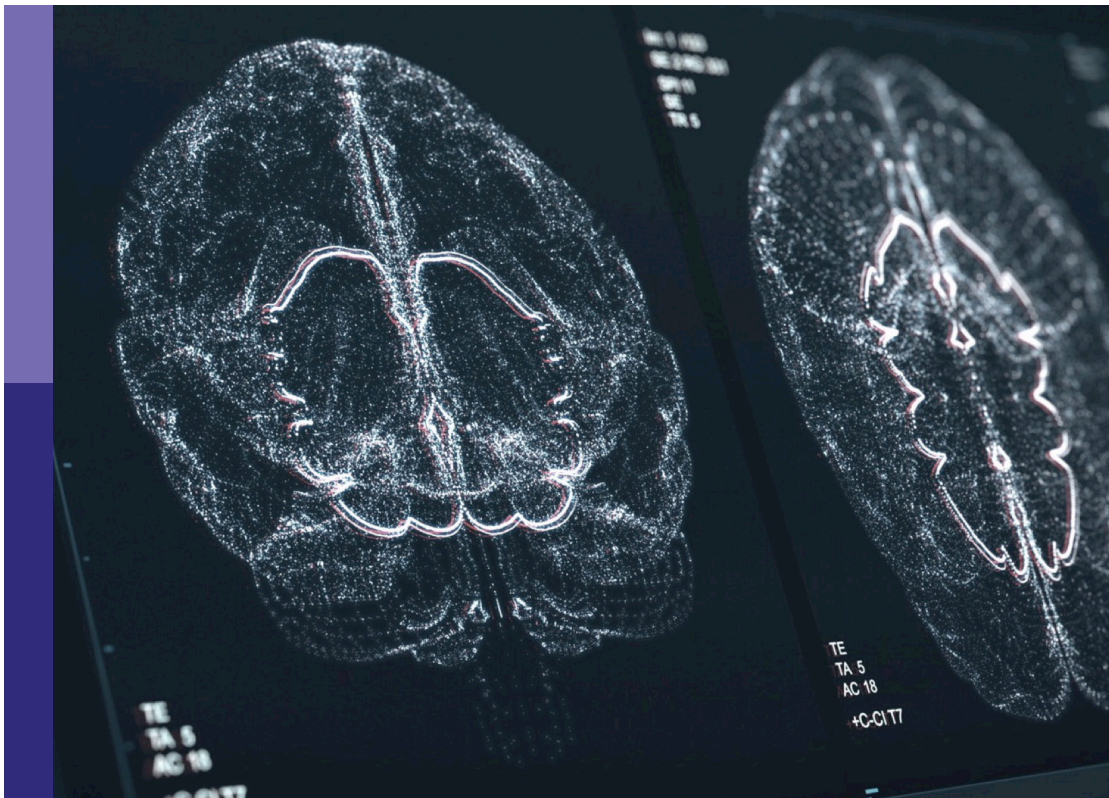
frontiers | Research Topics

Physical neuromorphic computing and its industrial applications

Edited by
Toshiyuki Yamane, Akira Hirose and Bert Offrein

Published in
Frontiers in Neuroinformatics
Frontiers in Neuroscience



ISSN 1664-8714
ISBN 978-2-8325-3128-0
DOI 10.3389/978-2-8325-3128-0

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

Physical neuromorphic computing and its industrial applications

Topic editors

Toshiyuki Yamane — IBM Research, Tokyo, Japan

Akira Hirose — The University of Tokyo, Japan

Bert Offrein — IBM Research - Zurich, Switzerland

Citation

Yamane, T., Hirose, A., Offrein, B., eds. (2023). *Physical neuromorphic computing and its industrial applications*. Lausanne: Frontiers Media SA.

doi: 10.3389/978-2-8325-3128-0

Table of contents

frontiers | Frontiers in Neuroinformatics

TYPE Editorial
PUBLISHED 17 July 2023
DOI 10.3389/fninf.2023.1238168

OPEN ACCESS

EDITED AND REVIEWED BY
Michael Denker,
Institute of Neuroscience and Medicine (INM-6/INM-10), Germany

*CORRESPONDENCE
Toshiyuki Yamane
✉ tyamane@jp.ibm.com

RECEIVED 10 June 2023
ACCEPTED 04 July 2023
PUBLISHED 17 July 2023

CITATION
Yamane T, Hirose A and Offrein BJ (2023) Editorial: Physical neuromorphic computing and its industrial applications.
Front. Neuroinform. 17:1238168.
doi: 10.3389/fninf.2023.1238168

Editorial: Physical neuromorphic computing and its industrial applications

Toshiyuki Yamane[1]*, Akira Hirose[2] and Bert Jan Offrein[3]

[1]IBM Research - Tokyo, Tokyo, Japan, [2]The University of Tokyo, Tokyo, Japan, [3]IBM Research—Zurich, Rüschlikon, Switzerland

KEYWORDS

analog neuromorphic devices, electric neuromorphic computing, material neuromorphic computing, quantum neuromorphic computing, IoT and edge computing

Editorial on the Research Topic
Physical neuromorphic computing and its industrial applications

1. Introduction

The importance of handling cognitive data such as images, voices, and natural languages is wide spreading not only at datacenters but also at networking, edge, and IoT. Artificial neural networks are a powerful and prospective concept to process such cognitive data.

The improved performance of neural networks is achieved by increasing the scale of neural network models. This unavoidably has a direct and tremendous impact on the energy required to training and inference by current software and general-purpose processors due to their serial operation. On the other hand, current hardware acceleration is based on well-matured ASIC technology and integrated electronics. However, with the downscaling limit of conventional technologies, the traditional electronic computing will face difficulties in further growing in terms of energy efficiency.

To address the power and performance constraints, neuromorphic computing is a promising approach. In fact, various CMOS-based neuromorphic devices have been reported so far. In recent years, motivated by potential computational capabilities of various natural physical phenomena, unconventional computing paradigms have been actively investigated in the interdisciplinary region of computer science and natural science. The objective of this Research Topic is to investigate the possibility of incorporating diverse natural physical phenomena to neuromorphic computing, which we call "physical neuromorphic computing."

In this Research Topic, we collected nine papers relevant to the theory, algorithm, and implementation of physical neuromorphic computing. They can be roughly classified into the following categories: electric, material, and quantum neuromorphic computing.

2. Electric neuromorphic computing

Undoubtedly, electric neuromorphic computing is the most actively investigated research area in neuromorphic computing, where non von-Neumann architectures are

pursued with brain-like features such as distributed and sparse information representations, massive parallelism, event-driven operation, analog signal processing and on-chip learning capability.

Stapmanns et al. derived two efficient algorithms for archiving postsynaptic membrane potentials, based on event-based synapse updates and compared two algorithms with a time-driven synapse update scheme in terms of memory and computations. They showed that the two event-based algorithms significantly outperform the time-driven scheme. Their results on information archiving efficiency provide guidelines for the design of learning rules and make them practical in large-scale networks.

Michaelis et al. developed an open source emulator named Brian2Loihi for Loihi, which is a neuromorphic many core processor for spiking neural network with on-chip learning capability. They demonstrated error-free emulation for a single neuron and a recurrent spiking neural network and implementation of on-chip learning. Their work provides a quick prototyping and deployment of new algorithms for Loihi.

Hopkins et al. presented a new concept "sparse binary coincidence (SBC) memory" and its realization on surrounding infrastructure called BitBrain. The SBC memory stores coincidences between features in a training set and infers the class of a test example by identifying the class with which it shares the highest number of feature coincidences. They applied these concepts to the MNIST and EMNIST benchmarks and showed very low training costs and robustness to noise. BitBrain is designed to be implemented efficiently on both neuromorphic devices such as SpiNNaker and conventional CPU and memory architectures and is well-suited for edge and IoT applications.

Md Abdullah-Al Kaiser et al. proposed an asynchronous non von-Neumann analog processing-in-pixel architecture to perform convolutional multiply and accumulate (MAC) operations by integrating *in-situ* multi-bit multi-channel convolution inside the pixel array. They verified the architecture on vision sensor datasets and showed that the solution consumes significantly less energy than their digital MAC alternative; less than half of the backend-processor energy while retaining front-end energy and a high test accuracy.

3. Material neuromorphic computing

The major energy consumer of today's digital processor is data movement between MAC processors and volatile main memories. This motivates integration of memory and computation called "in-memory computing." Use of intrinsic properties of materials for in-memory computing is becoming very promising approach for in-memory computing.

Gokmen and Haensch presented a new training algorithm, called "Tiki-Taka" algorithm for deep neural networks on resistive cross-point device arrays. Tiki-Taka alleviates stringent symmetry requirement that resistive devices must change conductance symmetrically for positive or negative pulse stimuli. Simulation results show that the accuracy of the SGD algorithm with symmetric device switching characteristics is matched in that of the Tiki-Taka algorithm with non-symmetric device switching characteristics.

Corti et al. presented an in-memory computing platform for convolutional neural networks by synchronization in phase and frequency of coupled VO2 oscillators. The neuromorphic architecture was fabricated in a crossbar configuration on silicon and achieved significant improvements of area density, oscillation frequency, variability and reliability, compared to existing digital convolutional filters. They applied the platform to MNIST recognition task and achieved high recognition accuracy.

Garg et al. demonstrated that the phase synchronization of glial cells can be reproduced by injected radio-frequency signals in the heavy metal layer of spin-orbit torque oscillators. They also proposed applications of such neural synchronization to the temporal binding problem and the design of a coupled neuron-synapse-astrocyte network.

4. Quantum neuromorphic computing

Quantum computing is an emerging technology based on completely different principles from classical computers, the laws of quantum mechanics. Quantum computing, if it happens in reality, has the potential to solve many industrial problems that classical computers cannot with a reasonable amount of resources. The nonlinearity of the quantum devices used in quantum computing can be applied to energy-efficient neuromorphic computing.

Tschirhart and Segall investigated how superconducting electronics by Josephson junctions address the requirements for large scale neuromorphic systems, such as scalability, programmability, biological fidelity, on-line STDP learning, efficiency, and speed. The result of detailed numerical analysis based on digital logic demonstrations showed that superconducting electronics is suitable for fast and efficient neuromorphic experimental platform in the future.

Rahman et al. reported a differential device by Fowler-Nordheim (FN) quantum-mechanical tunneling. They showed that a prototype FN-synapse array can achieve near-optimal memory consolidation characteristics with tunable plasticity-stability trade-offs, compared to other physical implementations. They also claimed that the proposed FN-synapse provides an ultra-energy-efficient approach for implementing both synaptic memory consolidation and continual learning in terms of an energy footprint per synaptic update.

Author contributions

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

Conflict of interest

TY and BO were employed by IBM Research.

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

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 26 February 2020
doi: 10.3389/fnins.2020.00103

Algorithm for Training Neural Networks on Resistive Device Arrays

Tayfun Gokmen and Wilfried Haensch*

IBM Research AI, Yorktown Heights, NY, United States

Hardware architectures composed of resistive cross-point device arrays can provide significant power and speed benefits for deep neural network training workloads using stochastic gradient descent (SGD) and backpropagation (BP) algorithm. The training accuracy on this imminent analog hardware, however, strongly depends on the switching characteristics of the cross-point elements. One of the key requirements is that these resistive devices must change conductance in a symmetrical fashion when subjected to positive or negative pulse stimuli. Here, we present a new training algorithm, so-called the "Tiki-Taka" algorithm, that eliminates this stringent symmetry requirement. We show that device asymmetry introduces an unintentional implicit cost term into the SGD algorithm, whereas in the "Tiki-Taka" algorithm a coupled dynamical system simultaneously minimizes the original objective function of the neural network and the unintentional cost term due to device asymmetry in a self-consistent fashion. We tested the validity of this new algorithm on a range of network architectures such as fully connected, convolutional and LSTM networks. Simulation results on these various networks show that the accuracy achieved using the conventional SGD algorithm with symmetric (ideal) device switching characteristics is matched in accuracy achieved using the "Tiki-Taka" algorithm with non-symmetric (non-ideal) device switching characteristics. Moreover, all the operations performed on the arrays are still parallel and therefore the implementation cost of this new algorithm on array architectures is minimal; and it maintains the aforementioned power and speed benefits. These algorithmic improvements are crucial to relax the material specification and to realize technologically viable resistive crossbar arrays that outperform digital accelerators for similar training tasks.

Keywords: deep learning, resistive device, analog hardware accelerator, resistive processing unit, training algorithms, memristor, crossbar array

OPEN ACCESS

Edited by:
Damien Querlioz,
Centre National de la Recherche Scientifique (CNRS), France

Reviewed by:
Huaqiang Wu,
Tsinghua University, China
Doo Seok Jeong,
Hanyang University, South Korea

****Correspondence:***
Tayfun Gokmen
tgokmen@us.ibm.com

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

Received: *06 October 2019*
Accepted: *27 January 2020*
Published: *26 February 2020*

Citation:
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

INTRODUCTION

In the past few years, deep neural networks (DNN) (LeCun et al., 2015) have made tremendous advances, in some cases surpassing human level performance, tackling challenging problems such as speech recognition (Hinton et al., 2012; Ravanelli et al., 2017), natural language processing (Collobert et al., 2012; Jozefowicz et al., 2016), image classification (Krizhevsky et al., 2012; He et al., 2015a,b; Chen et al., 2017), and machine translation (Wu, 2016). Training of large DNNs, however, is a time consuming and computationally intensive task that demands datacenter scale computational resources composed of state of the art GPUs (Krizhevsky et al., 2012; Coates et al., 2013). There have been many attempts to accelerate deep learning workloads beyond GPUs by

designing custom hardware utilizing reduced precision arithmetic to improve the throughput and energy efficiency of the underlying CMOS technology (Gupta et al., 2015). Alternative to digital approaches, resistive cross-point device arrays are proposed to further increase the throughput and energy efficiency of the overall system by performing the vector-matrix multiplications in the analog domain. In addition, these device arrays can perform the weight update operation locally with no weight movement and therefore they bring further benefits compared to digital approaches.

Resistive cross-point devices, so called resistive processing unit (RPU) (Gokmen and Vlasov, 2016) device arrays that can simultaneously store and process data locally and in parallel, are promising candidates for intensive DNN training workloads. The concept of using resistive cross-point device arrays (Burr et al., 2015, 2017; Chen et al., 2015a,b; Prezioso et al., 2015; Agarwal et al., 2016b; Gokmen and Vlasov, 2016; Fuller et al., 2017) as DNN accelerators has been tested on a variety of network architectures and datasets mainly by simulations but also with some limited hardware demonstrations. Considering state-of-the-art learning algorithms, to par training accuracy compared to the conventional digital hardware a restrictive set of RPU device specifications must be met. As shown empirically (Agarwal et al., 2016a; Gokmen and Vlasov, 2016; Gokmen et al., 2017), a key requirement is that these analog resistive devices must change conductance symmetrically when subjected to positive or negative voltage pulse stimuli. This requirement differs significantly from those needed for memory elements and accomplishing such symmetrically switching analog devices is a difficult task. Substantial efforts are devoted to engineer new material stacks or adopt the existing ones, originally developed for memory (Burr et al., 2015) and battery (Fuller et al., 2017; Tang et al., 2018) applications, to achieve the symmetry criteria needed for DNN training. Besides material engineering efforts, CMOS only (Li et al., 2018) and CMOS assisted solutions in tandem with existing memory device technologies (Ambrogio et al., 2018) are also considered but introduce an overhead of making the cross-point element increasingly more complex.

Here, we present a new technique that can address the issue of non-symmetric device switching characteristics at the algorithm level, in a physical-hardware invariant form. In the rest of the paper we show that the device switching characteristics introduces an additional cost term into the optimization objective of the conventional SGD algorithm. The presence of this additional term entails poor training results for non-symmetric devices as the system is in competition with minimizing the objection function of the neural network against this unintentional cost term. In this new technique we introduce a coupled dynamical system that simultaneously minimizes the objective function of the original SGD algorithm as well as the unintentional cost term due to device asymmetry in a self-consistent fashion. This algorithm learns by continuously exchanging information between two system's components and hence we call it the "Tiki-Taka" algorithm. We show that the "Tiki-Taka" algorithm is general enough to handle a large range of non-symmetric device switching behaviors and therefore applicable to a variety of device technologies. We tested the algorithm by performing training simulations using various device switching characteristics on three different network architectures: fully connected, convolutional and LSTMs. In all cases the results of the training performed with the "Tiki-Taka" algorithm using non-ideal device characteristics are indistinguishable from the ones achieved with the SGD algorithm using ideal devices. We also discuss the implementation cost of the "Tiki-Taka" algorithm on realistic RPU device arrays in terms of area, power and speed and show that the overall cost is minimal.

MATERIALS AND METHODS

Array Operations: Forward, Backward, and Update

The stochastic gradient descent (SGD) using the backpropagation algorithm is composed of three cycles – forward, backward and weight update – that are repeated many times until a convergence criterion is met. For a single fully connected layer where N inputs neurons are connected to M output (or hidden) neurons, the forward cycle involve computing a vector-matrix multiplication ($y = Wx$) where the vector x of length N represents the activities of the input neurons and the matrix W of size $M \times N$ that stores the weight values between each pair of input and output neurons. The resulting vector y of length M is further processed by performing a non-linear activation on each of the elements and then passed to the next layer. Once the information reaches to the final output layer, the error signal is calculated and backpropagated through the network. The backward cycle on a single layer also involves a vector-matrix multiplication on the transpose of the weight matrix ($z = W^{\top}\delta$), where the vector δ of length M represents the error calculated by the output neurons and the vector z of length N is further processed using the derivative of neuron non-linearity and then passed down to the next (previous) layer. Finally, in the update cycle the weight matrix W is updated by performing an outer product of the two vectors that are used in the forward and the backward cycles and usually expressed as $W \leftarrow W - \eta\ (\delta \otimes x)$ where η is a global learning rate. Consistently, the SGD update rule for each parameter w_{ij} corresponding to i^{th} column and j^{th} row (the layer index is dropped for simplicity) can be written as

$$w_{ij} \leftarrow w_{ij} - \eta \Delta w_{ij} \tag{1}$$

where Δw_{ij} is the gradient of the objective function with respect to parameter w_{ij}, and has a form $\Delta w_{ij} = x_i \times \delta_j$, where x_i is the input activation for the i^{th} column and δ_j is the backpropagated error signal for the j^{th} row.

The above three operations performed on the weight matrix W during the SGD\BP algorithm are implemented using 2D crossbar arrays of resistive devices all in parallel and constant time using the physical properties of the array. For instance, the stored conductance values in the crossbar array form the matrix W, however, physically only positive conductance values are allowed and therefore to encode both positive and negative weight values a pair of RPU devices is operated in differential

mode. For each parameter w_{ij} in the weight matrix W, there exists two devices that encode a single weight value

$$w_{ij} = \mathrm{K}(g_{ij} - g_{ij,ref}) \tag{2}$$

where g_{ij} is the conductance value stored on the first RPU device, $g_{ij,ref}$ is the conductance value stored on the second device used as a reference both corresponding to i^{th} column and j^{th} row and K is the gain factor that is controlled by a combination of factors, such as integration time, integration capacitor and voltage levels, at the peripheral circuitry. In the forward cycle, the input vector x is transmitted as voltage pulses through each of the columns and resulting vector y is read as a differential current signal from the rows (Steinbuch, 1961). Similarly, the backward cycle can be performed by inputting voltage pulses from the rows and reading the results from the columns. These two cycles simply rely on Ohm's law and the Kirchhoff's law in order to perform the vector-matrix multiplications. In contrast to the forward and backward cycles, implementing the update cycle is trickier and employs the device switching characteristics to trigger the necessary conductance change $\Delta g_{total,ij}$ that should practically match the required weight change $\eta \Delta w_{ij}$ of the SGD algorithm, such that $\mathrm{K}\Delta g_{total,ij} \cong \eta(x_i \times \delta_j)$. To perform the local multiplication operation needed to calculate $\Delta w_{ij} = x_i \times \delta_j$, different pulse encoding schemes (Xu et al., 2014; Burr et al., 2015; Gokmen and Vlasov, 2016) are proposed all of which reduce the multiplication to a simple coincidence detection that can be realized by RPU devices. For instance, in the stochastic update scheme numbers that are encoded from the columns and rows (x_i and δ_j) are translated to stochastic bit streams using stochastic translators (Gokmen and Vlasov, 2016). These stochastic translators adjust the pulse probabilities at the periphery, and hence they control the total number of the pulse coincidences happening at each crossbar element. In this scheme these pulses are sent into the crossbar array simultaneously for all rows and all columns and then for each coincidence event the corresponding RPU device changes its conductance by a small amount Δg_{min}. However, there exist many pulses in the pulse stream so that the total conductance change $\Delta g_{total,\ ij}$ required by the algorithm is implemented as series of small conductance changes Δg_{min} per pulse coincidence. As a result, the weight update happens as a series of coincidence events each triggering a conductance increment (or decrement) and the expected number of coincidences is

$$\begin{aligned}&\mathbb{E}(\#\ of\ pulse\ coincidences\ at\ i^{\text{th}}\ column\ and\ j^{\text{th}}\ row)\\ &= \frac{\eta(x_i \times \delta_j)}{\mathrm{K}\Delta g_{min}}\end{aligned} \tag{3}$$

where $\mathrm{K}\Delta g_{min} \triangleq \Delta w_{min}$ is the expected weight change due to a single coincidence event. We note that the pulses generated at the peripheral are applied to all RPU devices among the column (or the row), therefore stochastic translators can assume a single Δg_{min} (or equivalently Δw_{min}) value for the whole array when the pulse probabilities are calculated to result in the desired weight change at each RPU. However, we show next the actual changes triggered by each RPU device per pulse coincidence Δg_{ij} may differ from the Δg_{min}, and this mismatch will create artifacts in the SGD algorithm, which prevent its proper convergence.

