to reduce off-chip data movement while ensuring learning accuracy of the STDP process. The algorithmic simulations show comparable accuracy for MNIST and CIFAR-10 dataset to a pure software implementation. We also show the hybrid architecture and the opportunity of the supervised-unsupervised flexible weight architecture with on-line learning. The power and throughput analysis using 65 nm CMOS physical design show high throughput and energy efficiency. The programming model and compiler infrastructure necessary to map an arbitrary ConvSNN in MONETA is important future work.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found at: MNIST: http://yann.lecun.com/exdb/mnist/; https://www.cs.toronto.edu/~kriz/cifar.html.

AUTHOR CONTRIBUTIONS

DK developed the main concepts and algorithm, generated the RTL design and layout, and performed all hardware analysis. BC and XS developed the algorithm for a hybrid network and performed the software simulation for accuracy analysis. All authors assisted in developing the concept and writing this article. All authors contributed to the article and approved the submitted version.

FUNDING

This research was supported by the DARPA ERI 3DSoC Program under Award HR001118C0096.

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.

Buhler, F. N., Brown, P., Li, J., Chen, T., Zhang, Z., and Flynn, M. P. (2017). "A 3.43tops/w 48.9pj/pixel 50.1nj/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm cmos," in *2017 Symposium on VLSI Circuits* (Kyoto), C30–C31.

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chakraborty, B., She, X., and Mukhopadhyay, S. (2021). A fully spiking hybrid neural network for energy-efficient object detection. *IEEE Trans. Image Process.* 30, 9014–9029. doi: 10.1109/TIP.2021.3122092

Chen, G. K., Kumar, R., Sumbul, H. E., Knag, P. C., and Krishnamurthy, R. K. (2019). A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos. *IEEE J. Solid-State Circ.* 54, 992–1002. doi: 10.1109/JSSC.2018.2884901

Chen, Y., Krishna, T., Emer, J., and Sze, V. (2017). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circ.* 52, 127–138. doi: 10.1109/JSSC.2016.2616357

Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., et al. (2016). "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture ISCA '16* (Seoul: IEEE Press), 27–39.

Chuang, P. Y., Tan, P.-Y., Wu, C.-W., and Lu, J.-M. (2020). "A 90nm 103.14 tops/w binary-weight spiking neural network cmos asic for real-time object classification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)* (San Francisco, CA), 1–6.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255.

Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020). Tianjic: aunified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid-State Circ.* 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9, 99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8.

Gerstner, W., and Kistler, W. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Camebridge: Cambridge University Press.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778.

Imani, M., Gupta, S., Kim, Y., and Rosing, T. (2019). "Floatpim: in-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (Phoenix, AZ), 802–815.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kim, D., She, X., Rahman, N. M., Chekuri, V. C. K., and Mukhopadhyay, S. (2020). Processing-in-memory-based on-chip learning with spike-time-dependent plasticity in 65-nm cmos. *IEEE Solid-State Circ. Lett.* 3, 278–281. doi: 10.1109/LSSC.2020.3013448

Kim, S. Park, S., Na, B., and Yoon, S. (2020). "Spiking-yolo: spiking neural network for energy-efficient object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence* (New York, NY), vol. 34, 11270–11277.

Ledinauskas, E., Ruseckas, J., Juršėnas, A., and Buračas, G. (2020). Training deep spiking neural networks. *arXiv preprint* arXiv:2006.04436. doi: 10.48550/ARXIV.2006.04436

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12, 435. doi: 10.3389/fnins.2018.00435

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front Neurosci.* 14, 119. doi: 10.3389/fnins.2020.00119

Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2019). Deep spiking convolutional neural network trained with unsupervised spike-timing-dependent plasticity. *IEEE Trans. Cogn. Develop. Syst.* 11, 384–394. doi: 10.1109/TCDS.2018.2833071

Long, Y. et al. (2019). A ferroelectric fet-based processing-in-memory architecture for dnn acceleration. *IEEE J. Exp. Solid-State Comput. Dev. Circ.* 5, 113–122. doi: 10.1109/JXCDC.2019.2923745

Long, Y., Lee, E., Kim, D., and Mukhopadhyay, S. (2020). "Q-pim: a genetic algorithm based flexible dnn quantization method and application to processing-in-memory platform," in *2020 57th ACM/IEEE Design Automation Conference (DAC)* (San Francisco, CA), 1–6.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671.

Miquel, J. R., Tolu, S., Schöller, F. E., and Galeazzi, R. (2021). Retinanet object detector based on analog-to-spiking neural network conversion. *arXiv preprint* arXiv:2106.05624. doi: 10.48550/ARXIV.2106.05624

Narayanan, S., Taht, K., Balasubramonian, R., Giacomin, E., and Gaillardon, PE. (2020). "Spinalflow: an architecture and dataflow tailored for spiking neural networks," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (Valencia), 349–362.

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Panda, P., Aketi, S. A., and Roy, K. (2020). Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Front. Neurosci.* 14, 653. doi: 10.3389/fnins.2020.00653

Park, J., Lee, J., and Jeon, D. (2019). "7.6 a 65nm 236.5nj/classification neuromorphic processor with 7.5energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)* (San Francisco, CA), 140-142.

Peng, X., Huang, S., Jiang, H., Lu, A., and Yu, S. (2021). DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Trans. Comput. Aid. D. Integ. Circui.t Syst.* 40, 2306–2319. doi: 10.1109/TCAD.2020.3043731

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12, 774. doi: 10.3389/fnins.2018.00774

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: vgg and residual architectures. *Front. Neurosci.* 13, 95. doi: 10.3389/fnins.2019.00095

Seo, J., Brezzo, B., Liu, Y., Parker, B. D., Esser, S. K., Montoye, R. K., et al. (2011). "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *2011 IEEE Custom Integrated Circuits Conference (CICC)* (San Jose, CA), 1–4.

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Paul Strachan, J., Hu, M., et al. (2016). "Isaac: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of the 43rd International Symposium on Computer Architecture ISCA '16* (Seoul: IEEE Press), 14–26.

She, X., Long, Y., Kim, D., and Mukhopadhyay, S. (2021). Scienet: deep learning with spike-assisted contextual information extraction. *Pattern Recogn.* 118, 108002. doi: 10.1016/j.patcog.2021.108002

She, X., Long, Y., and Mukhopadhyay, S. (2019). "Fast and low-precision learning in gpu-accelerated spiking neural network," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Florence: IEEE), 450–455.

She, X., Saha, P., Kim, D., Long, Y., and Mukhopadhyay, S. (2020). "Safe-dnn: a deep neural network with spike assisted feature extraction for noise robust inference," in *2020 International Joint Conference on Neural Networks (IJCNN)* (Glasgow: IEEE), 1–8.

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint* arXiv:1409.1556. doi: 10.48550/ARXIV.1409.1556

Singh, S., Sarma, A., Jao, N., Pattnaik, A., Lu, S., Yang, K., et al. (2020). "Nebula: a neuromorphic spin-based ultra-low power architecture for snns and anns," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (Valencia), 363–376.

Srinivasan, G., Panda, P., and Roy, K. (2018). Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* 14, 1–12. doi: 10.1145/3266229

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2020). "Efficient processing of deep neural networks," in *Synthesis Lectures on Computer Architecture* San Rafael, CA: Morgan and Claypool, vol. 15. 1–341.

Tavanaei, A. and Maida, A. S. (2016). Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning. *arXiv [Preprint]*. arXiv: 1611.03000. Available online at: https://arxiv.org/pdf/1611.03000.pdf

Wang, G., Ma, S., Wu, Y., Pei, J., Zhao, R., and Shi, L. (2021). End-to-end implementation of various hybrid neural networks on a cross-paradigm

neuromorphic chip. *Front. Neurosci.* 15, 45. doi: 10.3389/fnins.2021.615279

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Author Disclaimer: The views and conclusions included in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. Government.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers | Frontiers in Artificial Intelligence

ORIGINAL RESEARCH
published: 09 May 2022
doi: 10.3389/frai.2022.891624

Neural Network Training With Asymmetric Crosspoint Elements

Murat Onen[1,2*‡], *Tayfun Gokmen*[1*‡], *Teodor K. Todorov*[1], *Tomasz Nowicki*[1], *Jesús A. del Alamo*[2], *John Rozen*[1], *Wilfried Haensch*[1] *and Seyoung Kim*[1*†]

[1] IBM Thomas J. Watson Research Center, Yorktown Heights, NY, United States, [2] Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, United States

OPEN ACCESS

Edited by:
Saban Öztürk,
Amasya University, Turkey

Reviewed by:
Mucahid Barstugan,
Konya Technical University, Turkey
Umut Özkaya,
Konya Technical University, Turkey
Bing Li,
Capital Normal University, China