Expected vs. Actual Weight Update

Using the formula of the expected number of pulse coincidences from Eq. 3, the actual algorithmic weight change caused by the update cycle performed by the RPU devices can be derived as

$$\Delta w_{ij,actual} = \Delta w_{ij} \begin{cases} \dfrac{\Delta g^p_{ij}\left(g_{ij}\right)}{\Delta g_{min}} & if\ \Delta w_{ij} < 0 \\ \dfrac{\Delta g^n_{ij}\left(g_{ij}\right)}{\Delta g_{min}} & if\ \Delta w_{ij} > 0 \end{cases} \tag{4}$$

where Δg^p_{ij} and Δg^n_{ij} are the actual device responses for the incremental conductance changes for positive and negative stimuli at the coincidence event. They may also be functions of the current device conductance g_{ij}. We assume that the update pulses are applied only to the first set of RPU devices and the reference devices are kept constant. This requires a bi-directional switching RPU device as we discuss in detail later. However, to enable both positive and negative conductance changes the polarity of the pulses are switched during the update cycle and hence there exists two branches for each device used for the updates. Using Eq. 4 in Eq. 1 results in an actual update rule implemented by the RPU devices

$$\begin{aligned}w_{ij} \leftarrow w_{ij} &- \eta \Delta w_{ij} \left[\frac{\Delta g^n_{ij}(g_{ij}) + \Delta g^p_{ij}(g_{ij})}{2\Delta g_{min}} \right] \\ &- \eta \left|\Delta w_{ij}\right| \left[\frac{\Delta g^n_{ij}(g_{ij}) - \Delta g^p_{ij}(g_{ij})}{2\ \Delta g_{min}} \right]\end{aligned} \tag{5}$$

that captures the deviation of the expected device conductance changes from the actual ones realized by the RPU devices. It can be interpreted as separating the even and odd part of the RPU switching behavior. Here we emphasize again that Δg_{min} is the single value expected by the periphery during pulse generation, whereas Δg^p_{ij} (or Δg^n_{ij}) are the actual changes triggered by each RPU device. Since the pulses generated at the periphery are common for the whole array (columns and rows) it is impossible to compensate for the mismatch between Δg_{min} and Δg^p_{ij} (or Δg^n_{ij}) at the periphery as each RPU device has a different Δg^p_{ij} (or Δg^n_{ij}) value due to device-to-device variability. Without loss of generality Eq. 5 can be rewritten as

$$w_{ij} \leftarrow w_{ij} - \eta \Delta w_{ij} F_{ij}(w_{ij}) - \eta \left|\Delta w_{ij}\right| G_{ij}(w_{ij}) \tag{6}$$

where $F_{ij}(w_{ij})$ and $G_{ij}(w_{ij})$ are the symmetric (additive) and antisymmetric (subtractive) combinations of the positive and negative update branches parametrized using the weight values corresponding i^{th} column and j^{th} row. Note that the functions F_{ij} and G_{ij} can generally be functions of the current weight value w_{ij} as well as vary from one cross-point to another due to device-to-device variability. Although we used the stochastic pulsing scheme for the derivation of Eq. 5 and 6, the equations are general and do not depend on the underlying pulse implementations

up to some rounding errors. **Table 1** compares the desired SGD update rule (Eq. 1) to the hardware induced update rule (Eq. 6), that has contributions from the device switching characteristics.

To understand the significance of the hardware induced update rule, the behavior of Eq. 8 is described below for three different device switching characteristics, as illustrated in **Figure 1**. For the first device, that changes the conductance in a linear fashion and has the same value for the positive and negative branches, the hardware induced update rule simplifies back to the desired SGD update rule as $F(w) = 1$ and $G(w) = 0$. This is the case usually considered as the ideal device behavior required for good convergence. For the second device, that changes conductance in a non-linear but symmetric fashion for both up and down branches, then again $G(w)$ term drops, and Eq. 8 simplifies to a form $w \leftarrow w - \eta \Delta w F(w)$, where more specifically $F(w) = 1 - 1.66w$ for the example device illustrated in **Figure 1B**. Although this update rule is different from the original SGD update rule, the existence of $F(w)$ only modifies the effective learning rate and therefore does not affect the convergence. Indeed, empirically it is shown that RPU devices need to have only symmetrical switching characteristics and the linearity is not required for good convergence (Agarwal et al., 2016a; Gokmen and Vlasov, 2016; Gokmen et al., 2017, 2018). Only if updates are performed on two separate devices that change their conductances monotonically, such as PCM devices with one-sided switching, then pair-wise matching and linearity are mandated to satisfy the symmetry requirement (Haensch et al., 2019). Finally, for the third device with non-symmetric device switching characteristics the hardware induced update rule becomes $w \leftarrow w - \eta \Delta w F(w) - \eta |\Delta w| G(w)$. Since $|\Delta w|$ can only be non-negative, the last term is solely dictated by the functional form of $G(w)$ and will act as an unintentional cost term introduced into optimization objective by the underlying hardware behavior. For the device illustrated in **Figure 1C**, where $F(w) = 0.65 - 0.54w$ and $G(w) = 1.12w$, the hardware induced update rule becomes $w \leftarrow w - \eta F(w) \Delta w - \eta |\Delta w| (1.12w)$ which corresponds to an optimization objection that is a combination of the original problem with an additional quadratic cost term w^2. This is similar to adding ℓ_2 regularization term into the optimization objective but unfortunately its magnitude cannot be controlled and more strictly its amplitude is proportional to the updates $|\Delta w|$. This creates a competition between the original optimization objective of the neural network and an internal cost term due to device characteristic; providing theoretical justification to the empirically observed poor training results obtained for non-symmetric RPU devices.

Note that even for the non-symmetric device illustrated in **Figure 1C** there exists a single point (conductance value) at which the strengths of the conductance increment and decrement are equal. This point is called the symmetry point of the updated device and it may correspond to any weight value (not necessary to zero as illustrated in **Figure 1C**) due to the device-to-device variations. As we show next there exists a method, *symmetry point shifting* technique (Kim H. et al., 2019), that can guarantee that the symmetry point of the updated device matches the conductance of the corresponding reference device and hence satisfy the condition $G_{ij}\left(w_{ij} = 0\right) \cong 0$ for all elements of the matrix. Note that the strengths of conductance increment and decrement are equal at the symmetry point and therefore $G\left(\textit{at the symmtery point}\right) = 0$ by definition. We will show below that this property is accomplished by copying the symmetry point of the active device to its reference. However, the behavior of $G_{ij}\left(w_{ij}\right)$ away from zero is still dictated by the updated device characteristics and for actual hardware implementations of RPU devices, each device would show different $G_{ij}\left(w_{ij}\right)$ characteristics due to device variability. Combination of device variability and conductance state dependent updates makes it impossible to compensate for this non-symmetric behavior for individual devices without breaking the parallel nature of the array operations. However, the "Tiki-Taka" algorithm, as we describe below, eliminates the undesired effects of the device asymmetry for realistic RPU devices without breaking the array parallelism during training.

TABLE 1 | Summary of the update rules.

Desired SGD Update Rule		*Hardware Induced Update Rule*	
$w \leftarrow w - \eta \Delta w$	(7)	$w \leftarrow w - \eta \Delta w F(w) - \eta \lvert \Delta w \rvert G(w)$	(8)

All sub-indexes corresponding to i^{th} column, j^{th} row and the layer index is dropped for simplicity.

Symmetry Point Shifting Technique

The first step of the symmetry point shifting techniques is to apply a sequence of alternating (positive and negative) update pulses to the whole array all in parallel to all columns and all rows. In an alternating pulse sequence, the two consecutive pulses eliminate the $\eta \Delta w_{ij} F_{ij}(w_{ij})$ term from Eq. 8 and the dynamics of the whole array is dictated by the individual $G_{ij}\left(w_{ij}\right)$ terms. The behavior of $G_{ij}\left(w_{ij}\right)$ is expected to be different for each device due to the device variability and initial conductance variations, however, after sufficiently long sequence of pulses is applied, at steady state all elements are expected to converge to a point where $G_{ij}\left(w_{ij}\right) \cong 0$, although the corresponding weight value is not necessarily at zero, $w_{ij} \neq 0$.

This behavior is expected from any device where dependence of conductance increments (and decrements) on current conductance value can be described by a single curve (bi-directional switching behavior), such as the device illustrated in **Figure 1C**. As shown in **Figure 1C**, if the device conductance is smaller than the symmetry point ($g_{ij} < s_{ij}$) then the conductance increments are stronger than the decrements ($\Delta g^p_{ij} > \Delta g^n_{ij}$), and similarly, if the device conductance is larger than the symmetry point ($g_{ij} < s_{ij}$) then the conductance decrements are stronger than the increments ($\Delta g^p_{ij} > \Delta g^n_{ij}$). Therefore, independent of the initial conductance value, this alternating pulse sequence pushes the device conductance toward the symmetry point, as illustrated in **Figure 2**. Note that a convergence toward the symmetry point is expected independent of the functional form the conductance increments and decrements as long as there exists a conductance value where the strengths of conductance increments and decrements are equal in magnitude. Indeed, symmetry point measurement and shifting technique is

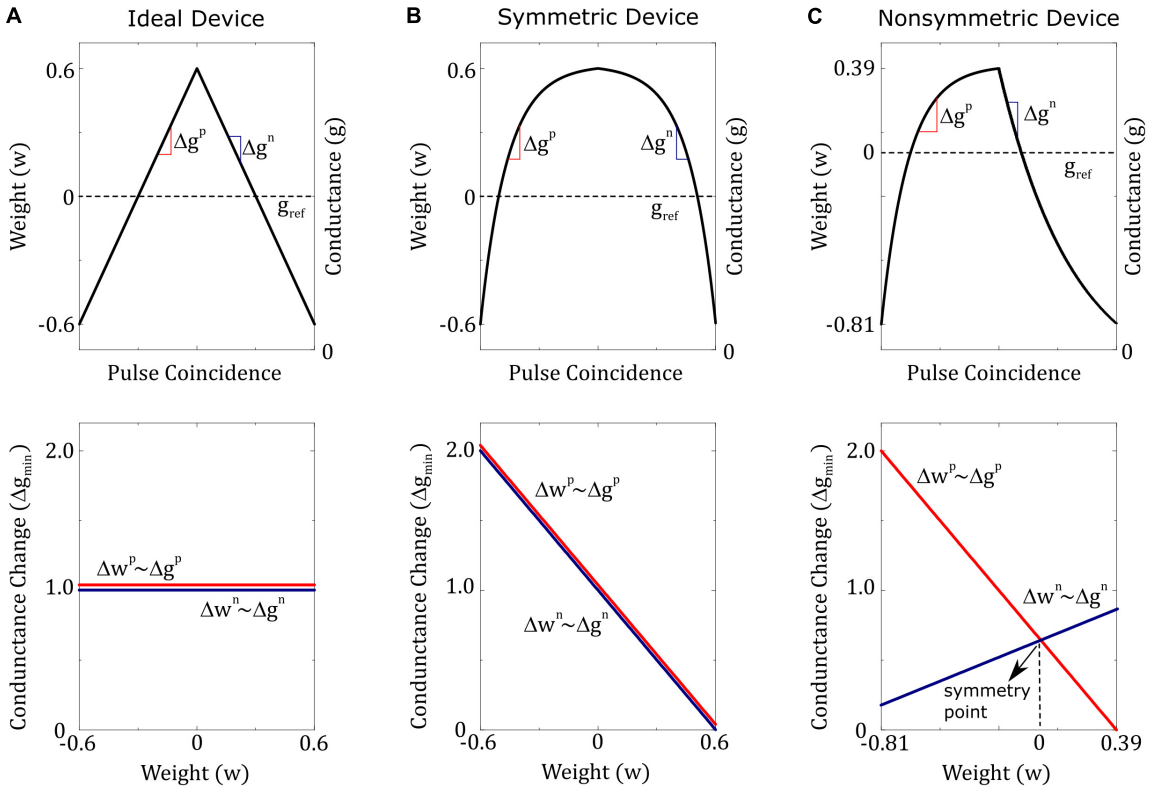


FIGURE 1 | Three different device switching characteristics are illustrated. **(A)** Ideal device: conductance increments and decrements are equal in size and do not depend on device conductance. **(B)** Symmetric device: conductance increments and decrements are equal in strength, but both have a dependence on device conductance. **(C)** Non-symmetric device: conductance increments and decrements are not equal in strength and both have different dependencies on device conductance. However, there exists a single point (conductance value) at which the strengths of the conductance increment and decrement are equal. This point is called the symmetry point and for the illustrated example matches the reference device conductance and hence happens at $w = 0$.

experimentally illustrated for RRAM (Kim H. et al., 2019) and ECRAM (Kim S. et al., 2019) devices, and this general behavior is expected for most physically plausible RPU devices. For instance, it is not realistic to expect alternative pulse sequence to give divergent conductance behavior, and instead, two consecutive pulses would push the conductance of the updated device toward the symmetry point s_{ij} at which the up and down conductance changes are equal in strength and satisfy $\Delta g_{ij}^{p}\left(s_{ij}\right) = \Delta g_{ij}^{n}\left(s_{ij}\right)$. However, it is not always guaranteed that all devices would have a symmetry point. For instance, PCM devices show only one-sided incremental switching (SET) behavior whereas a single RESET pulse completely switches the device back to the high resistance state (Burr et al., 2017; Haensch et al., 2019). Therefore, for one-sided devices, such as PCM, either the symmetry point cannot be defined, or it can be defined at the conductance extremum (RESET conductance) making it unfit for the "Tiki-Taka" algorithm. However, for device arrays composed of RRAM, ECRAM or similarly behaving two-sided switching devices (Haensch et al., 2019), the alternating pulse sequence would bring the conductance of each updated RPU device close to its symmetry point s_{ij} for the whole array. After this initial alternating pulse sequence is applied, as a second and last step these conductance values s_{ij} are transferred to the corresponding reference devices so that $g_{ij,ref} \cong s_{ij}$ and hence $G_{ij}\left(w_{ij} = 0\right) \cong 0$ for all elements in the matrix. Since this is a onetime cost, the conductance transfer can be performed iteratively in a closed loop fashion to overcome hardware limitations.

"Tiki-Taka" Algorithm

In "Tiki-Taka," each weight matrix of the neural network is represented by a linear combination of two matrices

$$W = \gamma A + C \tag{9}$$

where A is the first matrix, C is the second matrix and γ is a scalar factor that controls the mixing of the two matrices. The elements of A and C matrices, a_{ij} and c_{ij}, respectively, are also encoded by a pair of devices and we use upper left superscripts a and c consistently to refer to the properties of the elements (and devices) in A and C. For instance, ${}^{a}g_{ij}$ and ${}^{c}g_{ij}$ denote the conductance values stored on devices used for updates, and similarly ${}^{a}g_{ij,ref}$ and ${}^{c}g_{ij,ref}$ denote the conductance values stored on devices used as references corresponding to i^{th} column and j^{th} row. For "Tiki-Taka" to be successful, important criteria, ${}^{a}G_{ij}\left(a_{ij} = 0\right) \cong 0$ for all elements of A, must be realized by the hardware. Therefore, we assume that the symmetry point shifting

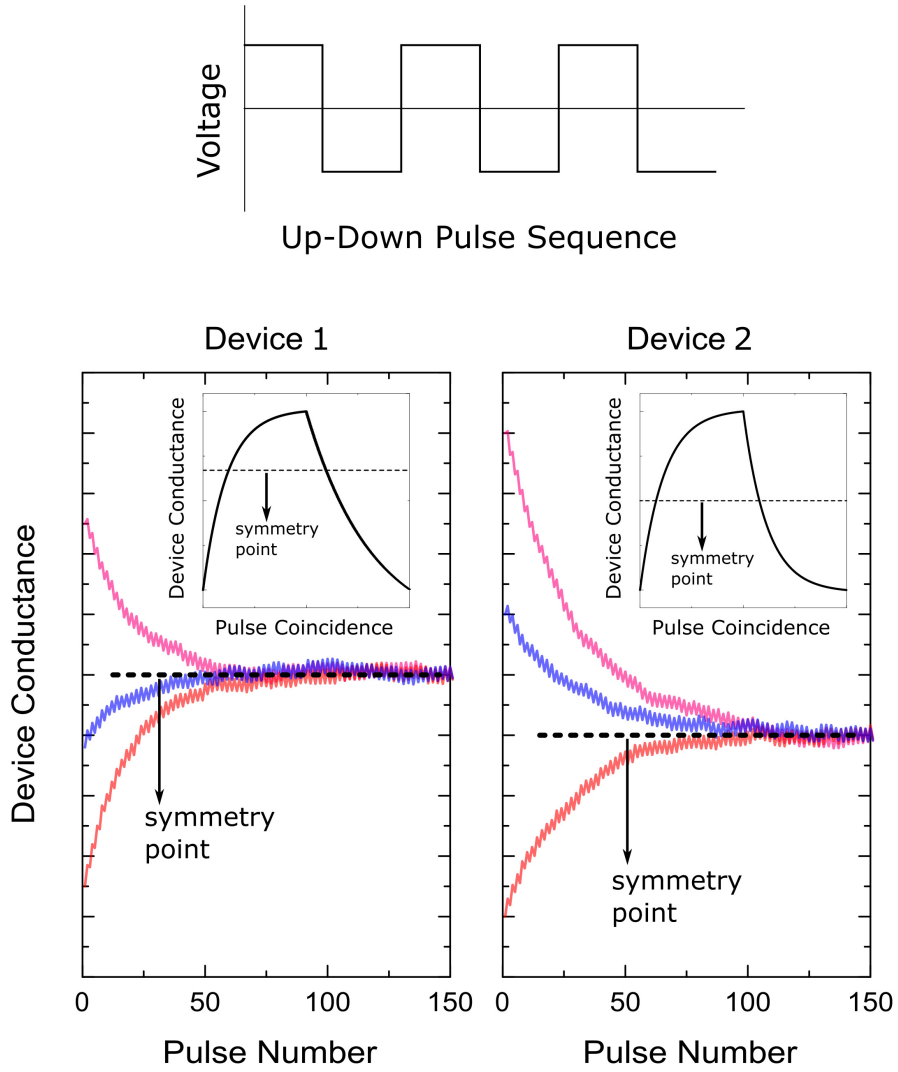


FIGURE 2 | Illustration of the symmetry point shifting technique. Response of two separate devices to the alternating (up and down) pulse sequence starting from different initial conductance values.

technique is applied to A before starting the training procedure described below.

Training Procedure

To simplify the exposition of the key idea, we shall omit for now the non-linear activation functions. In its most general form, the weight matrix W is a linear combination of two matrices A and C and γ is a scalar factor that controls the mixing of the two matrices. During training, the weight updates are accumulated on A that has symmetric behavior around the zero point, and then moved to C. The operations performed during "Tiki-Taka" is summarized in **Table 2** along with the ones performed during the SGD\BP algorithm for comparison.

The conventional SGD\BP algorithm is composed of three cycles: (1) forward, (2) backward and (3) weight update; whereas for "Tiki-Taka" there exist five cycles: (1) forward, (2) backward, (3) update A, (4) forward A, and (5) update C. The first two (forward and backward) cycles of the "Tiki-Taka" algorithm are identical to the ones in SGD\BP, as "Tiki-Taka" also uses the conventional BP algorithm to calculate the gradients. However, instead of using a single weight matrix, the linear combination of two matrices is used to perform the forward and backward computations. The third (update A) cycle is identical to the weight update cycle of the SGD\BP algorithm and the update operation on A is performed using the outer product of the two vectors that are used in the forward and the backward cycles. These three cycles are repeated ns times before the fourth and fifth cycles of the "Tiki-Taka" algorithm are performed. In the fourth (forward A) cycle a vector-matrix multiplication is performed on A using an input vector u_t. We discuss different choices for u_t later but in its most simple form u_t is a single column of an identity matrix (a one-hot encoded vector) where for each artificial time step a new column is used in a cyclic fashion and the sub index t denotes that time index. This operation effectively

TABLE 2 | Operations for SGD\BP algorithm and "Tiki-Taka" algorithm on a single layer.

SGD\BP Algorithm	"Tiki-Taka" Algorithm
	$k = 0$
	$t = 0$
for each data in training dataset	for each data in training dataset
{	{
	$k = k + 1$
(1) $y = Wx$	**(1)** $y = (\gamma A + C)x$
(2) $z = W^{\top}\delta$	**(2)** $z = (\gamma A + C)^{\top}\delta$
(3) $W \leftarrow W + \eta\ (\delta \otimes x)$	**(3)** $A \leftarrow A - \eta\ (\delta \otimes x)$
	if ($k == ns$)
	{
	$k = 0$
	$t = t + 1$
	(4) $v = Au_t$
	(5) $C \leftarrow C + \lambda\ (v \otimes u_t)$
	}
}	}

For simplicity, only the operations performed on the weight matrices are shown.

reads a single column of A into the resulting vector v. In the fifth (update C) cycle, C is updated using the outer product of the same two vectors u_t and v from the fourth cycle. This update operation changes only the elements corresponding to a single column of C proportional to the values stored in A, and λ is the learning rate used for updating C. Note that since different u_t is used at different time steps eventually all elements of C get updated.

In this algorithm the updates performed on A accumulate the gradients from different data examples and therefore A is actively used. In contrast, the updates on C are very sparse and only a single column of C is updated while the remaining elements are kept constant. Therefore, there is a big difference in the update frequency of the elements in these two matrices and C learns on a much slower time scale only using the information accumulated on A. As described above, the gradient accumulation happening on A has artifacts due to the hardware induced update rule, however, thanks to the symmetry point shifting technique the sign of the average gradient information is very likely to be correct (up to a limit that is dictated by the hardware noise). For instance, any kind of randomness in updates due to the random sampling of the data examples pushes the elements of A toward zero while the true average gradients push them away from zero. Therefore, when the elements of A are read, all elements are likely to have the correct sign information of the accumulated gradients although the amplitudes are probably underestimated. This information is then transferred to C which effectively grows the total weight in the correct direction that minimizes the objective function. At the end of the training process, at the steady state (independent of the choice of γ value), we expect the elements of C would get very close to a point in space, $c_{ij} \cong w_{ij,opt}$, where the original objective function is in its local minima and the elements of A would be close to zero, $a_{ij} \cong 0$. This is indeed a stable point for the coupled system to the first order. When $c_{ij} \cong w_{ij,opt}$, by definition the average gradients from different data samples are close to zero, $\langle \Delta w_{ij} \rangle \cong 0$, but since $\langle |\Delta w_{ij}| \rangle$ is always finite due to stochastic data sampling the hardware induced update rule for A drives all elements toward zero, $a_{ij} \cong 0$, which in return diminishes the updates on C. We note that the hardware induced update rule for C also has artifacts that repels c_{ij} away from $w_{ij,opt}$, however, these updates are sparse and happens across much slower time scales, rendering such artifacts negligible.

In contrast, for the SGD\BP algorithm even if the weights somehow get close to an optimum point corresponding to a local minimum, $w_{ij} \cong w_{ij,opt}$, the randomness in the updates pushes the weights away from the optimum point toward the symmetry point (or toward zero if symmetry point shifting technique is applied). Using the same arguments presented above, at a local minimum not only the average gradients from different data samples are close to zero, $\langle \Delta w_{ij} \rangle \cong 0$, but also it is guaranteed that $\langle |\Delta w_{ij}| \rangle > 0$ due to random data sampling. Therefore, at an optimum point the hardware induced updates are totally predominated by the non-symmetric device switching characteristics, $G_{ij}\left(w_{ij}\right)$, and therefore optimum points are not stable points for the SGD\BP algorithm running on RPU hardware. The only stable points that SGD\BP can find are the ones that has a tension between original optimization objective and the internal cost term due to device asymmetry, such that $\langle \Delta w_{ij} \rangle F_{ij}\left(w_{ij}\right) \cong -\langle |\Delta w_{ij}| \rangle G_{ij}\left(w_{ij}\right)$, and therefore give non-satisfactory training results. The artifacts of the hardware induced update rule for the SGD\BP algorithm are mitigated using the "Tiki-Taka" algorithm where the optimum points of the original objective function are turned into stable points for the coupled dynamical system. Therefore, this new training approach is expected to give superior results compared to the SGD\BP algorithm when running on RPU hardware.

RESULTS

To test the validity of the proposed "Tiki-Taka" algorithm we performed DNN training simulations on three different network architectures: (1) **FCN-MNIST** – a fully connected network trained on MNIST dataset, (2) **CNN-MNIST** – LeNet5 like convolutional neural network trained on MNIST dataset, and (3) **LSTM-WP** – a doubly stacked LSTM network trained on Leo Tolstoy's War and Peace (WP) novel. For all these three networks, the training performance of the SGD\BP algorithm with realistic RPU device specifications was studied carefully in previous publications (Gokmen and Vlasov, 2016; Gokmen et al., 2017, 2018). It was shown that a very tight symmetry requirement is needed to achieve training accuracies comparable to the ones achieved with high precision floating point numbers. Here, we use the same network settings from those publications, such as the activations, the layer sizes and the layer mappings onto the arrays; and follow a similar methodology for the RPU models, such that *they capture the device-to-device and cycle-to-cycle variations of the RPU devices as well as the non-idealities of the peripherical circuitry driving the arrays*. However, we emphasize that different from those studies, here we use a significantly non-symmetric device switching behavior as described below to evaluate the performance of the "Tiki-Taka" algorithm.

RPU Baseline Model

The RPU-baseline model uses the stochastic update scheme in which the numbers that are encoded from the periphery (x_i and δ_j) are implemented as stochastic bit streams. Each RPU device then performs a stochastic multiplication via simple coincidence detection. In our simulation tool, each coincidence event triggers an increment or decrement in the corresponding device conductance using a device switching characteristics. As a baseline model we use a *non-symmetric* device behavior, similar to one shown in **Figure 1C**, and this behavior introduces a weight update (conductance change) that depends on the current weight value (current device conductance) and the direction of the update. The dependence of the incremental weight updates for both branches are assumed to be linear: for the positive branch $\Delta w^p_{min}(w) = \Delta w_{min0}(1 - slope^p \times w)$ and for the negative branch $\Delta w^n_{min}(w) = \Delta w_{min0}(1 + slope^n \times w)$, where $slope^p$ and $slope^n$ are the slopes that control the dependence of the weight changes on the current weight values, and Δw_{min0} is the weight change due to a single coincidence event at the symmetry point corresponding to the zero weight value. In the RPU-baseline model in order to account for the device-to-device variability for each RPU device there exists three unique parameters that are sampled independently from Gaussian distributions at the beginning of the training and then used throughout the training. The average values of Δw_{min0}, $slope^p$ and $slope^n$ are respectively 1×10^{-3}, 1.66 and 1.66 with standard deviations of 0.3×10^{-3}, 0.42 and 0.42. Therefore, in the baseline model it is likely to find a device (in one of the cross-points in one of the layers) similar to one illustrated in **Figure 1C** with $\Delta w_{min0} = 0.65 \times 10^{-3}$, $slope^p = 1.66$ and $slope^n = 0.58$ as all values are within 3σ values of the model. However, we emphasize that it is very unlikely to have two devices to have identical behaviors due to the device-to-device variability introduced by the sampling process. Moreover, to capture the cycle-to-cycle variations for each coincidence event an additional 30% Gaussian noise is introduced to Δw^p_{min} or Δw^n_{min} relative to their expected values for each device before incrementing or decrementing the corresponding weight value. In this model, the weight saturations, corresponding to conductance saturations, are automatically taken into account due to the weight dependent updates, and the weight values cannot grow bigger than $1/slope^p$ or smaller than $-1/slope^n$. For statistically the most likely device in the model this corresponds to weight bounds between −0.6 and 0.6 but note that for each device $slope^p$ and $slope^n$ are sampled independently and therefore they don't necessarily match and deviate from the nominal values. In the context of "Tiki-Taka," since we use a new set of random variables for each device in the model there is no correlation between the elements of A and C, and in this context the weight changes refer to the changes that occur in the elements of A and C. Note that the baseline model already implicitly assumes that the symmetry point shifting technique is applied before training as the expected weight changes for the positive and negative branches are equal in strength at the zero weight value, $\Delta w^n_{min}(w = 0) = \Delta w^p_{min}(w = 0)$ for each matrix element. Therefore, this model assumes that the symmetry point shifting technique is applied perfectly without any noise both to A and C, such that all reference device conductances are set to the symmetry point of the corresponding device used for updates. Later we relax this assumption to test the tolerance of "Tiki-Taka" to the symmetry point variations.

We emphasize that the chosen mean values for $slope^p = slope^n = 1.66$ that control the device asymmetry are the largest possible values that can be used without introducing any side effects. For instance, it is shown in Ref. (Gokmen and Vlasov, 2016) that the acceptable criteria for the weight bounds is between −0.6 and 0.6 and this range is consistently used in Refs. (Gokmen et al., 2017, 2018). Therefore, increasing the slope parameters beyond 1.66 would limit the weights into a range that is tighter than the acceptable criteria. Although "Tiki-Taka" is expected to deal with the device asymmetry, it cannot improve over these weight bounds resulting in side effects into the training. The chosen mean value for $slope^p = slope^n = 1.66$ is therefore the most aggressive asymmetric device switching behavior that can used without violating the other RPU specs derived in Ref. (Gokmen and Vlasov, 2016). However to show the robustness of the "Tiki-Taka" algorithm to various device switching behaviors, we also tested a case where the mean slopes for positive and negative branches are not matched over the population of devices and we call this model the Skewed-RPU model. Furthermore, in another model, the Quadratic-Skewed-RPU model, we added a quadratic term to the dependence of the weight increment (and decrement) on the current weight value. The switching behavior all these models including the device variabilities are illustrated in the top panel of **Figure 3**.

In addition to the non-idealities mentioned above, for any real hardware implementations of RPU arrays the results of the vector matrix multiplications will be noisy as well and this noise is considered by introducing an additive Gaussian noise, with zero mean and standard deviation of $\sigma = 0.06$. Moreover, the results of the vector-matrix multiplications are bounded to a value of $|\alpha| = 12$ to account for signal saturation. 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. Although the input signals going into the array and the output signals coming from the arrays are bounded, we use the noise management and the bound management techniques described in Ref. (Gokmen et al., 2017). We note that apart from the non-symmetric update behavior used for RPU devices, all other hardware constraints, such as variations, noise, limited resolutions and signal bounds, are identical to the ones used in publications (Gokmen and Vlasov, 2016; Gokmen et al., 2017, 2018).

Fully Connected Network on MNIST (FCN-MNIST)

The same network from Ref. (Gokmen and Vlasov, 2016), composed of fully connected layers with 784, 256, 128 and 10 neurons, is trained with the standard MNIST dataset composed of 60,000 training images. For hidden and output layers *sigmoid* and *softmax* activations are used, respectively. For the floating point (FP) model, training is performed with the SGD algorithm using a mini-batch size of unity and a fixed learning rate of

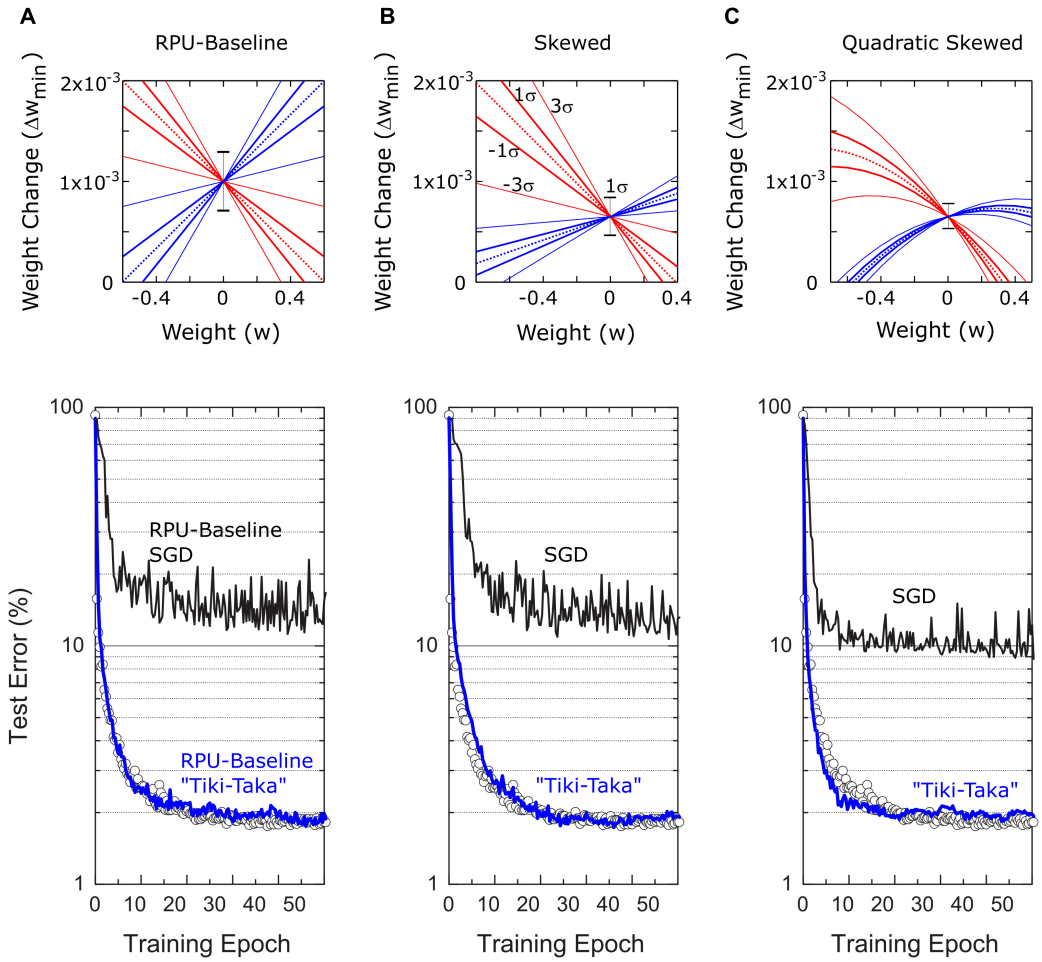


FIGURE 3 | Bottom panel shows the test errors of FCN on MNIST dataset for three different RPU models. In all figures at the bottom panel open white circles correspond to a model where the training is performed using high precision floating point (FP) numbers and the SGD algorithm. Black and blue lines correspond to the model where training is performed using the SGD algorithm and the "Tiki-Taka" algorithm, respectively. The panels **(A–C)** are for the three different RPU models where the device behavior used in each model are shown in the top panel. In all models each device behavior is sampled from a distribution of devices in order to account for the device-to-device variability. Statistically the most likely device is illustrated with the dotted line. Variations in the slopes are shown with continues lines for the 1σ and 3σ envelope and the 1σ variation in Δw_{min0} is shown by the error bar signal at $w = 0$. **(A)** RPU-baseline model assumes a linear relation between Δw_{min} and w. The up and down weight changes have an average slope of –1.66 and 1.66, respectively and both have 1σ variation of 0.42. Δw_{min0} is on average 1×10^{-3} with 1σ variation of 0.3×10^{-3}. **(B)** Different from the RPU-baseline model Skewed-RPU model has different average slopes for the up and down weight changes, –1.66 and 0.58, respectively and with 1σ variations of 0.42 and 0.15. Δw_{min0} is on average 0.65×10^{-3} with 1σ variation of 0.2×10^{-3}. **(C)** Different from the Skewed-RPU model, the Quadratic-Skewed model further introduces additional negative quadratic term to the dependence of Δw_{min} on w for both up and down changes.

$\eta = 0.01$. As shown by open symbols in **Figure 3**, the FP-model reaches a classification error of 2.0% on the test data after 50 epochs of training. The same SGD based training using the RPU-baseline model however results in about 15% test error that is significantly higher than the error achieved by the FP-model. This is indeed expected, as the device characteristics in the RPU-baseline model is highly non-symmetric and well above the acceptable device symmetry criteria described in Ref. (Gokmen and Vlasov, 2016). When the training is performed using "Tiki-Taka" for the same the RPU-baseline model, the test error drops back to a value close to 2%. This level of error is indistinguishable from the one achieved by the FP-model, and shows that in contrast to SGD, "Tiki-Taka" gives good training results even with highly non-symmetric devices. We emphasize that the "Tiki-Taka" algorithm is no more sensitive to other hardware issues (such as stochastic updates, limited number of steps, noise, ADC, and DAC) than the SGD algorithm as the RPU-model captures all those hardware constraints. Moreover, we also tested the validity of the "Tiki-Taka" algorithm for other device switching behaviors as illustrated in **Figures 3B,C**. Independent of the model used for training the "Tiki-Taka" algorithm gives results that are indistinguishable from the one achieved by the FP-model whereas the conventional SGD results in test errors much higher than the FP-model.

Note that different from a single learning rate used for SGD, there exist additional hyperparameters for the "Tiki-Taka"

algorithm, as illustrated in **Table 2**, namely the learning rates η for updating A and λ for updating C, the parameter γ controlling the mixing between A and C, the parameter ns controlling the period of updates performed on C, and the choice of u_t used for the forward A cycle. In order to present the best results possible by "Tiki-Taka" we performed training simulations at different hyperparameter settings. For the results presented in **Figure 3** the learning rates η and λ are fixed at 0.01 and 0.02, respectively, the parameters γ and ns are set to unity, and for u_t a fixed set of one-hot encoded vectors are used in a cyclic fashion. Additional training results studying the sensitivity of "Tiki-Taka" to some of these hyperparameters are shown in **Figure 4** for the FCN-MNIST problem using the RPU-baseline model.

In **Figure 4A**, the training results for cases $\gamma = 1$ and $\gamma = 0$ are shown while all other hyperparameters are unchanged. When $\gamma = 1$, the gradient updates happening on A are directly visible in the next cycle while calculating the gradients corresponding to another image. In contrast, for $\gamma = 0$, the learning can happen only indirectly thorough the updates performed on C that are sparse and less frequent. As described above we don't expect the steady state solutions to be any different between cases $\gamma = 1$ and $\gamma = 0$. Consistently, these two simulations show very similar training curves that improve significantly over the SGD training and reach test accuracies comparable to one achieved by the FP-model. However, the training process is governed by the dynamics of the coupled system and not by the equilibrium properties. Therefore, one may argue that the case $\gamma = 0$ learns slightly slower than case $\gamma = 1$ due to infrequent updates to explain the slight difference observed between the two cases for FCN-MNIST problem. Furthermore, these simulations also consider other possible hardware issues due to stochastic pulsing, variations as well as noise. Therefore, one may also expect case $\gamma = 0$ to perform better gradient estimations on A before transferring that information on to C and hence to show better training performance overall than case $\gamma = 1$. Although, there exist these interesting tradeoffs while choosing the mixing term, large improvements over the SGD training are consistently observed as illustrated in **Figure 4A** independent of γ .

In **Figure 4B**, we present the training results at various learning rates λ used to update C. It is clear that choosing a too large or a too small λ values are both undesirable. In the case of a too small λ, where $\lambda \rightarrow 0$, "Tiki-Taka" implements the original SGD algorithm (assuming $\gamma = 1$). In this setting only A learns using the same SGD algorithm on a weight space that is shifted by values stored in C, but, A cannot find good solutions because of the artifacts introduced by the hardware induced update rule. On the other extreme choosing a large λ results in an unstable behavior for the coupled system. In our simulations, we try a few λ values that are close to the learning rate η. We believe choosing similar learning rates keeps the updates on both systems comparable in strength and therefore the couple system can minimize the both objective functions simultaneously in a self-consistent fashion. The simulation results at three different λ values, 0.005, 0.02 and 0.04, are show in **Figure 4B**, all of which are achieving comparable test errors at the end of 50 epochs.