***Correspondence:**
Murat Onen
monen@mit.edu
Tayfun Gokmen
tgokmen@us.ibm.com
Seyoung Kim
kimseyoung@postech.ac.kr

†Present address:
Seyoung Kim,
Department of Materials Science and Engineering, POSTECH, Pohang, South Korea

‡These authors have contributed equally to this work

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: 08 March 2022
Accepted: 01 April 2022
Published: 09 May 2022

Citation:
Onen M, Gokmen T, Todorov TK, Nowicki T, del Alamo JA, Rozen J, Haensch W and Kim S (2022) Neural Network Training With Asymmetric Crosspoint Elements. Front. Artif. Intell. 5:891624. doi: 10.3389/frai.2022.891624

Analog crossbar arrays comprising programmable non-volatile resistors are under intense investigation for acceleration of deep neural network training. However, the ubiquitous asymmetric conductance modulation of practical resistive devices critically degrades the classification performance of networks trained with conventional algorithms. Here we first describe the fundamental reasons behind this incompatibility. Then, we explain the theoretical underpinnings of a novel fully-parallel training algorithm that is compatible with asymmetric crosspoint elements. By establishing a powerful analogy with classical mechanics, we explain how device asymmetry can be exploited as a useful feature for analog deep learning processors. Instead of conventionally tuning weights in the direction of the error function gradient, network parameters can be programmed to successfully minimize the total energy (Hamiltonian) of the system that incorporates the effects of device asymmetry. Our technique enables immediate realization of analog deep learning accelerators based on readily available device technologies.

Keywords: analog computing, DNN training, hardware accelerator architecture, neuromorphic accelerator, learning algorithm

INTRODUCTION

Deep learning has caused a paradigm shift in domains such as object recognition, natural language processing, and bioinformatics which benefit from classifying and clustering representations of data at multiple levels of abstraction (Lecun et al., 2015). However, the computational workloads to train state-of-the-art deep neural networks (DNNs) demand enormous computation time and energy costs for data centers (Strubell et al., 2020). Since larger neural networks trained with bigger data sets generally provide better performance, this trend is expected to accelerate in the future. As a result, the necessity to provide fast and energy-efficient solutions for deep learning has invoked a massive collective research effort by industry and academia (Chen et al., 2016; Jouppi et al., 2017; Rajbhandari et al., 2020).

Highly optimized digital application-specific integrated circuit (ASIC) implementations have attempted to accelerate DNN workloads using reduced-precision arithmetic for the computationally intensive matrix operations. Although acceleration of inference tasks was achieved by using 2-bit resolution (Choi et al., 2019), learning tasks were found to require at least hybrid 8-bit floating-point formats (Sun et al., 2019) which still imposes considerable energy consumption and processing time for large networks. Therefore, beyond-digital approaches that can efficiently handle training workloads are actively sought for.

The concept of in-memory computation with analog resistive crossbar arrays is under intense study as a promising alternative. These frameworks were first designed to make use of Ohm's and Kirchhoff's Laws to perform parallel vector–matrix multiplications (see **Supplementary Sections 2.1**, **2.2** for details), which constitute $\approx$ 2/3 of the overall computational load (Steinbuch, 1961). However, unless the remaining $\approx$ 1/3 of computations during the update cycle is parallelized as well, the acceleration factors provided by analog arrays will be a mere 3× at best with respect to conventional digital processors. It was much later discovered that rank-one outer products can also be achieved in parallel, using pulse-coincidence and incremental changes in device conductance (Burr et al., 2015; Gokmen and Vlasov, 2016). Using this method, an entire crossbar array can be updated in parallel, without explicitly computing the outer product[1] or having to read the value of any individual crosspoint element (Gokmen et al., 2017). As a result, all basic primitives for DNN training using the Stochastic Gradient Descent (SGD) algorithm can be performed in a fully-parallel fashion using analog crossbar architectures. However, this parallel update method imposes stringent device requirements since its performance is critically affected by the conductance modulation characteristics of the crosspoint elements. In particular, asymmetric conductance modulation characteristics (i.e., having mismatch between positive and negative conductance adjustments) are found to deteriorate classification accuracy by causing inaccurate gradient accumulation (Yu et al., 2015; Agarwal et al., 2016, 2017; Gokmen and Vlasov, 2016; Gokmen et al., 2017, 2018; Ambrogio et al., 2018). Unfortunately, all analog resistive devices to date have asymmetric characteristics, which poses a major technical barrier before the realization of analog deep learning processors.

In addition to widespread efforts to engineer ideal resistive devices (Woo and Yu, 2018; Fuller et al., 2019; Grollier et al., 2020; Yao et al., 2020), many high-level mitigation techniques have been proposed to remedy device asymmetry. Despite numerous published simulated and experimental demonstrations, none of these studies so far provides a solution for which the analog processor still achieves its original purpose: energy-efficient acceleration of deep learning. The critical issue with the existing techniques is the requirement of serial accessing to crosspoint elements one-by-one or row-by-row (Prezioso et al., 2015; Yu et al., 2015; Agarwal et al., 2017; Burr et al., 2017; Ambrogio et al., 2018; Li et al., 2018, 2019; Cai et al., 2019; Sebastian et al., 2020). Methods involving serial operations include reading conductance values individually, engineering update pulses to artificially force symmetric modulation, and carrying or resetting weights periodically. Furthermore, some approaches offload the gradient computation to digital processors, which not only requires consequent serial programming of the analog matrix, but also bears the cost of outer product calculation (Prezioso et al., 2015; Yu et al., 2015; Li et al., 2018, 2019; Cai et al., 2019; Sebastian et al., 2020). Updating an $N \times N$ crossbar array with these serial routines would require at least N or even N^2 operations. For practical array sizes, the update cycle would simply take too much computational time and energy. In conclusion, for implementations that compromise parallelism, whether or not the asymmetry issue is resolved becomes beside the point since computational throughput and energy efficiency benefits over conventional digital processors are lost for practical applications. It is therefore urgent to devise a method that deals with device asymmetry while employing only fully-parallel operations.

Recently, our group proposed a novel fully-parallel training method, *Tiki-Taka,* that can successfully train DNNs based on asymmetric resistive devices with asymmetric modulation characteristics (Gokmen and Haensch, 2020). This algorithm was empirically shown in simulation to deliver ideal-device-equivalent classification accuracy for a variety of network types and sizes emulated with asymmetric device models (Gokmen and Haensch, 2020). However, the missing theoretical underpinnings of the proposed algorithmic solution as well as the cost of doubling analog hardware previously limited the method described in Gokmen and Haensch (2020).

In this paper, we first theoretically explain why device asymmetry has been a fundamental problem for SGD-based training. By establishing a powerful analogy with classical mechanics., we further establish that the *Tiki-Taka* algorithm minimizes the total energy (Hamiltonian) of the system, incorporating the effects of device asymmetry. The present work formalizes this new method as Stochastic Hamiltonian Descent (SHD) and describes how device asymmetry can be exploited as a useful feature in a fully-parallel training. The advanced physical intuition allows us to enhance the original algorithm and achieve a reduction in hardware cost of 50%, improving its practical relevance. Using simulated training results for different device families, we conclude that SHD provides better classification accuracy and faster convergence with respect to SGD-based training in all applicable scenarios. The contents of this paper provide a guideline for the next generation of crosspoint elements as well as specialized algorithms for analog computing.

THEORY

Neural networks can be construed as many layers of matrices (i.e., weights, W) performing affine transformations followed by non-linear activation functions. Training (i.e., learning) process refers to the adjustment of W such that the network response to a given input produces the target output for a labeled dataset. The discrepancy between the network and target outputs is represented with a scalar error function, E, which the training algorithm seeks to minimize. In the case of the conventional SGD algorithm (Cauchy, 1847), values of W are incrementally modified by taking small steps (scaled by the learning rate, η) in the direction of the gradient of the error function sampled for each input. Computation of the gradients is performed by the backpropagation algorithm consisting of forward pass, backward pass, and update subroutines (Rumelhart et al., 1986) (**Figure 1A**). When the discrete nature of DNN training is analyzed in the continuum limit, the time evolution of W can

[1]The result of the outer product is not returned to the user, but implicitly applied to the network.