Other important hyperparameters of the "Tiki-Taka" algorithm are the u_t vectors used while doing a forward pass on A and ns, the period used to update C. Note that there exist three

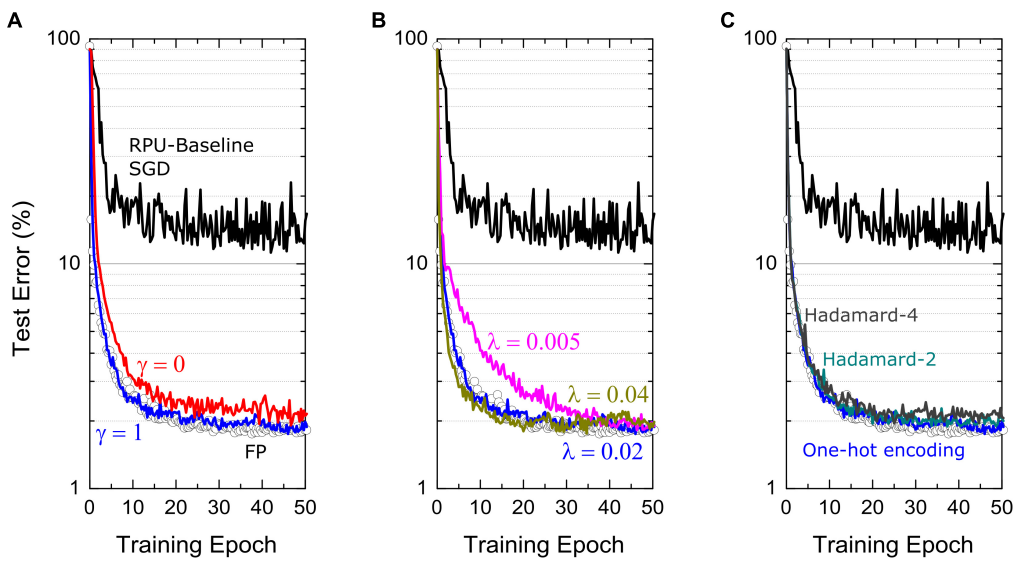


FIGURE 4 | Test error of FCN on MNIST dataset trained using Tiki-Taka algorithm and the RPU-baseline model at different hyperparameter settings. **(A)** The comparison of the training results at two different mixing terms, $\gamma = 1$ and $\gamma = 0$, corresponding to blue and red curves, respectively. **(B)** The comparison of the training results at three different learning rates on C matrix, $\lambda = 0.005$, 0.02 and 0.04, corresponding to magenta, blue and green curves, respectively. **(C)** The comparison of training results at three different choices of u_t vectors. Blue curves use one-hot encoded vectors in a cyclic fashion. Cyan and gray curves respectively use the vectors of Hadamard-2 and Hadamard-4 matrices in a cyclic fashion. For all figures, open white circles correspond to a model where the training is performed using high precision floating point numbers using the SGD algorithm, and black curves corresponds to the RPU-baseline model where the training is performed using the SGD algorithm.

weight matrices for FCN-MNIST, each having the dimensions of 256×785, 128×257, and 10×129 including the bias terms, where each weight matrix is represented by a pair of matrices A and C in the "Tiki-Taka" algorithm. Therefore, even when $ns = 1$ and a fixed set of one-hot encoded vectors is used for u_t, for the first layer it takes 785 images for all elements of C to get updated only once. Similarly, 257 and 129 images are required for the following layers. Larger ns values can be used to reduce the number of updates happening on C compared to A and for each layer ns can be chosen independently. Increasing the update period on C makes the artifacts of the hardware induced update even less important for C. However, note that the randomness of the updates on A tends to push the values of A toward zero due to the hardware induced update rule and hence slowly erases true gradient information accumulated on A from earlier time steps (images). Therefore, increasing ns beyond a certain value would not make the gradient accumulation more accurate and therefore there exists an upper bound on how large ns can be increased meaningfully. On the contrary, one may want to perform updates more often than the case supported by $ns = 1$ in order to use the hardware induced updates for regularization purposes. As illustrated before, the randomness in the updates attracts the corresponding matrices toward zero which has effects similar to the ℓ_2 regularization for some specific device switching characteristics, but the strength wasn't controllable for the SGD algorithm. In contrast, in "Tiki-Taka" we can control the strength of this term by performing updates on C using various u_t vectors. For instance, instead of using a set of one-hot encoded vectors in a cyclic fashion, the vectors of the normalized Hadamard-2 matrix padded with zeros, such as $\left[\frac{1}{2}\ \frac{1}{2}\ 0\ 0\cdots\right]\left[\frac{1}{2}\ -\frac{1}{2}\ 0\ 0\cdots\right]$, $\left[0\ 0\ \frac{1}{2}\ \frac{1}{2}\cdots\right]$ and so on, can be used in a cyclic fashion. This results in twice more updates on each element of C, yet a similar information is transferred from A to C. Because of the cancelations happening between two back-to-back updates on C, it would experience a stronger regularization toward zero thanks to the hardware induced update rule. Examples of the training performed using the vectors of the Hadamard-2 and Hadamard-4 matrices are shown in **Figure 4C**. These examples show that similar information can be transferred from A to C independent of the choice of u_t and yet the choice of u_t can be used as a knob to control the regularization term. We note that FCN-MNIST is a simple problem and does not overfit and hence does not require regularization, however, it is important to understand the consequences of different hyperparameters, so they can be tuned properly when they are really needed for large scale networks.

Convolutional Neural Network on MNIST (CNN-MNIST)

The CNN network used here is the same network from Ref. (Gokmen et al., 2017) and is composed of two convolutional and two fully connected layers. The first two convolutional layers use 5×5 kernels each having 16 and 32 kernels, respectively. Each convolutional layer is followed by a subsampling layer that implements the max pooling function over non-overlapping pooling windows of size 2×2. The output of the second pooling layer, consisting of 512 neuron activations, feeds into a fully connected layer consisting of 128 *tanh* neurons, which is then connected into a 10-way *softmax* output layer. Including the biases there exist four weight matrices with dimensions of 16×26 and 32×401 for the first two convolutional layers and, 128×513 and 10×129 for the following two fully connected layers.

We note that different from the fully-connected layers, convolutional layers have weight sharing that changes the vector operations performed on the weight matrices to matrix operations that are implemented as a series of vector operations on the arrays as described in Ref. (Gokmen et al., 2017). For "Tiki-Taka" this means that the first 3 cycles corresponding to the convolutional layers are now matrix operations and can be written as $y = (\gamma A + C)X$, $z = (\gamma A + C)^T D$, and $A \leftarrow A - \eta\ (D \otimes X)$, where X and D are the inputs and the errors feed into the weight matrices in the forward and backward directions, while the 4th and 5th cycles remains as before. The weight sharing factors for the two convolutional layers are 576 and 64, respectively. Therefore, when $ns = 1$ and a one-hot encoded vector is used as u_t, the A matrix of the first convolutional layer is updated 576 times before a single column of C is updated. Similarly, for the second convolutional layer A is updated 64 times, before C gets a single column update. All other operations remains identical for fully connected layers.

CNN-MNIST simulation results are shown in **Figure 5**. For the FP-model, trained with $\eta = 0.01$ and mini-batch of unity, the network achieves a test error of 0.8%. However, when RPU-baseline model is trained with the SGD algorithm, the test error is very high at around 8%. This large discrepancy from the FP-model significantly drops when the RPU-baseline model is trained with the "Tiki-Taka" algorithm, resulting in 1.0% test error. To understand the cause of this remaining 0.2% offset from the FP-model, we repeat the SGD training assuming a model with perfect symmetry ($slope^p = slope^n = 0$ for all devices) but with the remaining hardware constraints. This perfectly symmetric case trained with the SGD algorithm gives a test accuracy not any better than the one achieved by the non-symmetric case trained with "Tiki-Taka," suggesting that the remaining 0.2% discrepancy from the FP-model is due to other hardware constraints and not due to the device asymmetry. These results further highlight the power of this new training technique that compensates for non-symmetric device behavior at the algorithm level.

Sensitivity to Symmetry Point Variations

The simulations results presented so far assume the symmetry point shifting techniques is applied perfectly and hence the update strength for positive and negative branches are equal in strength at $w = 0$. It is clear that the symmetry point shifting technique cannot be perfect due to hardware limitations: To test the tolerance of the "Tiki-Taka" algorithm to the symmetry point variations, we performed training simulations by relaxing this assumption such that the condition $\Delta w^n_{min}(w_s) = \Delta w^p_{min}(w_s)$ happens at a weight value w_s that is different for each element in A and C. This is simply achieved by setting the reference device conductance different from the symmetry point of the

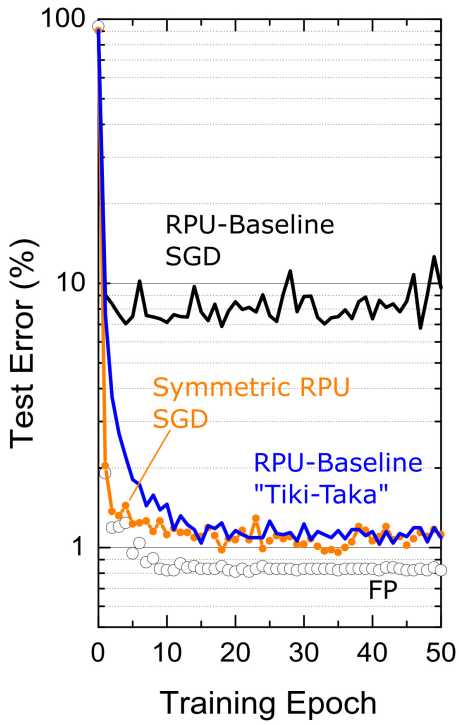


FIGURE 5 | Test error of CNN on MNIST dataset. Open white circles correspond to a model where the training is performed using high precision floating point (FP) numbers using the SGD algorithm. Black and blue lines correspond to the RPU-baseline model where training is performed using the SGD algorithm and the "Tiki-Taka" algorithm, respectively. The orange points\line correspond to SGD based training of a RPU model where all devices are perfectly symmetric while all other variations are identical to RPU-baseline model.

corresponding device used for the updates in the RPU baseline model. The simulation results for both FCN-MNIST and CNN-MNIST are presented in **Figure 6** using the modified RPU-baseline model, where w_s value for each device is sampled from a Gaussian distribution with zero mean but varying standard deviation, σ_{ws}. When the standard deviation of the distribution is $\sigma_{ws} \leq 0.01$, the training results are indistinguishable from the one achieved with no variations, and therefore, these results provide $\sigma_{ws} = 0.01$ as the acceptable threshold value for the symmetry point variations.

It is important that this acceptable threshold value of $\sigma_{ws} = 0.01$ is achieved by the symmetry point shifting technique. The symmetry point shifting technique may introduce two sources of noise while matching the reference device conductance to the symmetry point of the active (updated) device: (1) the noise in the converge to the symmetry point of the updated device and (2) the noise in the conductance transfer to the reference device. Given than the weight change due to a single coincidence event at the symmetry point is about $\Delta w_{min0} = 0.001$, which is 10 times smaller than the threshold $\sigma_{ws} = 0.01$, the alternating pulse sequence would result in convergence to the symmetry point that is much smaller than this acceptable threshold. Furthermore, using the ratio of this acceptable threshold value $\sigma_{ws} = 0.01$ to a nominal weight range of 1.2, this specification can be mapped to physical quantities. For instance, the matching of the reference device conductance to the symmetry point of the updated device must be accurate within a few percent compared to the whole conductance range. Therefore, after this initial converge, the stored conductance on the update device needs to be copied to the reference device within a few percent accuracy. Given that this is a onetime cost, this conductance transfer can be performed using a closed loop programming while achieving this required few perfect matching. Therefore, we emphasize that the acceptable threshold for symmetry point variations can be achieved with the symmetry point shifting technique and does not introduce any additional constraints to the required device specifications.

LSTM Network on War and Peace Dataset (LSTM-WP)

As a third example, the validity of the "Tiki-Taka" algorithm is tested on a more challenging LSTM network. This network is composed of 2 stacked LSTM blocks each with a hidden state size of 64 and it is identical to one studied in Ref. (Gokmen et al., 2018). As a dataset Leo Tolstoy's War and Peace (WP) novel is used and it is split into training and test sets as 2,933,246 and 325,000 characters with a total vocabulary of 87 characters. Following the same mapping described in Ref. (Gokmen et al., 2018) results in 3 different weight matrices with sizes $256 \times (64 + 87 + 1)$ and $256 \times (64 + 64 + 1)$ for the two LSTM blocks and a third matrix of size $87 \times (64 + 1)$ for the last fully connect layer before the *softmax* activation. Note that each matrix is now mapped to 2 separate matrices in "Tiki-Taka." The simulation results corresponding to the SGD algorithm and "Tiki-Taka" for various RPU models are shown in **Figure 7**. For all models the training is performed using fixed learning rate $\eta = 0.005$, mini-batch of unity and time unrolling steps of 100. Additionally, the hyperparameters of the "Tiki-Taka" algorithm are $\lambda = 0.005$, $\gamma = 1$, $ns = 5$, and for u_t one-hot encoded vectors are used in a cyclic fashion.

The simulation results presented in **Figure 7A** for LSTM-WP are qualitatively in good agreement with the ones presented for FCN-MNIST and CNN-MNIST above. For instance, the RPU-baseline model trained with the SGD algorithm results in a test error (cross-entropy loss) significantly larger than the one achieved by the FP-model. However, the same RPU-baseline model performs much better when "Tiki-Taka" is used for training, further validating this new approach for training DNNs. The perfectly symmetric case trained with the SGD algorithm is also shown as a comparison, and interestingly, it shows quantitative differences compared to ones presented for FCN-MNIST and CNN-MNIST: First, the perfectly symmetric case trained with the SGD algorithm cannot reach the level of accuracy achieved by the FP-model. Second, the RPU-baseline model trained with "Tiki-Taka" cannot reach the level of accuracy achieved by the perfectly symmetric case trained with the SGD algorithm. The former is understandable as it is shown in Ref. (Gokmen et al., 2018) that LSTM networks are more challenging to train on crossbar arrays; and even for perfectly symmetric devices, FP model accuracies cannot

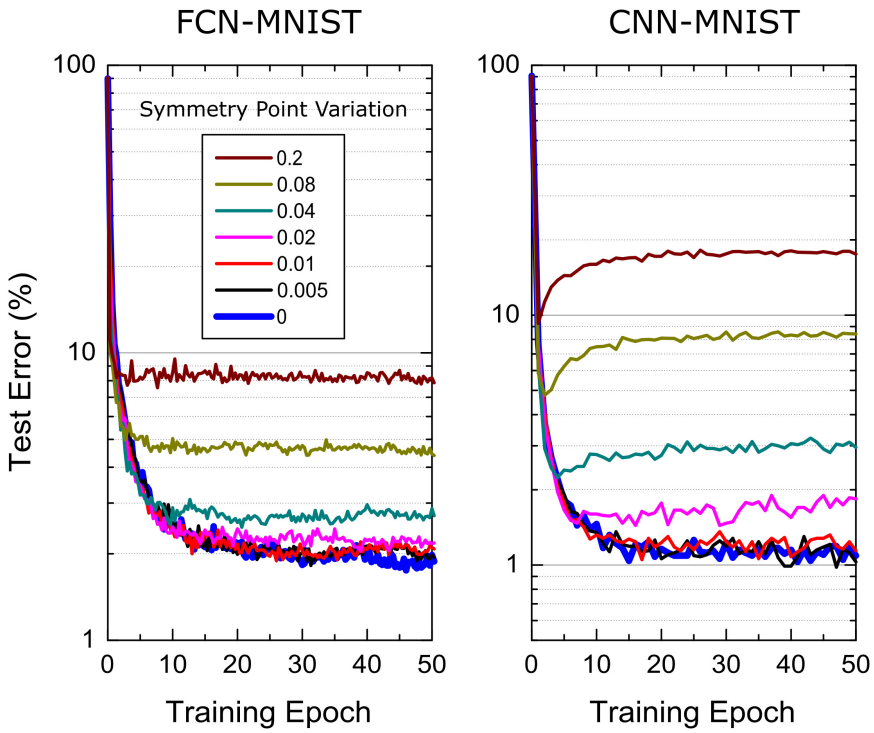


FIGURE 6 | Sensitivity of the "Tiki-Taka" algorithm to the symmetry point variations for FCN-MNIST and CNN-MNIST. The same hyperparameter settings from **Figures 3** and **5** are used for FCN-MNIST and CNN-MNIST, respectively. Different colors correspond to RPU-baseline models at different symmetry point variations.

be reached due to the limited number of states on RPU devices. Given that "Tiki-Taka" only addresses issues arising due to device asymmetry, it is not expected to reach the same level of accuracy of the FP model. It is only reasonable to expect it to perform at the same level of accuracy achieved by the perfectly symmetric case trained with the SGD algorithm, as all other hardware constraints are the same. Therefore, it is worth investigating the quantitative difference observed between the RPU-baseline model trained with the "Tiki-Taka" algorithm and the perfectly symmetric case trained with the SGD algorithm.

When the same RPU models with significant device asymmetry are used, it is clear that the training results using "Tiki-Taka" outperforms the results achieved by the SGD algorithm. This relaxes the acceptable device symmetry requirement by a large margin at equivalent accuracy, however, it is also obvious that reducing device asymmetry improves the training accuracy of the "Tiki-Taka" algorithm if the accuracy is already far from ideal to start with. Therefore, one can easily blame the very aggressive device asymmetry used in the RPU-baseline model to explain the quantitative difference observed between the "Tiki-Taka" algorithm and the perfectly symmetric case trained with the SGD algorithm. Trivially this deficit can be minimized by using a less asymmetric device switching characteristics (data not shown). However, there exist other hardware issues that may hinder the convergence, and more interestingly there may exist different tradeoffs between device switching characteristics and other hardware limitations for "Tiki-Taka" that are otherwise not applicable to the SGD algorithm.

To understand whether other existing hardware limitations play a role in the convergence of the "Tiki-Taka" algorithm we performed additional simulations using the same device model but assuming different hardware settings at the peripheral circuits. For instance, the simulation results presented in **Figure 7B** assume that the noise level for the vector-matrix multiplications is reduced by 10× from its original value in the RPU-baseline model. This reduction does not affect the performance of SGD based training both for the RPU-baseline model and the perfectly symmetric case. However, "Tiki-Taka" based training improves and the difference observed in **Figure 7A** between the RPU-baseline model trained with "Tiki-Taka" and the perfectly symmetric case trained with the SGD algorithm disappears in **Figure 7B**. More interestingly, in **Figure 7C** when we repeat the same experiment using an RPU-baseline model where only the noise spec of the forward A cycle in the "Tiki-Taka" algorithm is reduced by 10×, the training result remains unchanged and are very close to the perfectly symmetric case trained with the SGD algorithm. These simulation results show that the noise introduced during the transfer of the information accumulated on A to C may play a role in the convergence of the "Tiki-Taka" algorithm.