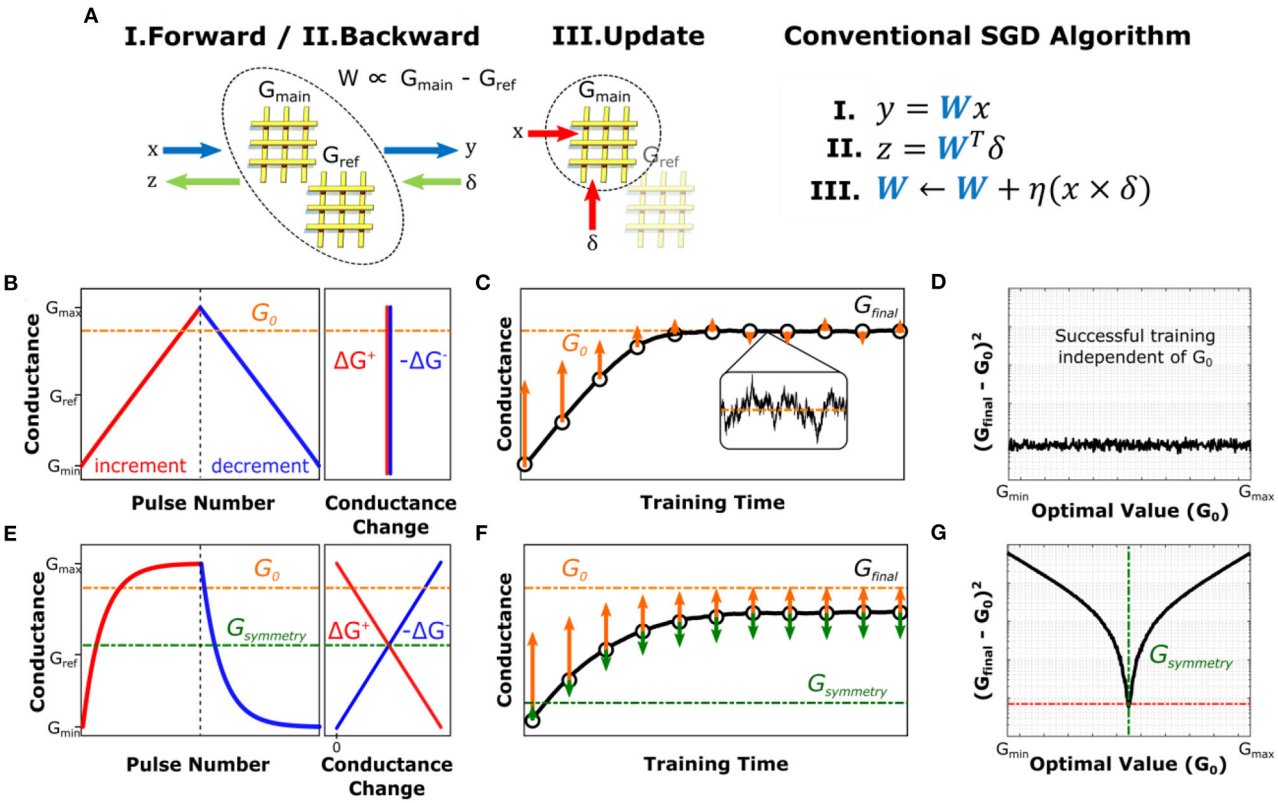


FIGURE 1 | Effect of asymmetric conductance modulation for SGD-based training. **(A)** Schematic and pseudocode of processes for conventional SGD algorithm (Cauchy, 1847). Vectors $x, y,$ represent the input and output vectors in the forward pass whereas δ, z contain the backpropagated error information. The analog architecture schematic is only shown for a single layer, where all vectors are propagated between upper and lower network layers in general. The pseudocode only describes operations computed in the analog domain, whereas digital computations such as activation functions are not shown for simplicity. **(B)** Sketch of conductance modulation behavior of a symmetric crosspoint device. **(C)** Simulated single-parameter optimization result for the symmetric device shown in **(B)**. conductance successfully converges to the optimal value for the problem at hand, G_0. **(D)** Simulated residual distance between the final converged value, G_{final}, and G_0 for training the device with characteristics shown in **(B)** for datasets with different optimal values. **(E)** Sketch of conductance modulation behavior of an asymmetric crosspoint device. The point at which $\Delta G^+ = \Delta G^-$ is defined as the symmetry point of the device ($G_{symmetry}$) **(F)** Simulated training result for the same single-parameter optimization with the asymmetric device shown in **(E)**. Device conductance fails to converge to G_0, but instead settles at a level between G_0 and $G_{symmetry}$. **(G)** Simulated residual distance (in semilog scale) between the final value, G_{final}, and G_0 for training the device with characteristics shown in **(E)** for datasets with different optimal values.

be written as a Langevin equation:

$$\dot{W} = -\eta \left[\frac{\partial E}{\partial W} + \epsilon(t) \right] \tag{1}$$

where η is the learning rate and $\epsilon(t)$ is a fluctuating term with zero-mean, accounting for the inherent stochasticity of the training procedure (Feng and Tu, 2023). As a result of this training process, W converges to the vicinity of an optimum W_0, at which $\frac{\partial E}{\partial W} = 0$ but $\dot{W}$ is only on average 0 due to the presence of $\epsilon(t)$. For visualization, if the training dataset is a cluster of points in space, W_0 is the center of that cluster, where each individual point still exerts a force ($\epsilon(t)$) that averages out to 0 over the whole dataset.

In the case of analog crossbar-based architectures, the linear matrix operations are performed on arrays of physical devices, whereas all non-linear computations (e.g., activation and error functions) are handled at peripheral circuitry. The strictly positive nature of device conductance requires representation of each weight by means of the differential conductance of a pair of crosspoint elements (i.e., $W \propto G_{main} - G_{ref}$). Consequently, vector-matrix multiplications for the forward and backward passes are computed by using both the main and the reference arrays (**Figure 1A**). On the other hand, the gradient accumulation and updates are only performed on the main array using bidirectional conductance changes while the values of the reference array are kept constant[2]. In this section, to illustrate the basic dynamics of DNN training with analog architectures, we study a single-parameter optimization problem (linear regression) which can be considered as the simplest "neural network".

[2]For implementations using devices showing unidirectional conductance modulation characteristics, both the main and the reference array are updated. When SGD is used as the training algorithm, values of G_{ref} are not critical as long as they fall in the midrange of G_{main}'s conductance span (Gokmen and Vlasov, 2016).

The weight updates in analog implementations are carried out through modulation of the conductance values of the crosspoint elements, which are often applied by means of pulses. These pulses cause incremental changes in device conductance ($\Delta G^{+,-}$). In an ideal device, these modulation increments are of equal magnitude in both directions and independent of the device conductance, as shown in **Figure 1B**. It should be noted that the series of modulations in the training process is inherently non-monotonic as different input samples in the training set create gradients with different magnitudes and signs in general. Furthermore, as stated above, even when an optimum conductance, G_0, is reached ($W_0 \propto G_0 - G_{ref}$), continuing the training operation would continue modifying the conductance in the vicinity of G_0, as shown in **Figure 1C**. Consequently, G_0 can be considered as a dynamic equilibrium point of the device conductance from the training algorithm point of view.

Despite considerable technological efforts in the last decade, analog resistive devices with the ideal characteristics illustrated in **Figure 1B** have yet to be realized. Instead, practical analog resistive devices display asymmetric conductance modulation characteristics such that unitary (i.e., single-pulse) modulations in opposite directions do not cancel each other in general, i.e., $\Delta G^{+}(G) \neq -\Delta G^{-}(G)$. However, with the exception of some device technologies such as Phase Change Memory (PCM) which reset abruptly (Burr et al., 2017; Sebastian et al., 2017; Ambrogio et al., 2018), many crosspoint elements can be modeled by a smooth, monotonic, non-linear function that shows saturating behavior at its extrema as shown in **Figure 1E** (Kim et al., 2019b, 2020; Yao et al., 2020). For such devices, there exists a unique conductance point, $G_{symmetry}$, at which the magnitude of an incremental conductance change is equal to that of a decremental one. As a result, the time evolution of G during training can be rewritten as:

$$\dot{G} = -\eta \left[\frac{\partial E}{\partial G} + \epsilon\,(t)\right] - \eta\kappa \left|\frac{\partial E}{\partial G} + \epsilon\,(t)\right| . f_{hardware} \qquad (2)$$

where κ is the asymmetry factor and $f_{hardware}$ is the functional form of the device asymmetry (see **Supplementary Section 1.1** for derivation). In this expression, the term $-\eta \left|\frac{\partial E}{\partial G} + \epsilon(t)\right|$ signifies that the direction of the change related to asymmetric behavior is solely determined by $f_{hardware}$, irrespective of the direction of the intended modulation. For the exponentially saturating device model shown in **Figure 1E**, $f_{hardware} = G - G_{symmetry}$, which indicates that each and every update event has a component that drifts the device conductance toward its symmetry point. A simple observation of this effect is when enough equal number of incremental and decremental changes are applied to these devices in a random order, the conductance value converges to the vicinity of $G_{symmetry}$ (Kim et al., 2020). Therefore, this point can be viewed as the physical equilibrium point for the device, as it is the only conductance value that is dynamically stable.