We emphasize that the hardware induced update rule for C also has artifacts that push the elements of C away from the optimum points at equilibrium. Although these artifacts are less important compared to the SGD algorithm, increased

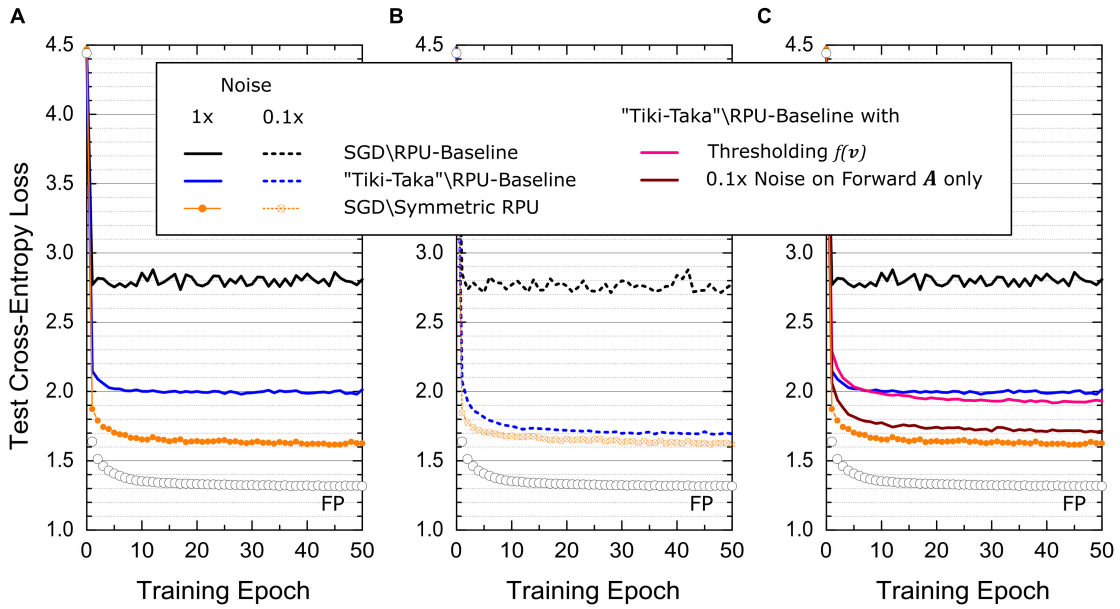


FIGURE 7 | Test cross-entropy error for LSTM network trained on WP dataset. Open white circles correspond to a model where the training is performed using high precision floating point (FP) numbers using the SGD algorithm. **(A)** Black and blue lines correspond to the RPU-baseline model where training is performed using the SGD algorithm and the "Tiki-Taka" algorithm, respectively. The orange points\line correspond to SGD based training of an RPU model where all devices have a perfectly symmetric switching characteristics while all other variations are identical to RPU-baseline model. **(B)** Shows the same training results from **(A)** at 10× reduced noise levels for all vector-matrix multiplications. **(C)** Pink curve corresponds to the RPU-baseline model trained using "Tiki-Taka" but the noise spec of the forward *A* cycle in the "Tiki-Taka" algorithm is reduced by 10×. The maroon curve also uses the RPU-baseline model trained using "Tiki-Taka" where the update *C* cycle of the "Tiki-Taka" algorithm is modified to $C \leftarrow C + \lambda\ (f(v) \otimes u_t)$, where $f(v)$ is a pointwise thresholding function that returns *v* only if $|v| > 0.06$ and otherwise zero. Black, blue lines and orange points\line are plotted again from **(A)** for comparison.

noise in updating *C* due to the randomness in reading *A* clearly impacted the training accuracy of the "Tiki-Taka" algorithm as illustrated above. Therefore, in order to further filter the updates happening on *C*, we changed the update *C* cycle of the "Tiki-Taka" algorithm as follows $C \leftarrow C + \lambda\ (f(v) \otimes u_t)$, where $f(v)$ is a pointwise thresholding function that returns *v* only if $|v| > T_v$ and otherwise zero. The simulation results of this modified "Tiki-Taka" algorithm for the RPU-baseline model is shown in **Figure 7C**, where T_v is set to the 1σ value of the read noise from the baseline model, $T_v = 0.06$. Although the improvement is not large, this filtering approach performs slightly better than the original unfiltered version and it suggests that there may exists other strategies that may outperform this simple thresholding-based filtering.

Speed, Area and Power Costs

Compared to the SGD algorithm, the "Tiki-Taka" algorithm introduces additional computations and requires additional hardware resources (crossbar arrays) to perform those computations, and therefore, their area, power and speed costs need to be sized properly.

The "Tiki-Taka" algorithm requires two sets of weight matrices for each layer hence it may increase the area requirement by a factor of 2. In this worst-case scenario *A* and *C* matrices can simply be allocated on two separate RPU tiles resulting in twice more area. However, if the RPU devices are integrated at the back-end-of-line (BEOL) in-between metal levels and stacked up as multiple layers, then this area cost can be eliminated. Given that the operations performed on *A* and *C* matrices are identical to the ones performed during the SGD algorithm, the same peripheral circuitry can be used to drive the lines corresponding to *A* and *C* matrices selectively to perform the forward, backward and update cycles in a time multiplex fashion. In this setting, the computations for the forward and backward cycles corresponding to $\gamma = 1$ case can also be realized by driving the lines of *A* and *C* matrices simultaneously while integrating the results from both matrices into the same capacitor. Also note that the update cycle on both matrices uses the common stochastic multiplication scheme. Therefore, 4 layers of stacked crossbar arrays can be operated as *A* and *C* matrices needed for "Tiki-Taka" without changing the peripheral circuitry design. Given that the same hardware specifications derived in Ref. (Gokmen and Vlasov, 2016) are sufficient for the "Tiki-Taka" algorithm, speed and power of each cycle remains identical to the ones performed in the SGD algorithm. However, "Tiki-Taka" introduced additional cycles to the training and its speed can be easily accounted by simply looking at the *ns* parameters used during training.

For the FCN-MNIST example $ns = 1$. This setting means that the "Tiki-Taka" algorithm repeatedly performs (1) forward, (2) backward, (3) update, (4) forward and (5) update cycles, 2 additional cycles compared to 3 cycles performed during the SGD algorithm. Since there are not any significant differences between the execution times of the forward, backward and

update cycles, the ratio of the wall clock times of "Tiki-Taka" to the SGD algorithm would be 5/3. Increasing ns further decreases this difference as illustrated for the LSTM-WP example where $ns = 5$. In this setting, for every 15 (3 by 5) cycles in the SGD algorithm, the "Tiki-Taka" algorithm introduces 2 additional cycles and hence it runs only ∼15% slower than the SGD algorithm. In contrast to the fully connected and LSTM networks where the weight sharing is uniform for all layers, the wall clock time of CNN networks are mainly dictated by the first convolutional layer with the largest weight sharing factor (Gokmen et al., 2017). For the CNN-MNIST example this weight sharing factor is 576. Therefore, even $ns = 1$ is used, for the first convolutional layer the "Tiki-Taka" algorithm introduces 2 additional cycles only after 3×576 cycles. This is a tiny difference and makes the run times of these two algorithms indistinguishable for CNN networks.

We note that there are additional computations that need to be performed outside the crossbar arrays, such as generation of u_t and calculation of $f(v)$. These computations can easily be handled by the digital units that are already responsible for calculating the activations and derivates used in the SGD algorithm. All these additional digital operations performed during the "Tiki-Taka" algorithm are local to the layer and are much simpler than the calculation of activations and derivates, therefore, their relative costs are no more than the relative costs already accounted above for the crossbar arrays.

DISCUSSION AND SUMMARY

We emphasize that throughout the manuscript we assumed that one crossbar array is used to perform the updates and another separate array is used as fixed reference conductances. The "Tiki-Taka" algorithm therefore assumes that the updated RPU devices change their conductance bidirectionally and therefore it is not directly applicable to one-sided switching devices such as PCM. The stability and convergence of the "Tiki-Taka" algorithm rely on the fact that the random sequence of updates on the A matrix eventually drive all elements of A toward zero. This is indeed achieved by the symmetry point shifting technique, and if this technique is generalized for one-sided switching devices then "Tiki-Taka" can also be used for devices like PCM. However, note that "Tiki-Taka" cannot eliminate the conductance saturation problem. PCM elements change their conductance gradually at one polarity (SET) and very abruptly at the opposite polarity (RESET). Therefore, only SET pulses are send either in the first or the second PCM array depending on the polarity of the weight updates. This eventually results in saturation in the conductance values and therefore require a serial reset operation. None of these complications arise for bidirectional devices and the "Tiki-Taka" algorithm can run with a very limited speed penalty using only parallel operations on the crossbar arrays.

We derived the hardware induced update rule in presence of non-symmetric devices and then showed its relevance to the SGD algorithm. For instance, for some specific device switching characteristics the hardware induced update rule looks similar to adding ℓ_2 regularization term in the optimization objective. However, the strength of this additional term is large and not controllable, and hence resulting in poor training results. Note for the $\gamma = 0$ case of "Tiki-Taka" the weights of the neural network are stored in C which are updated using the gradients accumulated on A. In this setting the hardware induced rules on A and C matrices show resemblance to the momentum-based SGD algorithm providing further intuition into the "Tiki-Taka" algorithm. However, careful investigation shows that the "Tiki-Taka" algorithm is not just an instance of the momentum-based SGD and may require further investigation. We presented an empirical approach for different network topologies that show that "Tiki-Taka" surpasses the conventional SGD in accuracy for relaxed symmetry requirements for analog cross-point devices. This is an important step forward to take advantage of analog cross points for deep learning training with currently available switching materials. As a rigorous mathematical theory explaining the successes of SGD in form of backpropagation is still elusive, an interesting avenue to proceed is to theoretically analyze the stability and convergence properties of the "Tiki-Taka" algorithm for some realistic device switching characteristics by applying, for example, the same techniques used for the stability analysis of discrete or continues dynamical coupled system. To extend this work to larger deep learning networks is a general task for the feasibility of analog cross-point arrays, not only restricted to the work presented here.

In summary, we proposed a new DNN training algorithm, so called "Tiki-Taka" algorithm, that uses a coupled system in order to simultaneously minimize the objective function of the original network of interest and the hidden cost term that is unintentionally introduced due to non-symmetric device switching characteristics. Training simulations performed on various network architectures show that even a very aggressive device asymmetry can be compensated by "Tiki-Taka" giving indistinguishable training results compared to ones achieved with the perfectly symmetric (ideal) devices. We emphasize that the asymmetry behavior used in our simulations and shown to be sufficient for "Tiki-Taka" is already experimentally observed by many device technologies but declared unsatisfactory due to asymmetry. Assuming other device specifications are within the tolerable margins, all those non-symmetric device technologies can now be used for deep learning applications, as the "Tiki-Taka" algorithm significantly relaxes the challenging symmetric switching criteria needed from the resistive cross-point devices.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article.

AUTHOR CONTRIBUTIONS

TG conceived the original idea and developed the methodology. TG and WH analyzed and interpreted the results, and drafted and revised the manuscript.

REFERENCES

Agarwal, S., Plimpton, S., Hughart, D., Hsia, A., Richter, I., Cox, J., et al. (2016a). "Resistive memory device requirements for a neural network accelerator," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: Institute of Electrical and Electronics Engineers Inc.).

Agarwal, S., Quach, T., Parekh, O., Hsia, A., DeBenedictis, E., James, C., et al. (2016b). Energy scaling advantages of resistive memory crossbar computation and its application to sparse coding. *Front. Neurosci.* 9:484. doi: 10.3389/fnins.2015.00484

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., di Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural netrowk training using analog memory. *Nature* 558, 60–67. doi: 10.1038/s41586-018-0180-5

Burr, G., Narayanan, P., Shelby, R., Sidler, S., Boybat, I., di Nolfo, C., et al. (2015). "Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: comparative performance analysis (accuracy, speed, and power)," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)* (Washington, DC: Institute of Electrical and Electronics Engineers Inc.).

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

Chen, P., Kadetotad, D., Xu, Z., Mohanty, A., Lin, B., Ye, J., et al. (2015a). "Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip," in *Proceedings of the Design, Automation and Test in Europe (DATE)* (Grenoble: Institute of Electrical and Electronics Engineers Inc.).

Chen, P., Lin, B., Wang, I., Hou, I., Ye, J., Vrudhula, S., et al. (2015b). "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (Austin, TX: Institute of Electrical and Electronics Engineers Inc.).

Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., and Feng, J. (2017). Dual path networks. *arXiv* [Preprint] arXiv:1707.01629 [cs.CV].

Coates, A., Huval, B., Wang, T., Wu, D., and Ng, A. (2013). "Deep learning with COTS HPC systems," in *Proceedings of the 30 th International Conference on Machine Learning (ICML)*, Atlanta, GA.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2012). Natural language processing (almost) from Scratch. *J. Mach. Learn. Res.* 12, 2493–2537.

Fuller, E., El Gabaly, F., Leonard, F., Agarwal, S., Plimpton, S., Jacobs-Gedrim, R., et al. (2017). Li-Ion synaptic transistor for low power analog computing. *Adv. Mater.* 29:1604310. doi: 10.1002/adma.201604310

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

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). "Deep learning with limited numerical precision," in *Proceedings of the 32 nd International Conference on Machine Learning*, Lille.

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. (2015a). Deep residual learning for image recognition. *arXiv* [Preprint] arXiv:1512.03385 [cs.CV].

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

Hinton, G., Deng, L., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., 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

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv* [Preprint] arXiv:1602.02410 [cs.CL].

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* [Preprint] arXiv:1907.10228 [cs.ET].

Kim, S., Todorov, T., Onen, M., Gokmen, T., Bishop, D., Solomon, P., et al. (2019). "Metal-oxide based, CMOS-compatible ECRAM for deep learning accelerator," in *Proceedings of the 65th International Electron Devices Meeting*, San Francisco, CA.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in Neural Information Processing Systems 25 (NIPS)*, Lake Tahoe, NV. 1097–1105.

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

Li, Y., Kim, S., Sun, X., Solomon, P., Gokmen, T., Tsai, H., et al. (2018). "Capacitor-based cross-point array for analog neural network with record symmetry and linearity," in *Proceedings of the IEEE Symposium on VLSI Technology*, Honolulu, HI.

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G., Likharev, A., and Strukov, D. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 61–64. doi: 10.1038/nature14441

Ravanelli, M., Brakel, P., Omologo, M., and Bengio, Y. (2017). A network of deep neural networks for distant speech recognition. *arXiv* [Preprint] arXiv:1703.08002v1 [cs.CL].

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik* 1, 36–45. doi: 10.1007/bf00293853

Tang, J., Bishop, D., Kim, S., Copel, M., Gokmen, T., Todorov, T., et al. (2018). "ECRAM as scalable synaptic cell for high-speed, low-power neuromorphic computing," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, 13.1.1–13.1.4.

Wu, Y. (2016). Google's neural machine translation system: bridging the gap between human and machine translation. *arXiv* [Preprint] arXiv:1609.08144 [cs.CL].

Xu, Z., Mohanty, A., Chen, P., Kadetotad, D., Lin, B., Ye, J., et al. (2014). Parallel programming of resistive cross-point array for synaptic plasticity. *Procedia Comput. Sci.* 41, 126–133. doi: 10.1016/j.procs.2014.11.094

Conflict of Interest: TG and WH were employed by the company IBM.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 11 February 2021
doi: 10.3389/fnins.2021.628254

Coupled VO_2 Oscillators Circuit as Analog First Layer Filter in Convolutional Neural Networks

*Elisabetta Corti[1], Joaquin Antonio Cornejo Jimenez[1], Kham M. Niang[2], John Robertson[2], Kirsten E. Moselund[1], Bernd Gotsmann[1], Adrian M. Ionescu[3] and Siegfried Karg[1]**

[1] IBM Research Zürich, Rüschlikon, Switzerland, [2] Department of Engineering, University of Cambridge, Cambridge, United Kingdom, [3] Nanoelectronic Devices Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

In this work we present an in-memory computing platform based on coupled VO_2 oscillators fabricated in a crossbar configuration on silicon. Compared to existing platforms, the crossbar configuration promises significant improvements in terms of area density and oscillation frequency. Further, the crossbar devices exhibit low variability and extended reliability, hence, enabling experiments on 4-coupled oscillator. We demonstrate the neuromorphic computing capabilities using the phase relation of the oscillators. As an application, we propose to replace digital filtering operation in a convolutional neural network with oscillating circuits. The concept is tested with a VGG13 architecture on the MNIST dataset, achieving performances of 95% in the recognition task.

Keywords: oscillatory neural network, vanadium dioxide, phase-encoding, convolutional neural networks, pattern recognition, relaxation oscillators, coupled oscillators

OPEN ACCESS

Edited by:
Peng Li,
University of California, Santa Barbara, United States

Reviewed by:
Yang Cindy Yi,
Virginia Tech, United States
Anup Das,
Drexel University, United States

****Correspondence:***
Siegfried Karg
sfk@zurich.ibm.com

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

Received: *11 November 2020*
Accepted: *06 January 2021*
Published: *11 February 2021*

Citation:
Corti E, Cornejo Jimenez JA, Niang KM, Robertson J, Moselund KE, Gotsmann B, Ionescu AM and Karg S (2021) Coupled VO_2 Oscillators Circuit as Analog First Layer Filter in Convolutional Neural Networks. Front. Neurosci. 15:628254. doi: 10.3389/fnins.2021.628254

INTRODUCTION

Convolutional Neural Networks (CNNs) are the architecture of choice to compute image recognition tasks. Widely used in commercial technology for their recognition accuracy, they are hindered in speed and power efficiency by the frequent access to the memory they need to perform to train a high number of parameters for each convolutional layer in deep networks (Sebastian et al., 2020). The development of neuro-inspired hardware holds the promise of accelerating these algorithms by exploiting in-memory computing concepts and limiting the number of accesses to the memory. A system of coupled oscillator, or Oscillatory Neural Network (ONN) can be used to store and recognize multiple patterns in compact networks. As described in Hoppensteadt and Izhikevich (1999) and Izhikevich (2000), systems of coupled oscillators lock in frequency and establish programmable phase relations that can be used for in-time computing applications. An ONN comprises a system of oscillators, in the role of neurons, connected to each other with synaptic weights, that represent the strength of the oscillators' coupling and the memory of the network. The ONN systems therefore rely on encoding and processing the information with time-delays in the circuits, rather than the amplitude of a signal, therefore being resilient to voltage noise and to scaled power supply.

Exploiting the associative memory capabilities of such networks, tasks as image recognition can be performed. Numerous works have simulated through mathematical and circuit simulations the digit pattern retrieval with different coupled oscillators technologies: (Jackson et al., 2015) shows

20-pixel digit recognition using transition metal oxides and resistive ram technology; (Nikonov et al., 2015; Liyanagedera et al., 2016), perform similar simulations on Spin Torque Oscillators (STOs); (Hölzel and Krischer, 2011) with Van der Pool oscillators. These works are based on storing patterns with $n \times m$ pixels into an ONN that comprises $n \times m$ oscillators. To perform the recognition, a distorted pattern of the same pixel size is fed to the ONN, and using the minimum phase attractor of the circuit, the right stored pattern is retrieved. The output is an $n \times m$ pixels image of the correct digit. Image classifications tasks, however, work quite differently. Taking as an example digit classification through a neural network, an image of $n \times m$ pixels if fed into the network. The network output is an 1×10 array containing the classification probabilities of that image. This classification operation is most commonly performed by convolutional neural networks, that process the image with a series of trainable convolutional filters in the first few layers and achieve recognition after some fully connected layers (**Figure 1**).

The link between convolutional neural networks and the associative memory capabilities of oscillatory neural networks has so far been explored, to our knowledge, only in Liu and Mukhopadhyay (2018), where an associative memory bank (Hopfield network) replaces the fully connected layers in CNNs. The associative memory is here used to perform a classification with a combination of unsupervised learning and transfer learning techniques. Even though the concept is very interesting and promising, the Hopfield network that this technique envisions comprises between 256 and 2,024 neurons. However, the physical demonstrations of oscillatory neural networks features maximum 100 oscillators as neurons with standard Phase Locked Loop or equivalent CMOS technology (Jackson et al., 2018). The technological challenge in the physical realization of large oscillatory neural networks resides in the complex dynamics of the oscillators' frequency and phase synchronization when the electrical components are affected by variability. This is even more true when the ONNs are built with novel oscillator technology, such as STOs or vanadium-dioxide (VO_2) oscillators (Romera et al., 2018; Raychowdhury et al., 2019), for which a maximum of 6 oscillators have been connected into a network.

Alternatively, it is suggested that the computing capabilities of small oscillator networks, with up to 10 nodes, can be efficiently exploited to do various image processing tasks, like graph coloring or image saliency processing (Cotter et al., 2014; Tsai et al., 2016). These previous works propose an ONN scheme in which the computation is based on the distance between the input image and the feature to be recognized. For example in Tsai et al. (2016) this distance is encoded in the difference in gate voltages between two transistors which bias a phase transition device. Another popular configuration encodes the distance between the input pattern and the feature to be recognized in a frequency shift between oscillators, which are connected by a fixed coupling. The distance between the two patterns is then calculated on the time the oscillators need to converge to the same frequency (Cotter et al., 2014; Nikonov et al., 2015; Zhang et al., 2019). The implementation of these concepts does not use the associative memory capabilities of the ONNs to store multiple patterns. Instead, to perform the distance measure, the circuit needs to be reconfigured each time a different feature needs to be recognized. In our work, we propose an implementation of small, fully connected networks, which exploit the associative memory capabilities of an Hopfield network. This allows to store the different features to be recognized in the same network, and enables the recognition of up to 5 different features within one computation performed by the same filter. In addition, in our work we provide the missing the link to show how the feature extraction performed by ONNs can be used for image classification tasks. We show the potentiality of the ONN technology for the realization of reconfigurable CNN in hardware, therefore bridging the gap between previous demonstration of ONN pattern retrieval and the industry-standard algorithms.