It is essential to realize that there is in general no relation between $G_{symmetry}$ and G_0, as the former is entirely device-dependent while the latter is problem-dependent. As a result, for an asymmetric device, two equilibria of hardware and software create a competing system, such that the conductance value converges to a particular conductance somewhere between $G_{symmetry}$ and G_0, for which the driving forces of the training algorithm and device asymmetry are balanced out (**Figure 1F**). In examples shown in **Figures 1C,F**, G_0 of the problem is purposefully designed to be far away from $G_{symmetry}$, so as to depict a case for which the effect of asymmetry is pronounced. Indeed, it can be seen that the discrepancy between the final converged value, G_{final}, and G_0 strongly depends on the relative position of G_0 with respect to the $G_{symmetry}$ (**Figure 1G**), unlike that of ideal devices (**Figure 1D**). Detailed derivation of these dynamics can be found in **Supplementary Section 1.2**.

In contrast to SGD, our new training algorithm, illustrated in **Figure 2A**, separates both the forward path and error backpropagation from the update function. For this purpose, two array pairs (instead of a single pair), namely A_{main}, A_{ref}, C_{main}, C_{ref} are utilized to represent each layer (Gokmen and Haensch, 2020). In this representation, $A = A_{main} - A_{ref}$ stands for the auxiliary array and $C = C_{main} - C_{ref}$ stands for the core array.

The new training algorithm operates as follows. At the beginning of the training process, A_{ref} and C_{ref} are initialized to $A_{main,symmetry}$ and $C_{main,symmetry}$, respectively [reasons will be clarified later, see Section M1. Array Initialization (Zero-Shifting) for details] following the method described in Kim et al. (2020). As illustrated in **Figure 2A**, first, forward and backward pass cycles are performed on the array-pair C (Steps *I* and *II*), and corresponding updates are performed on A_{main} (scaled by the learning rate η_A) using the parallel update scheme discussed in Gokmen and Vlasov (2016) (Step *III*). In other words, the updates that would have been applied to C in a conventional SGD scheme are directed to A instead.

Then, every τ cycles, another forward pass is performed on A, with a vector u, which produces $v = Au$ (Step *IV*). In its simplest form, u can be a vector of all "0"s but one "1", which then makes v equal to the row of A corresponding to the location of "1" in u. Finally, the vectors u and v are used to update C_{main} with the same parallel update scheme (scaled by the learning rate η_c) (Step *V*). These steps (*IV* and *V* shown in **Figure 2A**) essentially partially add the information stored in A to C_{main}. The complete pseudocode for the algorithm can be found in Section M2. Pseudocode for SHD Algorithm.

At the end of the training procedure C alone contains the optimized network, to be later used in inference operations (hence the name core). Since A receives updates computed over $\frac{\partial E}{\partial C}$, which have zero-mean once C is optimized, its active component, A_{main}, will be driven toward $A_{main,symmetry}$. The choice to initialize the stationary reference array, A_{ref}, at $A_{main,symmetry}$ ensures that $A = 0$ at this point (i.e., when C is optimized), thus generating no updates to C in return.

With the choice of u vectors made above, every time steps *IV* and *V* are performed, the location of the "1" for the u vector would change in a cyclic fashion, whereas in general any set of orthogonal u vectors can be used for this purpose (Gokmen and Haensch, 2020). We emphasize that these steps should not be confused with weight carrying (Agarwal et al., 2017; Ambrogio et al., 2018), as C is updated by only a fractional amount in the direction of A as $\eta_C << 1$ and at no point information stored

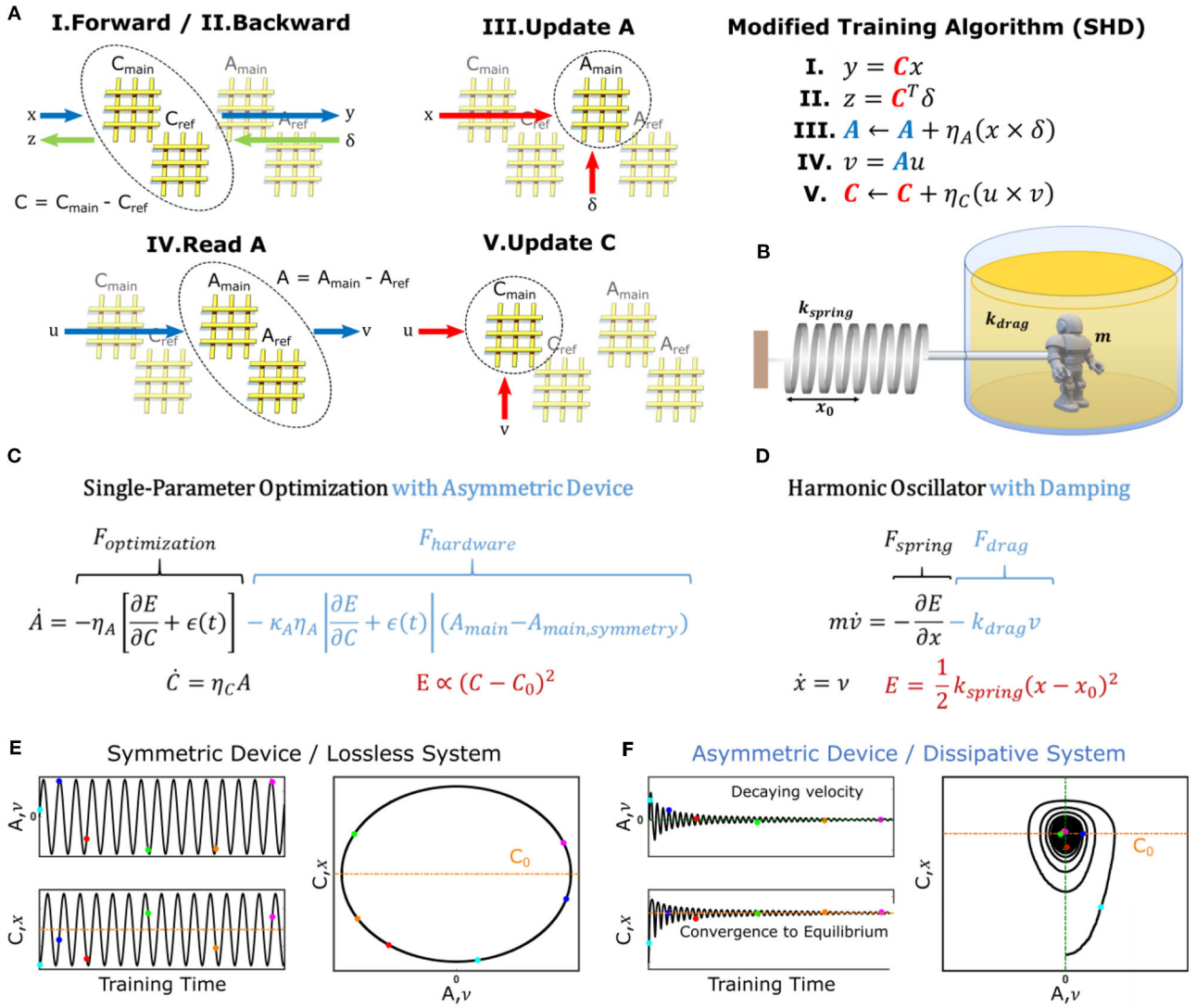


FIGURE 2 | DNN training with Stochastic Hamiltonian Descent (SHD) algorithm and dynamics of a dissipative harmonic oscillator. **(A)** Schematic and pseudocode of training process using the SHD algorithm. The pseudocode only describes operations computed in the analog domain, whereas digital computations such as non-linear error functions are not shown for simplicity. **(B)** Illustration of a damped harmonic oscillator system. **(C)** Differential equations describing the evolution of the parameters with the SHD training algorithm in the continuum limit. **(D)** Equations of motion describing the dynamics of a harmonic oscillator. **(E)** Simulated results for a single-parameter optimization task using the SHD algorithm with symmetric devices described in **Figure 1B**. **(F)** Simulated results for a single-parameter optimization task using the SHD algorithm with asymmetric devices described in **Figure 1E**.

in A is externally erased (i.e., A is never reset). Instead, A and C create a coupled-dynamical-system, as the changes performed on both are determined by the values of one another.