Among the new oscillator technologies, we concentrate our analysis on VO_2 oscillators, as they offer the advantage of realization of very compact oscillators, which can be easily coupled with standard electrical components, allowing for easy reconfigurability of the system (Parihar et al., 2015; Corti et al., 2018). VO_2 based oscillators also offer good scalability perspective and demonstrate operating voltage of less than 1 V and low power consumption ($\sim$20 μW per oscillator) (Shukla et al., 2016).

We exploit the feature extraction capabilities of small networks of VO_2 coupled oscillator to replace digital filters in CNNs (**Figure 1B**). We fabricate VO_2 oscillators on a Si platform, adopting a crossbar (CB) configuration with scaled device dimensions down to 70 nm. The CB devices exhibit improved variability and reliability compared to co-planar structures and enable the coupling of 4 oscillators. We demonstrate that such a 4-node ONN can memorize and perform 5 different filtering actions of a CNN in a single circuit. Simulations with a 3×3 ONN further show how the concept can be applied to replace digital filters in the first layer of a CNN with a VGG-13 inspired architecture and through the adoption of a transfer learning technique. The hybrid CNN-ONN platform has been tested on the MNIST algorithm reaching recognition performances up to 95%. As an outlook, we discuss the benchmark of this technology when extended to all the layers of a CNN, up to the fully connected layers, in comparison with existing hardware and conclude that ONNs might be used as fast and low-power inference machines.

MATERIALS AND METHODS

Device Fabrication

VO_2 is a phase change material that presents a volatile, temperature driven insulator to metal transition (IMT). The transition can be triggered by joule heating when a voltage is applied to a VO_2 device (Kim et al., 2010), and it is reversed when the voltage across the device is removed. Given its volatile phase-change characteristics, VO_2 cannot be used as memory element like chalcogenide-based phase change materials (PCM). However, the VO_2 phase transition can be instead

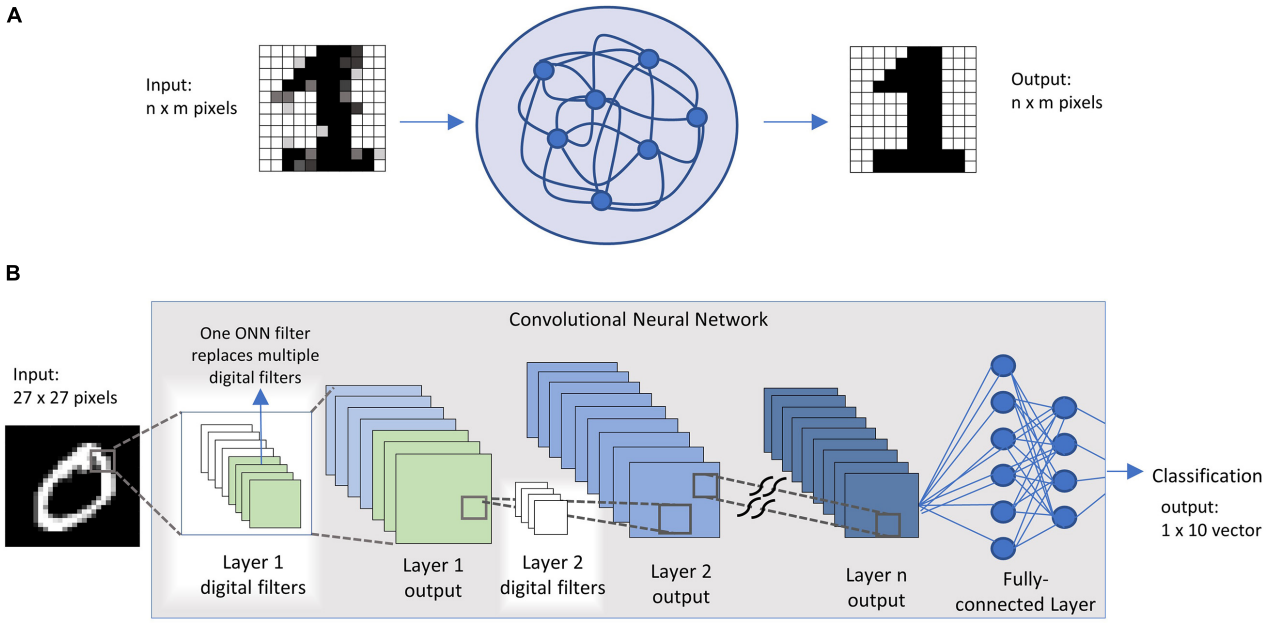


FIGURE 1 | (A) image inference process in an oscillatory neural network; **(B)** image classification through a convolutional neural network.

exploited to build very compact oscillators. Other materials have shown similar properties, for example tantalum oxide (Jackson et al., 2015; Sharma et al., 2015) or niobium oxide (Li et al., 2015), however, the near-room temperature phase transition of VO_2 and its proven high endurance up to 10^9 cycles (Shukla et al., 2015) make this material a most favorable choice of oscillators-based technology. VO_2 can be grown crystalline on TiO_2 substrates; however, when deposited on Si, the film forms grains of the average dimension of ∼50 nm (Premkumar et al., 2012). In the interest of future integration with electronics we have chosen to focus on integration on silicon in our work.

Figure 2 shows VO_2 devices fabricated in a CB geometry on a 4" Si wafer with a 1 μm thermal SiO_2 layer. Trenches are etched into the SiO_2 substrate and filled with Pt to provide the bottom electrode. Subsequently, an 80 nm thick VO_2 film is grown via atomic layer deposition and post-annealed, resulting in a policrystalline, granular film (Bai et al., 2020; Niang et al., 2020). Finally, top electrodes are formed using e-beam lithography and Pt evaporation. The smallest device area is 70 nm × 70 nm allowing a very compact design. The resistivity vs. temperature curve (RT) of a 250 nm × 250 nm device is shown in **Figure 3A** and exhibits an insulator-to-metal phase-transition with roughly a two-order of magnitude in resistance change. The step-like RT implies multi-grain transitions, as already shown in previous work (Ruzmetov et al., 2009; Takami et al., 2014; Corti et al., 2019). **Figure 3B** shows the insulator-to-metal and metal-to-insulator transition of an electrically activated device. A current source is used to control the current flowing in the device; a voltmeter is used to measure the voltage at each point. The IV characteristic of this device shows three different operating regions: a first region, in which the device is in its high resistance state, a negative differential resistance regime upon the phase change, and finally the low resistance region. A crossbar-geometry of VO_2 based-oscillators applications has previously been fabricated with point-probe contacts on TiN on Si, yielding record-speed oscillations performances of 9 MHz, almost an order of magnitude more than what demonstrated with planar structures (Mian et al., 2015). In another more recent work, the oscillating and coupling dynamics of such a vertical structure have been measured and modeled (Tobe et al., 2020). In this work, we further report that the cross-bar geometry yields a better reliability of the devices. In fact, in a previous work we discussed how coplanar devices needed a burn-in cycle to initialize the devices, which sometimes resulted in fatal irreversible changes in morphology (Corti et al., 2019). The crossbar devices do not need a burn-in cycle, improving reliability as virtually all the devices fabricated were able to produce oscillations. Compared to the planar VO_2 structure, the crossbar structure provides improved threshold voltage stability (device-to-device variability lowered from 20% to 10%) and resistance variability (from 10% to ∼5%). Compared to other demonstrations on silicon, the improved variability allows for coupling of more oscillator nodes, up to 4. However, to go to larger networks, careful material and device development is necessary to bring this figure down. The devices are tested in temperature-controlled chamber at 320 K and connected in the circuit configurations through external electrical components.

Oscillatory Neural Network

A single oscillator unit is realized biasing the VO_2 device in the negative differential resistance regime with a series transistor as described in Parihar et al. (2014). When the device is in its insulating state the bias voltage drops mainly across the VO_2, until the Joule heating brings its local temperature

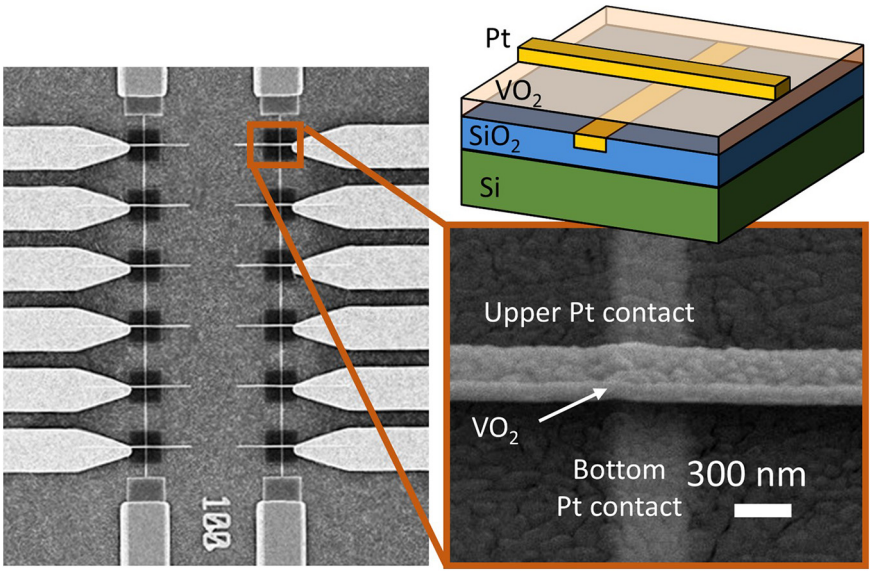


FIGURE 2 | Left: scanning electron microscopy (SEM) image of 12 VO_2 devices. Top right: schematic of a VO_2 device deposited on a Si/SiO_2 substrate. The device area is defined by the width of the Pt contact lines. On the bottom, a SEM image of a 350 nm device. Minimum device dimension demonstrated: 70 nm.

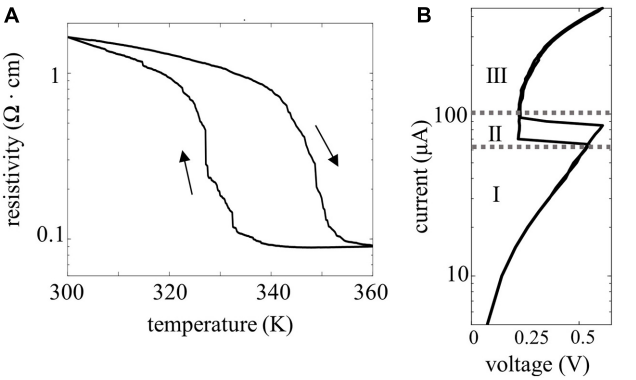


FIGURE 3 | (A) Resistivity measure of a 300 nm × 300 nm crossbar VO_2 device. The insulator to metal phase transition happens at around 340 K and registers 2 orders of magnitude phase change. **(B)** IV curve of a VO_2 device. Three different areas can be identified: the insulating region (I), the negative differential resistance region (II), and the metallic region (III).

above the phase transition, and the device switches to its metallic state. When the device is in the metallic state, the voltage drops mainly across the series transistor. When the bias is chosen such that the voltage drop does not exceed the upper threshold of the negative-differential resistance regime, Joule heating is reduced. The VO_2 device therefore cools and eventually switches back to its insulating state. The switching between the insulating and metallic state is therefore continuous and self-sustained, originating relaxation oscillations at the drain voltage of the transistor. The oscillators are coupled via resistive and capacitive elements, as shown in **Figure 4**, which ensure frequency and phase-locking of the drain voltage signals. The strength of the coupling element C_{ij} that connects oscillator i with oscillator j can be calculated starting from the patterns to be memorized, via the Hebbian Learning Rule (HLR) (Hoppensteadt and Izhikevich, 1999):

$$C_{ij} = \frac{1}{n}\sum_{k=0}^{m} \vartheta_i^k \overline{\vartheta_j^k}$$

where n is the total number of pixels per each image, or equivalently the number of oscillators in the network, ϑ_i^k is the value associated to the pixel i of pattern k, and m is the total number of patterns to be memorized in the ONN (**Figure 5**). These values C_{ij} are then translated in different values of the coupling resistance R_c between the oscillators. The memorized patterns appear in the operating ONN as stable phase relations between each oscillator i and j. An oscillator in phase with the reference oscillator is translated into a white pixel; an oscillator with 180° phase difference with the reference corresponds to a black pixel. Given m patterns memorized in the oscillatory neural network, the oscillators can stabilize their phase only according to one of the m memorized patterns. When the oscillators are initialized to an unstable phase relation, they will relax to the nearest stable ensemble of phase relations, i.e., to the nearest memorized pattern. In this way, from a distorted pattern a memorized pattern is retrieved. In our system in **Figure 4**, the oscillators are initialized to have different phase relations via a delay of the bias voltages to each oscillator. For instance, an oscillator representing a white pixel input is switched on at a time $t_d = 0$ compared to a reference signal; an oscillator representing a black pixel input is switched on at a time $t_d = T/2$ compared to the reference signal, with T indicating the period of one oscillation. Gray-scale values correspond to proportional delays. The output is represented by

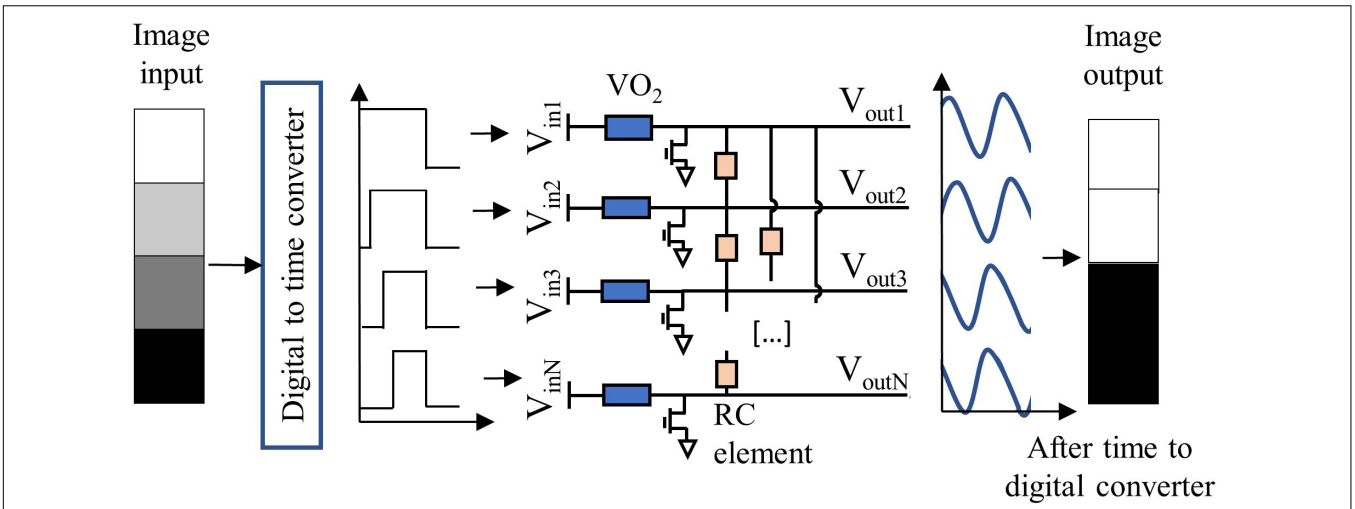


FIGURE 4 | The coupled oscillator network serves as an image filter. The image input is converted into a delay of the oscillators' input signal. A single oscillator unit comprises a VO_2 phase-change element in series with a transistor. The coupling is realized with an externally connected resistance and a capacitance. The capacitance value is fixed, while the resistance value can be changed to store different patterns in the network and can be later substituted with a memristor.

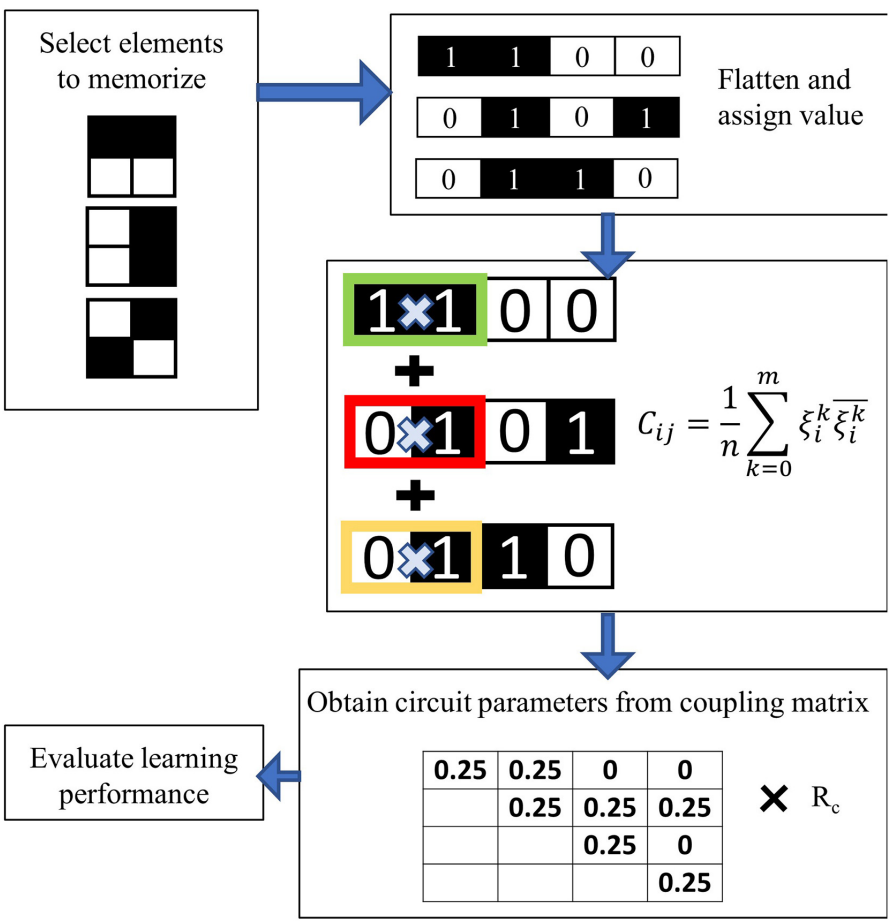


FIGURE 5 | Flow chart of the learning. Weights are assigned to the pixel of each training image; the Hebbian Learning Rule is used to compute the coupling weights, which are translated into circuit values of the coupling resistance R_c.

the phase of the oscillating transistor drain voltage, compared to a reference. When the network is initialized in this fashion to a phase relation between the oscillators that is different from the stored, stable phase relations, the system relaxes to the nearest stable phase relation, therefore achieving recognition of a pattern. The success of the input time-delay technique for image recognition is explained in detail in Corti et al. (2020). The settling time to the desired output typically varies between 4 and 5 oscillation cycles, nevertheless, after 5 oscillation cycles the phase information becomes stable and can be read-out. The information about the recognized feature is contained solely in the relative phase of the oscillators. In the experiments presented in this paper, the input delay signal is generated through a signal generator unit (National Instruments), the output of the oscillators is acquired by a signal acquisition set-up and the phase calculated with post-processing. The circuit coupling elements are realized with external electrical resistances. As an outlook, the input time-delay can be implemented in hardware via ring oscillators, and the phase-to-digital conversion can be tackled as described in Staszewski et al. (2006). Also, the coupling resistance can be implemented with reconfigurable phase change memories (Boybat et al., 2018).

The simulations for the circuit implementation of the ONNs have been done using a Spice simulator. The VO_2 device was simulated with a behavioral model as described in Maffezzoni et al. (2015). TensorFlow™ was used for the CNN and the hybrid ONN-CNN algorithms. The TensorFlow™ code is used to calculate the input delay of the driving voltage of the oscillators, as described above, from an input image, taken from the MNIST dataset. The code then launches the circuit simulation of the ONN, which are conducted in SPICE. The output of the simulation, corresponding to the pattern retrieval computed by the simulated ONN circuit, is then fed back to the TensorFlow™ algorithm as an output image. The image is then processed in the subsequent CNN layers with the TensorFlow™ code.

The choice of the MNIST dataset to perform this computation is justified by the reduced dimensions of the dataset itself. To obtain precise results with the simulation, a high time resolution is required, with very small time-steps for each computation. The simulations of the ONN are therefore rather slow and require an extended simulation time and many computational resources. This problem is not present in the circuit implementation, as the hardware realization is able to perform at frequencies in the order of MHz. We are, however, positive that a similar approach on a more complex dataset will yield the same results here discussed with the MNIST dataset.