Furthermore, it is critical to realize that the algorithm shown in **Figure 2** consists of only fully-parallel operations. Similar to steps I and II (forward and backward pass on C), steps IV is yet another matrix-vector multiplication that is performed by means of Ohm's and Kirchhoff's Laws. On the other hand, the update steps III and V are performed by the stochastic update scheme (Gokmen and Vlasov, 2016). This update method does not explicitly compute the outer products ($x \times \delta$ and $u \times v$), but instead uses a statistical method to modify all weights in parallel proportional to those outer products. As a result, no serial operations are required at any point throughout the training operation, enabling high throughput and energy efficiency benefits in deep learning computations.

For the same linear regression problem studied above, the discrete-time update rules given in **Figure 2A** can be rewritten as a pair of differential equations in the continuum limit that describe the time evolution of subsystems A and C (**Figure 2C**) as:

$$\dot{A} = -\eta_A \left[\frac{\partial E}{\partial C} + \epsilon(t)\right] - \eta_A \kappa_A \left|\frac{\partial E}{\partial C} + \epsilon(t)\right| \times (A_{main} - A_{main,\,symmetry}) \tag{3}$$

$$\dot{C} = \eta_C A + \eta_C \kappa_C |A| \times (C_{main} - C_{main,symmetry}) \tag{4}$$

It can be noticed that this description of the coupled system has the same arrangement as the equations governing the motion of a damped harmonic oscillator (**Figures 2B,D**). In this analogy, subsystem A corresponds to velocity, v, while subsystem C maps to position, x, allowing the scalar error function of the optimization problem[3], $(C - C_0)^2$, to map onto the scalar potential energy of the physical framework, $\frac{1}{2}k_{spring}\,(x - x_0)^2$. Moreover, for implementations with asymmetric devices, an additional force term, $F_{hardware}$, needs to be included in the differential equations to reflect the hardware-induced effects on the conductance modulation. As discussed earlier, for the device model shown in **Figure 1E** this term is proportional to $A_{main} - A_{main,symmetry}$. If we assume $A_{ref} = A_{main,symmetry}$ (this assumption will be explained later), we can rewrite $F_{hardware}$ as a function of $A_{main} - A_{ref}$, which then resembles a drag force, F_{drag}, that is linearly proportional to velocity ($v \propto A = A_{main} - A_{ref}$) with a variable (but strictly non-negative) drag coefficient k_{drag}. In general, the $F_{hardware}$ term can have various functional forms for devices with different conductance modulation characteristics but is completely absent for ideal devices. Note that, only to simplify the physical analogy, we ignore the effect of asymmetry in subsystem C, which yields the equation shown in **Figure 2C** (instead of Equation 4). This decision will be justified in the Section Discussions.

Analogous to the motion of a lossless harmonic oscillator, the steady-state solution for this modified optimization problem with ideal devices (i.e., $F_{hardware} = 0$) has an oscillatory behavior (**Figure 2E**). This result is expected, as in the absence of any dissipation mechanism, the total energy of the system cannot be minimized (it is constant) but can only be continuously transformed between its potential and kinetic components. On the other hand, for asymmetric devices, the dissipative force term $F_{hardware}$ gradually annihilates all energy in the system, allowing $A \propto v$ to converge to 0 ($E_{kinetic} \rightarrow 0$) while $C \propto x$ converges to $C_0 \propto x_0$ ($E_{potential} \rightarrow 0$). Based on these observations, we rename the new training algorithm as *Stochastic Hamiltonian Descent (SHD)* to highlight the evolution of the system parameters in the direction of reducing the system's total energy (Hamiltonian). These dynamics can be visualized by plotting the time evolution of A vs. that of C, which yields a spiraling path representing decaying oscillations for the optimization process with asymmetric devices (**Figure 2F**), in contrast to elliptical trajectories observed for ideal lossless systems (**Figure 2E**).

Following the establishment of the necessity to have dissipative characteristics, here we analyze conditions at which device asymmetry provides this behavior. It is well-understood in mechanics that for a force to be considered dissipative, its product with velocity (i.e., power) should be negative (otherwise it would imply energy injection into the system). In other words, the hardware-induced force term $F_{hardware} = -\kappa_A \eta_A \left|\frac{\partial E}{\partial C} + \epsilon(t)\right| (A_{main} - A_{main,symmetry})$ and the velocity, $v = A_{main} - A_{ref}$, should always have opposite signs. Furthermore, from the steady-state analysis, for the system to be stationary ($v = 0$) at the point with minimum potential energy ($x = x_0$), there should be no net force ($F = 0$). Both of these arguments indicate that, for the SHD algorithm to function properly, A_{ref} must be set to $A_{main,\,symmetry}$. Note that as long as the crosspoint elements are realized with asymmetric devices (opposite to SGD requirement) and a symmetry point exists for each device, the shape of their modulation characteristics is not critical for successful DNN training with the SHD algorithm. Importantly, while a technologically viable solution for symmetric devices has not yet been found over decades of investigation, asymmetric devices that satisfy the aforementioned properties are abundant.

A critical aspect to note is that the SGD and the SHD algorithms are fundamentally disjunct methods governed by completely different dynamics. The SGD algorithm attempts to optimize the system parameters while disregarding the effect of device asymmetry and thus converges to the minimum of a wrong energy function. On the other, the system variables in an SHD-based training do not conventionally evolve in directions of the error function gradient, but instead, are tuned to minimize the total energy incorporating the hardware-induced terms. The most obvious manifestation of these properties can be observed when the training is initialized from the optimal point (i.e., the very lucky guess scenario) since any "training" algorithm should at least be able to maintain this optimal state. For the conventional SGD, when $W = W_0$, the zero-mean updates applied to the network were shown above to drift W away from W_0 toward $W_{symmetry}$. On the other hand, for the SHD method, when $A = 0$ and $C = C_0$, the zero-mean updates applied on A do not have any adverse effect since A_{main} is already at $A_{main,\,symmetry}$ for $A = 0$. Consequently, no updates are applied to C either as $\dot{C} = A = 0$. Therefore, it is clear that SGD is fundamentally incompatible with asymmetric devices, even when the solution is guessed correctly from the beginning, whereas the SHD does not suffer from this problem. Note that the propositions made for SGD can be further generalized to other crossbar-compatible training methods such as equilibrium propagation (Scellier and Bengio, 2017) and deep Boltzmann machines (Salakhutdinov and Hinton, 2009), which can also be adapted to be used with asymmetric devices following the approach discussed in this paper.

Finally, we appreciate that large-scale neural networks are much more complicated systems with respect to the problem analyzed here. Similarly, different analog devices show a wide range of conductance modulation behaviors, as well as bearing other non-idealities such as analog noise, imperfect retention, and limited endurance. However, the theory we provide here finally provides an intuitive explanation for: (1) why device asymmetry is fundamentally incompatible with SGD-based training and (2) how to ensure accurate optimization while only using fully-parallel operations. We conclude that asymmetry-related issues within SGD should be analyzed in the context of competing equilibria, where the optimum for the classification problem is not even a stable solution at steady-state. In addition to this simple stability analysis, the insight to modify the optimization landscape

[3]Conventionally error functions are written in terms of the difference between the network response and the target output and gradients are computed accordingly. However, in the absence of any stochasticity, ϵ, it can instead be written in terms of the network weights and their optimal values as well for notational purposes.

to include non-ideal hardware effects allows other fully-parallel solutions to be designed in the future using advanced concepts from optimal control theory. As a result, these parallel methods enable analog processors to provide high computational throughput and energy efficiency benefits over their conventional digital counterparts.

EXPERIMENTAL DEMONSTRATION

In order to validate the SHD dynamics theorized above, we carried out an experimental demonstration of the SHD algorithm using metal-oxide based electrochemical devices reported in Sebastian et al. (2017) (**Figure 3A**). These devices are three-terminal[4], voltage-controlled crosspoint elements, absent of any compliance circuits or serial-access devices. The modulation characteristics obtained for one of the devices is shown in **Figure 3B**, where "crossed-swords" behavior is observed with a well-defined symmetry point.

To capture the essence of SHD-based training, we have chosen a 2-parameter optimization problem with a synthetic dataset $x_{1,2}$ and y generated of form $y = G_{0_1}x_1 + G_{0_2}x_2 + \gamma$, where $G_{01,2}$ are the unknowns searched for and γ is the Gaussian noise. During the forward and backward pass cycles, input values (from the training set) were represented with different voltage levels and output results were obtained *via* measuring the line currents. We note that in an actual implementation representing input values with different pulse widths rather than amplitudes might be beneficial, avoiding the impact of the non-linear conductance of the crosspoint elements for accurate vector-matrix multiplication. Following the generation of the update vectors, x and δ, the array is programmed in parallel using stochastic updating with half-bias voltage scheme, as explained in Gokmen and Vlasov (2016). Therefore, we neither computed the outer product explicitly nor accessed the devices serially at any point (**Figure 3C**).

The array training results using the SHD algorithm are shown in **Figure 3D**. It can be seen that both A_1 and A_2 converges to 0, while C_1 and C_2 successfully converge to the optimal values. Moreover, the distinctive spiraling behavior (i.e., decaying oscillations) was observed for both variables, displaying analogous dynamics to dissipative mechanical systems. We found that the success of the training operation strongly depends on the stability of the devices' symmetry points. As discussed earlier, any discrepancy between the symmetry point and the reference point (initialized to the symmetry point at the beginning of training) of a device indicates a non-zero steady-state velocity. Therefore, future crosspoint device technologies should exhibit a well-defined symmetry point that is at least quasi-static throughout the training operation.

[4]SHD algorithm is compatible with various configurations of resistive device, such as 2-terminal devices, as well as 3-terminal devices we show here.

DISCUSSION AND SIMULATED TRAINING RESULTS

In this section, we first discuss how to implement the SHD algorithm with 3 arrays (instead of 4) using the intuition obtained from the theoretical analysis of the coupled-system. Then we provide simulated results for a large-scale neural network for different asymmetry characteristics to benchmark our method against SGD-based training.

Considering a sequence of $m + n$ incremental and n decremental changes at random order, the net modulation obtained for a symmetric device is on average m. On the other hand, we have shown above that for asymmetric devices the conductance value eventually converges to the symmetry point for increasing n (irrespective of m or the initial conductance). It can be seen by inspection that for increasing statistical variation present in the training data (causing more directional changes for updates), the effect of device asymmetry gets further pronounced, leading to heavier degradation of classification accuracy for networks trained with conventional SGD (see **Supplementary Figure S1**). However, this behavior can alternatively be viewed as non-linear filtering, where only signals with persistent sign information, $\frac{m}{m+2n}$, are passed. Indeed, the SHD algorithm exploits this property within the auxiliary array, A, which filters the gradient information that is used to train the core array, C. As a result, C is updated with less frequency and only in directions with a high confidence level of minimizing the error function of the problem at hand. A direct implication of this statement is that the asymmetric modulation behavior of C is much less critical than that of A (see **Supplementary Figure S2**) for successful optimization as its update signal contains less amount of statistical variation. Therefore, symmetry point information of C_{main} is not relevant either. Using these results and intuition, we modified the original algorithm by discarding C_{ref} and using A_{ref} (set to $A_{main,\ symmetry}$) as a common reference array for differential readout. This modification reduces the hardware cost of SHD implementations by 50% to significantly improve their practicality.

Our description of asymmetry as the mechanism of dissipation indicates that it is a necessary and useful device property for convergence within the SHD framework (**Figure 2E**). However, this argument does not imply that the convergence speed would be determined by the magnitude of device asymmetry for practical-sized applications. Unlike the single-parameter regression problem considered above, the exploration space for DNN training is immensely large, causing optimization to take place over many iterations of the dataset. In return, the level of asymmetry required to balance (i.e., damp) the system evolution is very small and can be readily achieved by any practical level of asymmetry.

To prove these assertions, we show simulated results in **Figure 4** for a Long Short-Term Memory (LSTM) network, using device models with increasing levels of asymmetry, trained with both the SGD and SHD algorithms. The network was trained on Leo Tolstoy's War and Peace novel, to predict the next character for a given text string (Karpathy et al., 2015).

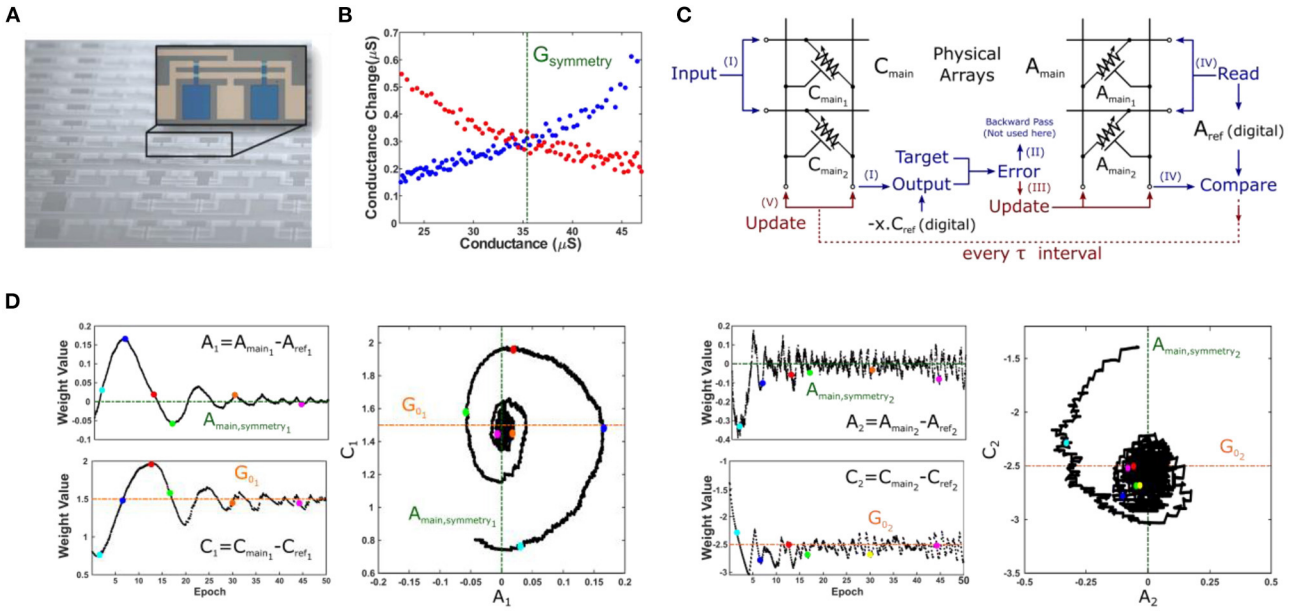


FIGURE 3 | Experimental demonstration of SHD training algorithm. **(A)** Optical micrograph of metal-oxide based electrochemical devices (Sebastian et al., 2017). Note that the image shows an integrated array whereas experiments were conducted with individual devices connected externally. **(B)** Conductance modulation characteristics obtained for one of the devices, showing "crossed-swords" behavior with a well-defined symmetry point. **(C)** Schematic for array configuration used in 2-parameter optimization with SHD algorithm. All steps are shown using the same notation used in **Figure 2** except for the backward pass (Step *II*) which is not required for a single layer network. For training, sum of squared errors is used to calculate the scalar error and vector δ, C_{main} is updated once every 10 samples (i.e., $\tau = 10$) whereas [1, 0] and [0, 1] were used in Step *IV* (as u vectors). The reference arrays containing symmetry point information are stored in digital (as they remain unchanged throughout the training) for simplicity. **(D)** Evolution of device conductances for the first (A_1, C_1) and the second (A_2, C_2) parameters. Plotting the values of A vs. C produces the distinctive spiraling image, as expected from the theoretical analysis.