RESULTS

Four-Coupled Oscillators

In this section we present a demonstration of four coupled VO_2 oscillators on Si, in which multiple patterns can be memorized. To form relaxation oscillator circuits, the VO_2 resistors on a silicon wafer are coupled through externally connected resistors and capacitances. An example of the measured waveforms of four coupled oscillators is shown in **Figure 6**. The oscillators

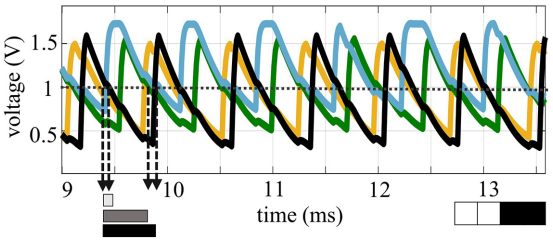


FIGURE 6 | Coupling of four VO_2 on Si oscillators. For reliable coupling, a hybrid R-C scheme was used, and the relative phase is calculated when the falling edge of the oscillations cross a 1 V threshold. In this experiment, an external capacitance of 150 nF was used on purpose to slow the oscillations, to enable a more precise sampling of the output signal.

appear to be locked in frequency and the phase relation is calculated taking the distance between the crossing of the 1 V line in the falling edge of the oscillator curves. The coupling network has been programmed to recognize features as in a first layer of a convolutional neural networks. Looking at available analysis of feature extraction in convolutional neural networks (Zeiler and Fergus, 2014), the filters in the first layer commonly select edge features, like borders, diagonal, horizontal and vertical edges. Therefore, for the experimental demonstration, the ONN was trained to store vertical, horizontal and diagonal patterns. The weights of the circuit elements were identified through the Hebbian learning rule. To the best of our knowledge, this is the first demonstration of 4 coupled VO_2 oscillators with memory capabilities realized on a silicon platform. The circuit parameters used for the experiments are: R_{12}, R_{13}, R_{24}, R_{34} = 82 kΩ, R_{23}, R_{14} = 130 kΩ, C_c = 5,6 nF, V_{gx} = 1.4–1.6 V, V_{in} = 1.8–2.2 V. The different values of gate voltages V_g and of the input signal V_{in} are used to achieve similar frequency for the oscillators, and to compensate from intrinsic differences in the devices, which present around 10% of device-to-device variability. The horizontal, vertical and diagonal patterns are identified over multiple experiments, as depicted in **Figure 7**. In addition, a fourth pattern in which all the oscillators result equally spaced was identified. The measurements are performed assuming Oscillator 1 as the reference oscillator; the phase of the other oscillators is calculated in respect to the crossing of the 1 V threshold of oscillator 1. Therefore, Oscillator 1 has always a phase equal to 0, with a minimal data scattering that is calculated taking into account the variability of the value of the first experimental point that crosses the 1 V line. The other oscillators present a larger scatter, which doesn't impair the clear identification of the various patterns. However, random fluctuation of the oscillations and cross-talk noise hindered the experimental pattern recognition using the input-delay to output phase inference process. This is expected to improve with further process and design optimization of the crossbar devices.

To demonstrate the filtering capabilities of the circuit on an entire image, without suffering from the non-idealities of the experimental demonstration, circuit simulations calibrated

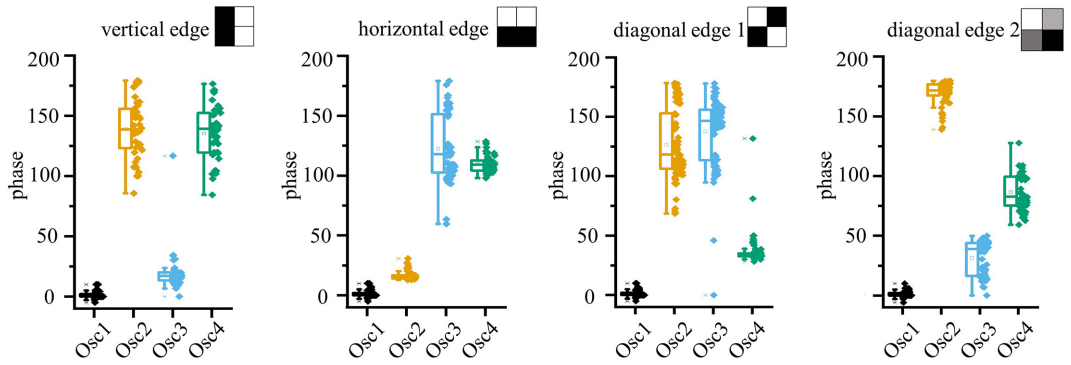


FIGURE 7 | Experimental phase data that demonstrate that 4 features (e.g., vertical, horizontal and 2 diagonals) can be stored simultaneously in one 4-oscillators network. The pattern can be controlled by the time-delay of the oscillator drive. Phase-noise due to device variability impedes practical application. Circuit parameters: R_{12}, R_{13}, R_{24}, R_{34} = 82 kΩ, R_{23}, R_{14} = 130 kΩ, C_c = 5,6 nF, V_{gx} = 1.4 –1.6 V, V_{in} = 1.8–2.2 V.

on the experiments were conducted using Spice, reducing the variability of the VO_2 devices from 10% to 5% and therefore increasing the recognition accuracy. In these simulations all the 4 patterns identified in the experiments were also observed; in addition, when the input delays of the circuit were chosen to be all the same, the oscillators in simulations were all oscillating in the in-phase configuration. This is an example of identification of a spurious pattern that was not encoded with the HRL. Spurious patterns arise when the memory capacity of the oscillatory neural network, that is studied to be 0.15n patterns for a n-oscillator network, is violated (Nishikawa et al., 2004; Follmann et al., 2015). Nevertheless, in such small oscillator networks the spurious patterns can be harvested as additional information. As shown in **Figure 8**, when using the 2 × 2 ONN filter on an image of the MNIST dataset, vertical, horizontal and diagonal edges can be identified. In addition, the background as well as the images parts that have little contrast, can be identified through the in-phase oscillating condition. This demonstrates that a single ONN filter can operate as convolutional feature edge extraction identifying 5 different features. Compared to previous work, the identification of the features does not need to proceed sequentially feature by feature, but it is done in parallel by the same filter. Moreover, the dimensionality of the filter should match the dimensionality of the input, i.e., 2D input arrays such as images are preferably processed using 2D filters. The 4 coupled oscillators system here discussed represents the minimum hardware realization to use ONNs as filters for image feature extraction.

ONN-CNN

Having shown that our simulations can reproduce experimental behavior, we extend the simulations to explore the use of ONNs in combination with CNNs. The ONN circuit described in section "Oscillatory Neural Network" is simulated with Spice simulations using for the VO_2 device a behavioral model as described in Maffezzoni et al. (2015). The simulations are done with 3 × 3 oscillators ONNs based on parameters extracted from experimental devices. A convolutional neural network with a structure similar to a VGG-13 is trained on the MNIST dataset with a standard back-propagation algorithm (**Table 1**). The trained weights are used to identify which features are recognized in the first layer of the CNN, that comprises 64 filters with a dimension of 3 × 3. In our network, as in Zeiler and Fergus (2014), it was also possible to identify multiple filters that selected horizontal, vertical and diagonal edges. We use the Hebbian Learning rule to store the same patterns in a 3 × 3 ONN matrix. The matrix dimension was chosen according to the dimension of the first layer convolution matrixes in the CNN. Ten thousand images from the MNIST dataset have been processed by the ONN matrix with a stride of 2, recognizing in each image vertical, diagonal, horizontal edges and uniform background. As already mentioned, storing of more than 0.15 n patterns, where n is the number of the oscillators (Follmann et al., 2015), results in the appearance of spurious patterns that can in principle hinder the feature edge extraction process. However, as already discussed for the 4-coupled oscillators experiments, the arising of spurious patterns is not detrimental for feature extraction operations. In the 3 × 3 filter case, we derived the pattern information from 3 key oscillators that oscillate in-phase for each memorized edge. For example, referring to what is depicted in **Figure 9**, each time oscillators 2, 5, and 8 oscillate in-phase a vertical edge is recognized, and similarly for the other edges.

With this technique, a dataset of 10,000 images filtered by the single ONN was calculated, with dimensions 13 × 13 × 5, where 5 represent the number of features recognized by the single ONN filter. The dataset was split in 6,000 training images and 4,000 test images.

Subsequently, five filters in the pre-trained CNN that provide the same filtered images were identified and replaced by the ONN with a simple transfer learning process:

1. The 64 CNN filters were convolved with the same images from the MNIST dataset and activated with a Relu function.

TABLE 1 | Schematic of the convolutional neural network architecture used in this work for performing the MNIST classification task.

MNIST dataset	**27 × 27 × 1,000**
ONN-CNN 5 ONN filters + 59 CNN filters	3 × 3 × 64, stride = 2, padding = same
CNN1	3 × 3 × 64, stride = 1, padding = same
Max pool 1	2 × 2, stride = 2, padding = same
CNN 2(×2)	3 × 3 × 128, stride = 1, padding = same
Max pool 2	2 × 2, stride = 2, padding = same
CNN 3(×2)	3 × 3 × 256, stride = 1, padding = same
Max pool 3	2 × 2, stride = 2, padding = same
Fully connected 1	4,096
Fully connected 2	1,000
Fully connected 3	10

The network architecture is inspired by the VGG-13 architecture.

2. The CNN-filtered images were compared to the ONN-filtered images calculating the mean square error; the minimum of the mean square error was used to identify the filters from the CNN that can be substituted with the ONN.
3. A new dataset is created after the first layer, substituting the images filtered by 5 CNN filters with the 5 filtered images from the ONN.

The remaining neural network layers are trained on the new dataset, achieving a recognition accuracy on the training set of 100% and on the test set of 95%. The original CNN, in comparison, reported better accuracy on the test set, of 97%. The reason for the worsening of the neural network performances is attributed to the cases in which the ONN fails the feature edge extraction. In fact, insufficient training of the ONN (just using HLR) also leads to recognition errors. The implementation of a backpropagation algorithm to the ONN layer would allow to increase the recognition performance in the network. We therefore implemented and tested a backpropagation scheme in our simulations. In **Figure 10** we show an input image feature that should be recognized as a vertical edge. However, when the ONN is trained with the HLR the recognition fails. A cost function $C = (\varphi_{train} - \varphi_{out})^2/2$ is calculated from on the phases of the desired output φ_{train} and the obtained output φ_{out}. Assuming an exponential dependence of the rising and falling edge of the relaxation oscillator waveforms, the derivative in time can be derived and an improved coupling matrix calculated. During subsequent epochs of this training the phase error is reduced. In the example shown in **Figure 10**, the feature is recognized after 8 epochs of training. While blurred features (allowing 40% gray scale) were only recognized with 30% probability using the untrained ONN, 100% of the features were recognized with the trained ONN. The extension of the backpropagation algorithm to the entire ONN-CNN is yet to be implemented, but is expected to boost the recognition performance. In addition, the direct implementation of the backpropagation algorithm would allow for direct training of a CNN algorithm on an ONN platform and should ultimately result in an increase of the training speed. Despite the reduction in recognition performances, the proposed ONN implementation

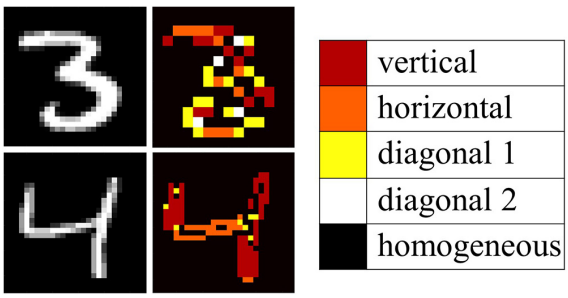


FIGURE 8 | Simulation of convolution operation on MNIST images with a 2 × 2 VO_2 oscillator filter, which corresponds to 5 digital filters of the first layer of a CNN. The simulations are calibrated with the experimental results in **Figure 7**.

allows for a reduction of the number of parameters that need to be trained by the network. In fact, 45 parameters undergo training for 5 CNN filters of 3 × 3 pixels size, however, only 36 parameters need to be trained for a single ONN that performs all filtering actions. The number of parameters to be trained is therefore reduced of 20%: this can represent an important advantage in terms of speed and power consumption when training larger networks. In addition, a further acceleration of the network speed and a further reduction of the number of memory accesses is achieved by the parallel processing of 5 filters from a single ONN unit, whilst in the standard CNN these five convolution actions are performed sequentially.

Benchmark

In this section we benchmark the convolution operations conducted with the ONN compared to a conventional CPU or GPU. We assume that the first layer of the convolutional neural network presented in this paper is integrally realized via ONN filters operating in parallel. The first layer of the CNN consists of 64 filters of 3 × 3 dimension passing through a 27 × 27 pixel image with a stride of 2, accounting to total of 13 × 13 operations per filter. Assuming that each ONN can perform 5 filtering actions inherently, a total amount of 13 × 13 × 64/5 ≈ 2,200 ONNs is required, which corresponds

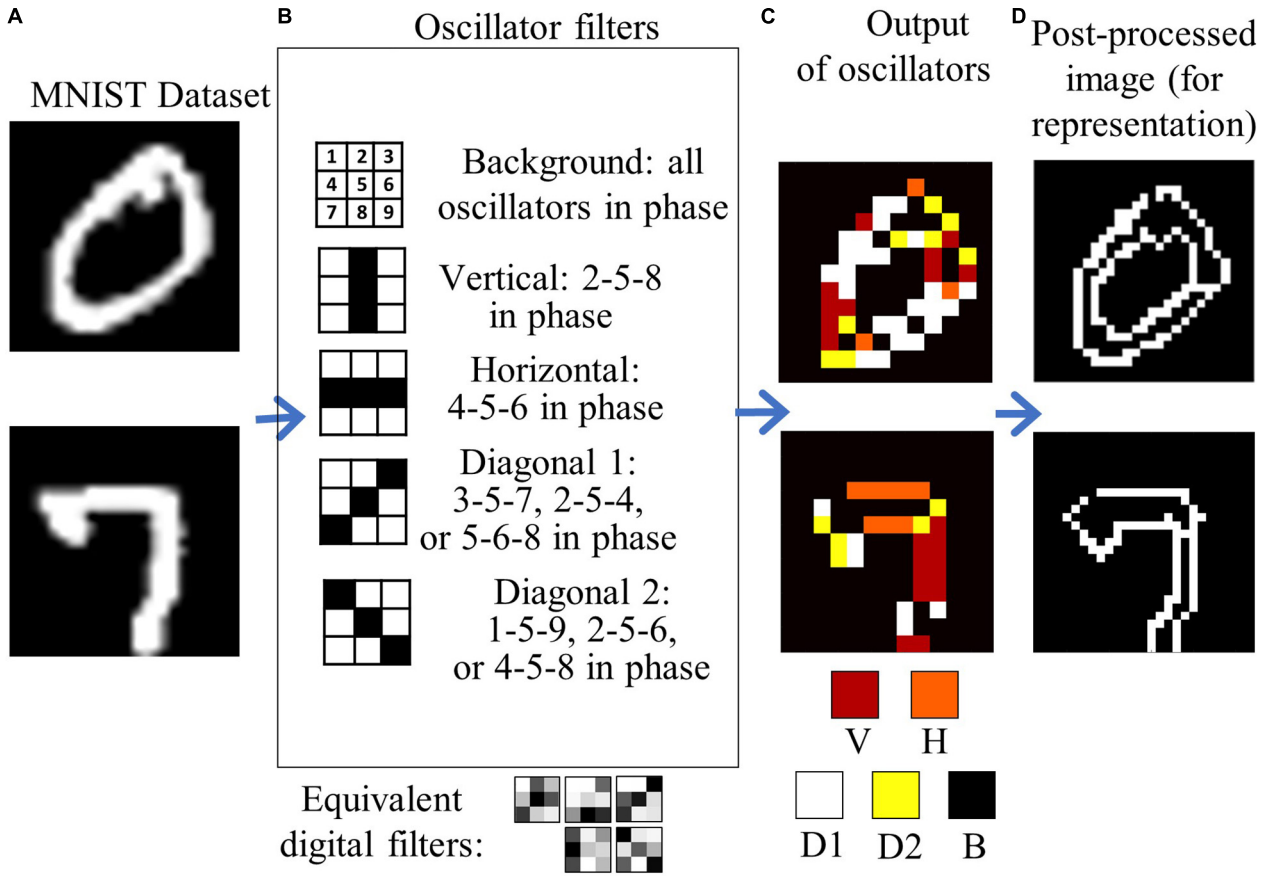


FIGURE 9 | Extension of the convolution filtering operation to a 3 × 3 oscillator matrix with stride of 2. **(A)** example of handwritten digit images from the MNIST dataset. **(B)** Five digital filters are replaced with 1 ONN filter that performs equivalent edge extraction actions. The 9-bit information output is compressed in a 5-bit fashion for better representation of the edge direction in **(C)**. In **(D)**, the image in **(C)** is post-processed and expanded to a 27 × 27 pixel image to show the effectiveness of the ONN filter in recognizing the image features.

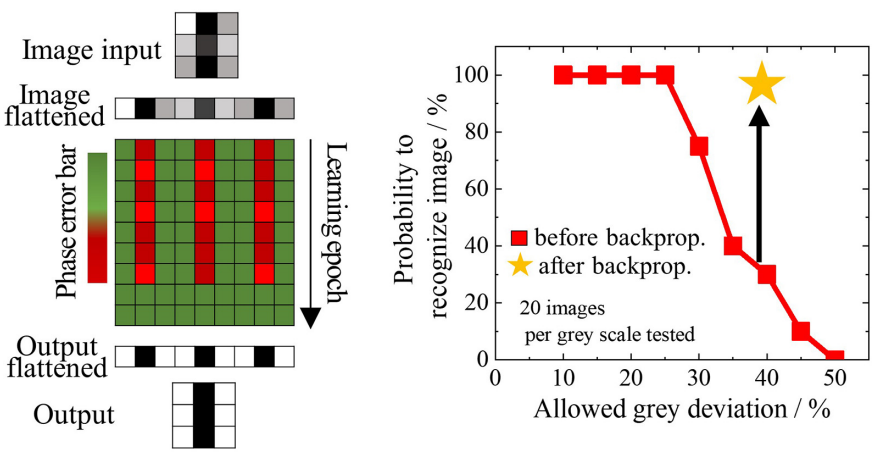


FIGURE 10 | Example of backpropagation algorithm applied to a vertical edge recognition problem of the VO_2 ONN filter. Left: a distorted edge is given as an input to the ONN filter. The difference between the expected output phase and the output phase of the filter is depicted as pixel coloring from green (no phase error) to red (phase error). At the beginning the ONN filter fails the recognition, but after 8 learning epochs the filter is able to recognize the edge as a vertical edge. Right: the backpropagation algorithm allows the recognition of the image for increasingly distorted input features. This backpropagation algorithm is suitable for implementing filter training in an ONN-CNN.

TABLE 2 | Benchmark of the ONN technology against currently available platforms for convolutional neural network applications.

	ONN (current)	ONN (projected)	CPU Intel's Core I9	GPU Tesla V100
Frames/s	0.6×10^6	20×10^6	5×10^6	600×10^6
Energy/frame	3.4 μJ	3 nJ	20 μJ	500 nJ
TFLOP/s	0.12	4	1	120
TFLOP/s W^{-1}	0.06	67	0.01	0.4

roughly to 20,000 oscillator units and 80'000 memristors for implementing the coupling. Assuming a minimum feature size of 100 nm for both the VO_2 oscillators as well as the memristor, the total estimated area would be around 0.001 mm^2.

For calculating the power consumption of the circuit, we refer to Shukla et al. (2016), Corti et al. (2020), that demonstrate operations of the oscillators at the power $P = 20$ μW with a scaled supply voltage <1 V and f = 3 MHz frequency operation. The total energy for the ONN to process one image with 64 filters at 3 MHz, including the waiting time of 5 oscillating period for the output stabilization, is calculated as

$$P \times f \times 5 = 0.6\,\mu J/frame$$

Similarly, assuming the mean value of the coupling resistance to be around 100 kΩ, and the voltage drop across it 0.7 V, the total energy consumption of the memristors is calculated to be 3.4 μ J/frame.

Scaling of the device dimensions, it is envisioned that the VO_2 oscillator could be driven with 1 μW @ 0.3 V at a moderately increased oscillation frequency of 20 MHz. Moreover, through improved processing and resulting device uniformity, the coupling strength could be weakened allowing 1 MΩ coupling resistance (Shukla et al., 2015). The figure of merit for such a scaled system would improve by 3 orders of magnitude resulting in an energy consumption of 3 nJ/frame.