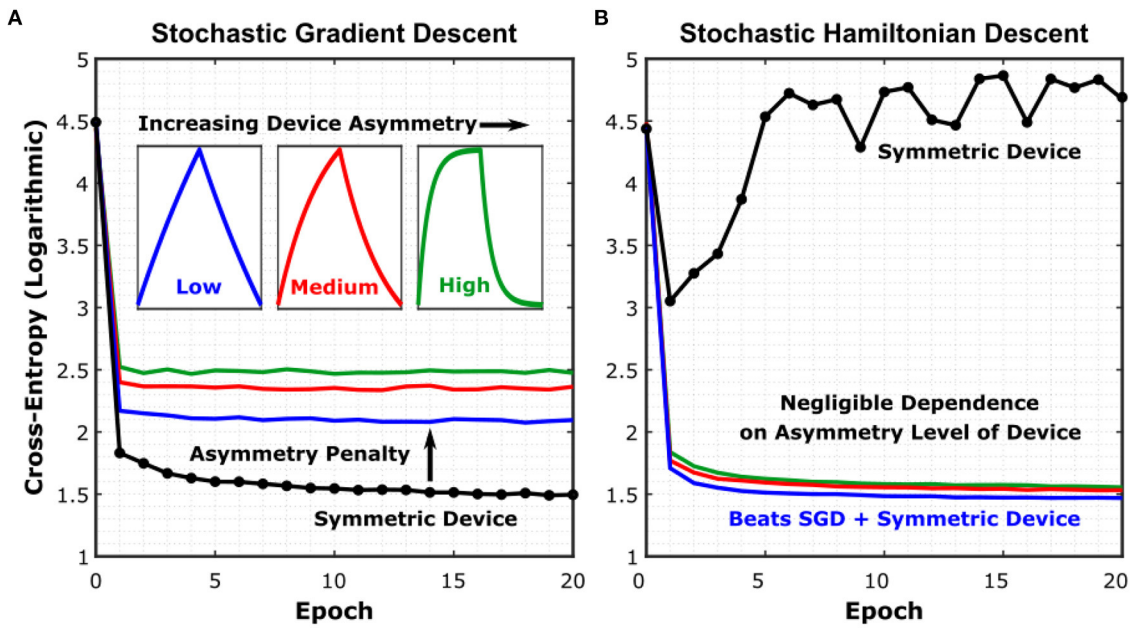


FIGURE 4 | Simulated training results for different resistive device technologies. **(A)** Simulated learning curves of a Long Short-Term Memory (LSTM) network trained on Leo Tolstoy's War and Peace novel, using different crosspoint device models under the SGD algorithm. Details of the network can be found in Karpathy et al. (2015). **(B)** Simulated learning curves for the same network using the SHD algorithm. All simulation details can be found in Section M3. Training Simulator and LSTM Network. See **Supplementary Figure S4** for device-to-device variation included in the simulations and **Supplementary Figure S6** for floating-point baseline comparison.

For reference, training the same network with a 32-bit digital floating-point architecture yields a cross-entropy level of 1.33 (complete learning curve shown in **Supplementary Figure S6**). We have particularly chosen this network as LSTM's are known for being particularly vulnerable to device asymmetry (Gokmen et al., 2018).

The insets in **Figure 4** show the average conductance modulation characteristics representative for each asymmetry level. The simulations further included device-to-device variation, cycle-to-cycle variation, analog read noise, and stochastic updating similar to the work conducted in Gokmen and Vlasov (2016). The learning curves show the evolution of the cross-entropy error, which measures the performance of a classification model, with respect to the epochs of training. First, **Figure 4A** shows that even for minimally asymmetric devices (blue trace) trained with SGD, the penalty in classification performance is already severe. This result also demonstrates once more the difficulty of engineering a device that is symmetric-enough to be trained accurately with SGD. On the other hand, for SHD (**Figure 4B**), all depicted devices are trained successfully, with the sole exception being the perfectly symmetric devices (black trace), as expected (see **Supplementary Figure S3** for devices with abrupt modulation characteristics). Furthermore, **Figure 4B** demonstrates that SHD can even provide training results with higher accuracy and faster convergence than those for perfectly symmetric devices trained with SGD. As a result, we conclude that SHD is generically superior to SGD for analog deep learning architectures.

Finally, although we present SHD in the context of analog computing specifically, it can also be potentially useful on conventional processors (with simulated asymmetry). The filtering dynamics described above allows SHD to guide its core component selectively in directions with high statistical persistence. Therefore, at the expense of increasing the overall memory and number of operations, SHD might outperform conventional training algorithms by providing faster convergence, better classification accuracy, and/or superior generalization performance.

CONCLUSION

In this paper, we described a fully-parallel neural network training algorithm for analog crossbar-based architectures, Stochastic Hamiltonian Descent (SHD), based on resistive devices with asymmetric conductance modulation characteristics, as is the case for all practical technologies. In contrast to previous work that resorted to serial operations to mitigate asymmetry, SHD is a fully-parallel and scalable method that can enable high throughput and energy-efficiency deep learning computations with analog hardware. Our new method uses an auxiliary array to successfully tune the system variables in order to minimize the total energy (Hamiltonian) of the system that includes the effect of device asymmetry. Standard techniques, such as Stochastic Gradient Descent, perform optimization without accounting for the effect of device asymmetry and thus converge to the minimum of a wrong energy function. Therefore, our theoretical framework describes the inherent fundamental incompatibility of asymmetric devices with conventional training algorithms. The SHD framework further enables the exploitation of device asymmetry as a useful feature to selectively filter and apply the updates only in directions with high confidence. The new insights shown here have allowed a 50% reduction in the hardware cost of the algorithm. This method is immediately applicable to a variety of existing device technologies, and complex neural network architectures, enabling the realization of analog training accelerators to tackle the ever-growing computational demand of deep learning applications.

METHODS

M1. Array Initialization (Zero-Shifting)

Initialization of the reference array requires identification of the conductance values of each and every element in A_{main}, and programming the reference array conductances (A_{ref}) to those values. Given that under those conditions $A = A_{main} - A_{ref}$ becomes 0, the method is also referred to as zero-shifting (Kim et al., 2019a). To identify $A_{main,\ symmetry}$, a sufficiently long sequence of increment-decrement pulses is applied to A_{main}. Given the asymmetric nature of the devices, each pair results in a residual conductance modulation toward each device's respective symmetry point. Following this step, the resultant $A_{main} = A_{main,\ symmetry}$ is then copied to the reference array. Since these steps only occur once per training, the time and energy costs are negligible with respect to the rest of the operation (even for serial copying).

M2. Pseudocode for SHD Algorithm

Initialize

k : *iteration step* $\leftarrow 1$, l : *layer index*

Set τ, η

For each layer

$A^l_{main}[k] = A^l_{main,symmetry}\ (m \times n\ matrix,\ dynamic)$

$A^l_{ref} = A^l_{main,symmetry} \quad (m \times n\ matrix,\ static)$

$C^l_{main}[k] = C^l_{main,symmetry}\ (m \times n\ matrix,\ dynamic)$

For each labeled data pair $[x_i,\ t_i]$

Convert input x_i to time encoded voltage pulse for the first layer (x^1_i)

$MAC\ O^1[k] = x^1_i[k].[C^1_{main}[k] - A^1_{ref}]$

Convert analog output to digital to store and apply non-linear functions (activations, pooling etc.)

Forward propagate $O^1[k]$ as the input for next layer (always using C^l_{main} arrays) for all *layers*

Compute error (cost) using network output $O^{final}[k]$ and target output $t[\ k]$

Backward propagate using the same dynamics (again using C^l_{main} arrays) to compute all error matrices δ^l[k]

Update $A^l_{main}[k+1] \leftarrow A^l_{main}[k] - \eta.x^l[k] \bigotimes \delta^l[k]$ using stochastic update scheme

If mod (k,τ) = 0

$u^l[k] = [0, 0, 0 \ldots 1, \ldots 0, 0]$, where "1" is at k^{th} location

MAC $v^l[k] = u^l[k].[A^l[k] - A^l_{ref}]$

$$\text{Update } C^l_{main}\left[k+1\right] \leftarrow C^l_{main}\left[k\right] - \eta.u^l[k] \bigotimes v^l[\,k]$$

M3. Training Simulator and LSTM Network

The simulation framework used here is the same that was used in Gokmen and Vlasov (2016), Gokmen et al. (2017, 2018), and Gokmen and Haensch (2020). The simulations start with instantiating 3 devices per weight. Each device parameter (e.g., number of states, asymmetry factor, and symmetry point) is generated with a given mean and standard variation, such that no two devices are the same. Moreover, these device parameters also bear cycle-to-cycle variation, defined by another parameter, to make the operation more realistic. An open access version of the simulator we used in this work can be found in github.com/ibm/aihwkit for reproduction of the results.

The incremental changes are set such that devices have on average 1,200 programmable states within their dynamic range. Through setting the gain factors at the integrator terminals appropriately, the average full conductance range of devices are adjusted to be equivalent $\pm$ 2 arbitrary units. Consistent with this notation, the integrators are set to saturate at $\pm$ 40 arbitrary units. We have used 9-bit resolution for the ADCs and 7-bit resolution for the DACs where the output-referred noise level was set at 0.02 arbitrary units. This selection was made in order not to be limited by noise-related performance degradation, as studied by Gokmen et al. (2017). In the update cycle, the maximum allowed number of pulses (i.e., bit length, BL) was set to be 100. However, as update management determines this number on-the-go depending on certain characteristics of the update vectors and device parameters, real BL was $<$10 for the most of the training.

The War and Peace dataset consists of 3, 258, 246 characters, which we split into training and test sets as 2, 933, 246 and 325, 000 characters, respectively. The network is trained to have a vocabulary of 87 distinct characters. We have selected to use hidden vectors of 64-cell size, which corresponds to $\sim$77K weights for the complete network. Full details of the network architecture can be found in Karpathy et al. (2015).

The selection of the LSTM problem studied here in detail is found to be optimal, which is complex enough to validate the training algorithm, while it still is trainable with limited number of conductance states, analog noise, variations, and limited resolution. Given that SHD only resolves asymmetry related issues, whereas other imperfections related with analog processors such as device-to-device variability, cycle-to-cycle variability, noise, and resolution can still deteriorate the training performance significantly, we recommend future studies to explore larger problems, once there are additional solutions for these other non-idealities related to analog crossbar architectures.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

MO and TG conceived the original idea and performed software experiments. TT fabricated devices. MO and SK performed hardware experiments. All authors contributed to the theory development and contributed to the preparation of the manuscript. All authors contributed to the article and approved the submitted version.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2022.891624/full#supplementary-material

REFERENCES

Agarwal, S., Gedrim, R. B. J., Hsia, A. H., Hughart, D. R., Fuller, E. J., Talin, A. A., et al. (2017). Achieving ideal accuracies in analog neuromorphic computing using periodic carry. *Symp. VLSI Technol.* 174–175. doi: 10.23919/VLSIT.2017.7998164

Agarwal, S., Plimpton, S. J., Hughart, D. R., Hsia, A. H., Richter, I., Cox, J. A., et al. (2016). Resistive memory device requirements for a neural algorithm accelerator. *Proc. Int. Jt. Conf. Neural Networks.* 929–938. doi: 10.1109/IJCNN.2016.7727298

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature.* 558, 60–67. doi: 10.1038/s41586-018-0180-5

Burr, G. W., Shelby, R. M., Sebastian, A., Kim, S., Sidler, S., Virwani, K., et al. (2017). Neuromorphic computing using non-volatile memory. *Adv. Phys.* 2, 89–124. doi: 10.1080/23746149.2016.1259585

Burr, G. W., Shelby, R. M., Sidler, S., Di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large-scale neural network (165 000 Synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Devices.* 62, 3498–3507. doi: 10.1109/TED.2015.2439635

Cai, F., Correll, J. M., Lee, S. H., Lim, Y., Bothra, V., Zhang, Z., et al. (2019). A fully integrated reprogrammable memristor– CMOS system for efficient multiply–accumulate operations. *Nat. Electron.* 2, 1. doi: 10.1038/s41928-019-0270-x

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris.* 25, 536–538.

Chen, Y., Member, S., Krishna, T., Emer, J. S., and Sze, V. (2016). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits.* 52, 127–138. doi: 10.1109/JSSC.2016.2616357

Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. *Proc. 2nd SysML Conf.* 348–359.

Feng, Y., and Tu, Y. (2023). *How Neural Networks Find Generalizable Solutions: Self-Tuned Annealing in Deep Learning.* Available online at: https://arxiv.org/abs/2001.01678 (accessed March 01, 2022).

Fuller, E. J., Keene, S. T., Melianas, A., Wang, Z., Agarwal, S., Li, Y., et al. (2019). Parallel programming of an ionic floating-gate memory array for scalable neuromorphic computing. *Science* 364, 570–574. doi: 10.1126/science.aaw5581

Gokmen, T., and Haensch, W. (2020). Algorithm for training neural networks on resistive device arrays. *Front. Neurosci.* 14, e00103. doi: 10.3389/fnins.2020.00103

Gokmen, T., Onen, M., and Haensch, W. (2017). Training deep convolutional neural networks with resistive cross-point devices. *Front. Neurosci.* 11, 538. doi: 10.3389/fnins.2017.00538

Gokmen, T., Rasch, M. J., and Haensch, W. (2018). Training LSTM networks with resistive cross-point devices. *Front. Neurosci.* 12, 745. doi: 10.3389/fnins.2018.00745

Gokmen, T., and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* 10, 333. doi: 10.3389/fnins.2016.00333

Grollier, J., Querlioz, D., Camsari, K. Y., Everschor-Sitte, K., Fukami, S., and Stiles, M. D. (2020). Neuromorphic spintronics. *Nat. Electron.* 3, 360–370. doi: 10.1038/s41928-019-0360-9

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). In - datacenter performance analysis of a tensor processing unit. *Proc. 44th Annu. Int. Symp. Comput. Archit.* 1–12. doi: 10.1145/3079856.3080246

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). "Visualizing and understanding recurrent networks", in *ICLR* 2016 (San Juan), 1–12.

Kim, H., Rasch, M., Gokmen, T., Ando, T., Miyazoe, H., Kim, J.-J., et al. (2020). *Zero-Shifting Technique for Deep Neural Network Training on Resistive Cross-point Arrays*. Available online at: https://arxiv.org/abs/1907.10228 (accessed March 01, 2022).

Kim, H., Rasch, M., Gokmen, T., Ando, T., Miyazoe, H., Kim, J. J., et al. (2019a). Zero-shifting Technique for deep neural network training on resistive cross-point arrays. *arXiv* 2019–2022.

Kim, S., Todorov, T., Onen, M., Gokmen, T., Bishop, D., Solomon, P., et al. (2019b). Oxide based, CMOS-compatible ECRAM for deep learning accelerator. *IEEE Int. Electron Devices Meet.* 847–850. doi: 10.1109/IEDM19573.2019.8993463

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature.* 521, 436–444. doi: 10.1038/nature14539

Li, C., Hu, M., Li, Y., Jiang, H., Ge, N., Montgomery, E., et al. (2018). Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* 1, 52–59. doi: 10.1038/s41928-017-0002-z

Li, C., Wang, Z., Rao, M., Belkin, D., Song, W., Jiang, H., et al. (2019). Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* 1, 49–57. doi: 10.1038/s42256-018-0001-4

Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*. 521, 61–64. doi: 10.1038/nature14441

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2020). *Zero: Memory Optimizations Toward Training Trillion Parameter Models*. Atlanta, GA: IEEE Press.

Rumelhart, D. E., Hinton, G. E., and Willams, R. J. (1986). Learning representations by back-propagating errors. *Nature*. 323, 533–536. doi: 10.1038/323533a0

Salakhutdinov, R., and Hinton, G. (2009). Deep Boltzmann machines. *J. Mach. Learn. Res.* 5, 448–455.

Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11, e00024. doi: 10.3389/fncom.2017.00024

Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E. (2020). Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* 15, 246–253. doi: 10.1038/s41565-020-0655-z

Sebastian, A., Tuma, T., Papandreou, N., Le Gallo, M., Kull, L., Parnell, T., et al. (2017). Temporal correlation detection using computational phase-change memory. *Nat. Commun.* 8. 1–10. doi: 10.1038/s41467-017-01481-9

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik*. 1, 36–45. doi: 10.1007/BF00293853

Strubell, E., Ganesh, A., and McCallum, A. (2020). Energy and policy considerations for deep learning in NLP. *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.* 3645–3650. doi: 10.18653/v1/P19-1355

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V., et al. (2019). Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. *Adv. Neural Inf. Process. Syst.*

Woo, J., and Yu, S. (2018). Resistive memory-based analog synapse: the pursuit for linear and symmetric weight update. *IEEE Nanotechnol. Mag.* 12, 36–44. doi: 10.1109/MNANO.2018.2844902

Yao, X., Klyukin, K., Lu, W., Onen, M., Ryu, S., Kim, D., et al. (2020). Protonic solid-state electrochemical synapse for physical neural networks. *Nat. Commun.* 11, 1–10. doi: 10.1038/s41467-020-16866-6

Yu, S., Chen, P. Y., Cao, Y., Xia, L., Wang, Y., and Wu, H. (2015). Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect. *Tech. Dig. - Int. Electron Devices Meet. IEDM*. 17–3. doi: 10.1109/IEDM.2015.7409718

Conflict of Interest: MO, TG, TT, TN, JR, WH, and SK were employed by IBM Thomas J. Watson Research Center.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Advantages of publishing in Frontiers

OPEN ACCESS
Articles are free to read for greatest visibility and readership

FAST PUBLICATION
Around 90 days from submission to decision

HIGH QUALITY PEER-REVIEW
Rigorous, collaborative, and constructive peer-review

TRANSPARENT PEER-REVIEW
Editors and reviewers acknowledged by name on published articles

Frontiers
Avenue du Tribunal-Fédéral 34
1005 Lausanne | Switzerland

Visit us: www.frontiersin.org
Contact us: frontiersin.org/about/contact

REPRODUCIBILITY OF RESEARCH
Support open data and methods to enhance research reproducibility

DIGITAL PUBLISHING
Articles designed for optimal readership across devices

FOLLOW US
@frontiersin

IMPACT METRICS
Advanced article metrics track visibility across digital media

EXTENSIVE PROMOTION
Marketing and promotion of impactful research

LOOP RESEARCH NETWORK
Our network increases your article's readership