For conducting the same operation, a standard GPU needs to perform (13 × 13) convolutions × 64 filters × (3 × 3) pixels/filter = 97,344 multiply-accumulation operation, that correspond to around 200,000 flops. In Intel's CPU Core I9, which runs 1 TFLOP/s at 95 W, the total energy accounts for 20 μJ/frame; in the NVIDIA Tesla V100 GPU, that operates 120 TFLOP/s @ 300 W, the total energy is 500 nJ/frame (**Table 2**). We can conclude that the ONN system, when built with the current VO_2 technology, is operating now at less power consumption of a conventional CPU, and given the scaling capabilities presented in other works, has the possibility of outperforming the top GPU available on the market. This analysis has been conducted not considering the peripheral circuitry that the ONN system will require, and therefore should be taken just as a projection of the potentiality of this technology and as an indication on the reduction in power consumption that this architecture can bring. Further benchmark should, however, be conducted at a stage when the technology is more advanced, to compare the performances to other specialized hardware that serve as accelerators for neural networks applications.

CONCLUSION

A concept for exploiting oscillatory neural networks as hardware accelerators in convolutional neural networks is presented in this paper. A 4-nodes oscillatory neural network was built with scaled VO_2 oscillators' technology on a Si platform. We show that the time-encoded output signal can store up to 5 trained filters and performs the equivalent function of multiple digital convolutional filters in a neural network. We expand the concept to a 3 × 3 VO_2-ONN trained with Hebbian learning rule and simulate back-propagation for performance optimization. With the 3 × 3 filter and a transfer learning approach, we show that multiple digital filters of a CNN can be trained on a single ONN platform, achieving competitive recognition performances.

DATA AVAILABILITY STATEMENT

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

AUTHOR CONTRIBUTIONS

EC and SK ideated and designed the concepts and experiments proposed in this work. EC was responsible for the device fabrication, the measurements, circuit simulations, and the neural network coding. JC conducted the characterization on the crossbar devices. KN and JR conducted the deposition and annealing of the VO_2 films. KM, BG, and AI were involved in the discussion, experiment design, and editing of the manuscript and provided valuable input at multiple stages of this work. All authors contributed to the article and approved the submitted version.

FUNDING

We would like to acknowledge the support of our partners and the founding of the HORIZON 2020 NEURONN project (Grant no. 871501) and PHASE-CHANGE SWITCH project (Grant no. 737109).

REFERENCES

Bai, G., Niang, K. M., and Robertson, J. (2020). Preparation of atomic layer deposited vanadium dioxide thin films using tetrakis(ethylmethylamino) vanadium as precursor. *J. Vac. Sci. Technol. A* 38:052402. doi: 10.1116/6.0000353

Boybat, I., Le Gallo, M., Nandakumar, S. R., Moraitis, T., Parnell, T., Tuma, T., et al. (2018). Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* 9:2514. doi: 10.1038/s41467-018-04933-y

Corti, E., Gotsmann, B., Moselund, K., Ionescu, A. M., Robertson, J., and Karg, S. (2019). Scaled resistively-coupled VO2 oscillators for neuromorphic computing. *Solid State Electron.* 168:107729. doi: 10.1016/j.sse.2019.107729

Corti, E., Gotsmann, B., Moselund, K., Stolichnov, I., Ionescu, A., and Karg, S. (2018). "Resistive coupled VO2 oscillators for image recognition," in *Proceedings of the 2018 IEEE International Conference on Rebooting Computing (ICRC), McLean, VA* (New York, NY: IEEE), 1–7. doi: 10.1109/ICRC.2018.8638626

Corti, E., Khanna, A., Niang, K., Robertson, J., Moselund, K. E., Gotsmann, B., et al. (2020). Time-delay encoded image recognition in a network of resistively coupled VO on Si oscillators. *IEEE Electron Device Lett.* 41, 629–632. doi: 10.1109/LED.2020.2972006

Cotter, M. J., Fang, Y., Levitan, S. P., Chiarulli, D. M., and Narayanan, V. (2014). "Computational architectures based on coupled oscillators," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI, ISVLSI, Tampa, FL* (New York, NY: IEEE Computer Society), 130–135. doi: 10.1109/ISVLSI.2014.87.

Follmann, R., Macau, E. E. N., Rosa, E., and Piqueira, J. R. C. (2015). Phase oscillatory network and visual pattern recognition. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1539–1544. doi: 10.1109/TNNLS.2014.2345572

Hölzel, R. W., and Krischer, K. (2011). Pattern recognition with simple oscillating circuits. *New J. Phys.* 13:073031. doi: 10.1088/1367-2630/13/7/073031

Hoppensteadt, F. C., and Izhikevich, E. M. (1999). Oscillatory neurocomputers with dynamic connectivity. *Phys. Rev. Lett.* 82, 2983–2986. doi: 10.1103/PhysRevLett.82.2983

Izhikevich, E. (2000). *Computing with Oscillators*. Available online at: http://citeseerx.ist.psu.edu/viewdoc/summary? (accessed October 19, 2020).

Jackson, T. C., Sharma, A. A., Bain, J. A., Weldon, J. A., and Pileggi, L. (2015). Oscillatory neural networks based on TMO nano-oscillators and multi-level RRAM cells. 5, 230–241. doi: 10.1109/JETCAS.2015.2433551

Jackson, T., Pagliarini, S., and Pileggi, L. (2018). "An oscillatory neural network with programmable resistive synapses in 28 nm CMOS," in *Proceedings of the 2018 IEEE International Conference on Rebooting Computing (ICRC), McLean, VA*, (New York, NY: IEEE), 1–7. doi: 10.1109/ICRC.2018.8638600

Kim, H. T., Kim, B. J., Choi, S., Chae, B. G., Lee, Y. W., Driscoll, T., et al. (2010). Electrical oscillations induced by the metal-insulator transition in VO2. *J. Appl. Phys.* 107:023702. doi: 10.1063/1.3275575

Li, S., Liu, X., Nandi, S. K., Venkatachalam, D. K., and Elliman, R. G. (2015). High-endurance megahertz electrical self-oscillation in Ti/NbOx bilayer structures. *Appl. Phys. Lett.* 106:212902. doi: 10.1063/1.4921745

Liu, Q., and Mukhopadhyay, S. (2018). "Unsupervised learning using pretrained CNN and associative memory bank," in *Proceedings of the International Joint Conference on Neural Networks, Rio de Janeiro* (New York, NY: IEEE), 1–8. doi: 10.1109/IJCNN.2018.8489408

Liyanagedera, C. M., Yogendra, K., Roy, K., and Fan, D. (2016). "Spin torque nano-oscillator based oscillatory neural network," in *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC* (New York, NY: IEEE), 1387–1394. doi: 10.1109/IJCNN.2016.7727360

Maffezzoni, P., Daniel, L., Shukla, N., Datta, S., and Raychowdhury, A. (2015). Modeling and simulation of vanadium dioxide relaxation oscillators. 62, 2207–2215. doi: 10.1109/TCSI.2015.2452332

Mian, M. S., Okimura, K., and Sakai, J. (2015). Self-oscillation up to 9 MHz based on voltage triggered switching in VO 2 /TiN point contact junctions. *J. Appl. Phys.* 117:215305. doi: 10.1063/1.4922122

Niang, K. M., Bai, G., and Robertson, J. (2020). Influence of precursor dose and residence time on the growth rate and uniformity of vanadium dioxide thin films by atomic layer deposition. *J. Vac. Sci. Technol. A* 38:042401. doi: 10.1116/6.0000152

Nikonov, D. E., Csaba, G., Porod, W., Shibata, T., Voils, D., Hammerstrom, D., et al. (2015). Coupled-oscillator associative memory array operation for pattern recognition. 1, 85–93. doi: 10.1109/JXCDC.2015.2504049

Parihar, A., Shukla, N., Datta, S., and Raychowdhury, A. (2014). Exploiting synchronization properties of correlated electron devices in a non-boolean computing fabric for template matching. 4, 450–459. doi: 10.1109/JETCAS.2014.2361069

Parihar, A., Shukla, N., Datta, S., and Raychowdhury, A. (2015). Synchronization of pairwise-coupled, identical, relaxation oscillators based on metal-insulator phase transition devices: a model study. *J. Appl. Phys.* 117, 1–12. doi: 10.1063/1.4906783

Premkumar, P. A., Toeller, M., Radu, I. P., Adelmann, C., Schaekers, M., Meersschaut, J., et al. (2012). Process study and characterization of VO2 thin films synthesized by ALD using TEMAV and O3 precursors. *ECS J. Solid State Sci. Technol.* 1, 169–174. doi: 10.1149/2.009204jss

Raychowdhury, A., Parihar, A., Smith, G. H., Narayanan, V., Csaba, G., Jerry, M., et al. (2019). Computing with networks of oscillatory dynamical systems. 107, 73–89. doi: 10.1109/JPROC.2018.2878854

Romera, M., Talatchian, P., Tsunegi, S., Abreu Araujo, F., Cros, V., Bortolotti, P., et al. (2018). Vowel recognition with four coupled spin-torque nano-oscillators. *Nature* 563, 230–234. doi: 10.1038/s41586-018-0632-y

Ruzmetov, D., Gopalakrishnan, G., Deng, J., Narayanamurti, V., and Ramanathan, S. (2009). Electrical triggering of metal-insulator transition in nanoscale vanadium oxide junctions. *J. Appl. Phys.* 106:083702. doi: 10.1063/1.3245338

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

Sharma, A. A., Li, Y., Skowronski, M., Bain, J. A., and Weldon, J. A. (2015). High-frequency taox-based compact oscillators. *IEEE Trans. Electron Devices* 62, 3857–3862. doi: 10.1109/TED.2015.2475623

Shukla, N., Parihar, A., Cotter, M., Barth, M., Li, X., Chandramoorthy, N., et al. (2015). "Pairwise coupled hybrid vanadium dioxide-MOSFET (HVFET) oscillators for non-boolean associative computing," in *Proceedings of the Technical Digest International Electron Devices Meeting IEDM 2015 (February ed.), San Francisco, CA* (New York, NY: IEEE), 28.7.1–28.7.4. doi: 10.1109/IEDM.2014.7047129

Shukla, N., Tsai, W. Y., Jerry, M., Barth, M., Narayanan, V., and Datta, S. (2016). "Ultra low power coupled oscillator arrays for computer vision applications," in *Proceedings of the 2016 IEEE Symposium on VLSI Technology: Digest of Technical Papers-Symposium on VLSI Technology*, (Honolulu, HI: Institute of Electrical and Electronics Engineers Inc.). doi: 10.1109/VLSIT.2016.7573439

Staszewski, R. B., Vemulapalli, S., Vallur, P., Wallberg, J., and Balsara, P. T. (2006). 1.3 V 20 ps time-to-digital converter for frequency synthesis in 90-nm CMOS. *IEEE Trans. Circ. Syst. II Express Briefs* 53, 220–224. doi: 10.1109/TCSII.2005.858754 *Cityq

Takami, H., Kanki, T., and Tanaka, H. (2014). Multistep metal insulator transition in VO2 nanowires on Al2O3 (0001) substrates. *Appl. Phys. Lett.* 104:23104. doi: 10.1063/1.4861720

Nishikawa, T., Lai, Y.-C., and Hoppensteadt, F. C. (2004). Capacity of oscillatory associative-memory networks with error-free retrieval. *Phys. Rev. Lett.* 92:108101.

Tobe, R., Mian, M. S., and Okimura, K. (2020). Coupled oscillations of VO 2-based layered structures: experiment and simulation approach. *J. Appl. Phys.* 127:195103. doi: 10.1063/5.0001382

Tsai, W.-Y., Li, X., Jerry, M., Xie, B., Shukla, N., Liu, H., et al. (2016). Enabling new computation paradigms with HyperFET-an emerging device. *IEEE Trans. Multi Scale Comput. Syst.* 2, 30–48. doi: 10.1109/TMSCS.2016.2519022

Zeiler, M. D., and Fergus, R. (2014). "Visualizing and understanding convolutional networks," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, eds D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars (Cham: Springer Verlag), 818–833. doi: 10.1007/978-3-319-10590-1_53

Zhang, T., Haider, M. R., Massoud, Y., and Alexander, J. I. D. (2019). An oscillatory neural network based local processing unit for pattern recognition applications. *Electronics* 8:64. doi: 10.3390/electronics8010064

Conflict of Interest: EC, JC, KM, BG, and SK were employed by IBM.

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 Neuroinformatics

ORIGINAL RESEARCH
published: 10 June 2021
doi: 10.3389/fninf.2021.609147

Event-Based Update of Synapses in Voltage-Based Learning Rules

Jonas Stapmanns[1,2]*, *Jan Hahne*[3], *Moritz Helias*[1,2], *Matthias Bolten*[3], *Markus Diesmann*[1,4,5] and *David Dahmen*[1]

[1] Institute of Neuroscience and Medicine (INM-6), Institute for Advanced Simulation (IAS-6), JARA Institute Brain Structure Function Relationship (INM-10), Jülich Research Centre, Jülich, Germany, [2] Department of Physics, Institute for Theoretical Solid State Physics, RWTH Aachen University, Aachen, Germany, [3] School of Mathematics and Natural Sciences, Bergische Universität Wuppertal, Wuppertal, Germany, [4] Department of Physics, Faculty 1, RWTH Aachen University, Aachen, Germany, [5] Department of Psychiatry, Psychotherapy and Psychosomatics, Medical Faculty, RWTH Aachen University, Aachen, Germany

Due to the point-like nature of neuronal spiking, efficient neural network simulators often employ event-based simulation schemes for synapses. Yet many types of synaptic plasticity rely on the membrane potential of the postsynaptic cell as a third factor in addition to pre- and postsynaptic spike times. In some learning rules membrane potentials not only influence synaptic weight changes at the time points of spike events but in a continuous manner. In these cases, synapses therefore require information on the full time course of membrane potentials to update their strength which a priori suggests a continuous update in a time-driven manner. The latter hinders scaling of simulations to realistic cortical network sizes and relevant time scales for learning. Here, we derive two efficient algorithms for archiving postsynaptic membrane potentials, both compatible with modern simulation engines based on event-based synapse updates. We theoretically contrast the two algorithms with a time-driven synapse update scheme to analyze advantages in terms of memory and computations. We further present a reference implementation in the spiking neural network simulator NEST for two prototypical voltage-based plasticity rules: the Clopath rule and the Urbanczik-Senn rule. For both rules, the two event-based algorithms significantly outperform the time-driven scheme. Depending on the amount of data to be stored for plasticity, which heavily differs between the rules, a strong performance increase can be achieved by compressing or sampling of information on membrane potentials. Our results on computational efficiency related to archiving of information provide guidelines for the design of learning rules in order to make them practically usable in large-scale networks.

Keywords: event-based simulation, voltage-based plasticity rules, spiking neural network simulator, NEST, Clopath rule, Urbanczik-Senn rule

OPEN ACCESS

Edited by:
Giacomo Indiveri,
University of Zurich, Switzerland

Reviewed by:
Thomas Nowotny,
University of Sussex, United Kingdom
Timoleon Moraitis,
Huawei Technologies, Switzerland
Gopalakrishnan Srinivasan,
MediaTek, Taiwan

***Correspondence:**
Jonas Stapmanns
jonas.stapmanns@rwth-aachen.de

Received: *22 September 2020*
Accepted: *07 April 2021*
Published: *10 June 2021*

Citation:
Stapmanns J, Hahne J, Helias M, Bolten M, Diesmann M and Dahmen D (2021) Event-Based Update of Synapses in Voltage-Based Learning Rules. Front. Neuroinform. 15:609147. doi: 10.3389/fninf.2021.609147

1. INTRODUCTION

One mechanism for learning in the brain is implemented by changing the strengths of connections between neurons, known as synaptic plasticity. Already early on, such plasticity was found to depend on the activity of the connected neurons. Donald Hebb postulated the principle "Cells that fire together, wire together" (Hebb, 1949). Later on, it was shown that plasticity is shaped by

temporal coordination of activities even down to the level of individual spikes (Markram et al., 1997; Bi and Poo, 1998). Synaptic plasticity rules for spiking neural networks, such as spike timing-dependent plasticity (STDP, Gerstner et al., 1996), consequently employ spike times of pre- and postsynaptic cells to predict the change in connections.

In recent years, a new class of biologically inspired plasticity rules has been developed that takes into account the membrane potential of the postsynaptic neuron as an additional factor (for a review, see Mayr and Partzsch, 2010; Gerstner et al., 2014). The rule by Clopath et al. (2010) can be seen as a prototypical example for a voltage-based plasticity rule since long-term potentiation of synapses depends on the presynaptic spike arrival and a filtered version of the postsynaptic membrane potential. This additional voltage dependence enables the Clopath rule to describe phenomena that are not covered by ordinary STDP but can be observed in experimental data, such as the complex frequency dependence of the synaptic weight changes in spike pairing experiments (Sjöström et al., 2001). Furthermore, it provides a mechanism for the creation of strong bidirectional connections in networks, which have been found to be overrepresented in some cortical areas (Song et al., 2005).

Further inspiration for recently proposed plasticity rules originates from the field of artificial neural networks. These networks showed great success in the past decade, for example in image or speech recognition tasks (Hinton et al., 2006; Krizhevsky et al., 2012; Hannun et al., 2014; LeCun et al., 2015). The involved learning paradigms, for example the backpropagation algorithm (Werbos, 1974; Lecun, 1985; Parker, 1985; Rumelhart et al., 1986), are, however, often not compatible with biological constraints such as locality of information for weight updates. To bridge the gap to biology, different biologically inspired approximations and alternatives to the backpropagation algorithm have been proposed (Neftci et al., 2017; Sacramento et al., 2018; Bellec et al., 2020; Cartiglia et al., 2020). A common feature of many of these rules is that weight updates not only depend on the output activity of pre- and postsynaptic cells, but also on a third factor, which is a time-continuous signal. A prominent example of such biologically and functionally inspired rules is the voltage-based plasticity rule proposed by Urbanczik and Senn (2014), where the difference between somatic and dendritic membrane potential serves as an error signal that drives learning. This rule, incorporated in complex microcircuits of multi-compartment neurons, implements local error-backpropagation (Sacramento et al., 2018).

Research on functionally inspired learning rules in biological neural networks is often led by the requirement to implement a particular function rather than efficiency. Present studies are therefore primarily designed to prove that networks with a proposed learning rule minimize a given objective function. Indeed many learning rules are rather simple to implement and to test in *ad-hoc* implementations where at any point the algorithm has access to all state variables. While the latter implementations are sufficient for a proof of principle, they are hard to reuse, reproduce, and generalize. In particular, simulations are restricted to small network sizes, as the simulation code cannot be straight-forwardly distributed across compute nodes and thus parallelized. This also limits the simulation speed which is, in particular, problematic given that successful learning requires simulating networks for long biological times.

In parallel to the above efforts are long-term developments of simulation software for biological neural networks (for a review, see Brette et al., 2007). Such open-source software, combined with interfaces and simulator-independent languages (Davison et al., 2008; Djurfeldt et al., 2010, 2014), supports maintainability and reproducibility, as well as community driven development. The design of such simulators is primarily led by implementation efficiency. Code is optimized for neuron and synapse dynamics, with the aim to upscale simulations to biologically realistic network sizes. A modular structure of the code facilitates re-use and extensions in functionality. Therefore, one aim of the community should be the transfer of *ad-hoc* proof-of-principle implementations to these well-tested platforms. Given the differences in design principles behind the exploratory development of specific models and general-purpose simulation technology, this transfer is not trivial. In the current study, we show how to make voltage-based learning rules compatible with spiking neural network simulators that employ an event-driven update scheme for synapses.

Modern network simulators use individual objects for different neurons and synapses. One common strategy of parallelization is to distribute these objects across many compute processes (Lytton et al., 2016; Jordan et al., 2018). Communication between neurons then implies exchange of information between compute processes. Neurons in the brain primarily communicate in an event-based fashion via spikes. The duration of these spike events is on the order milliseconds, which together with typical rates during physiological brain states of a few spikes per second yields a coupling that is sparse in time (**Figure 1A**). Spiking simulators emulate this communication by idealizing spikes as instantaneous events. Thus, in the absence of direct electrical coupling via gap junctions (Kumar and Gilula, 1996; Hahne et al., 2015; Jordan et al., 2020a), there is no neuronal interaction in between two spike events such that the dynamics of neuronal and synaptic state variables can be evolved independently in time. This led to the development of event-based simulation schemes, where synapses are only updated in their state at the times of incoming spikes (Watts, 1994; Morrison et al., 2005). This significantly reduces the amount of function calls to synapse code and optimizes computational performance in network simulations. Modern spiking network simulators such as Auryn (Zenke and Gerstner, 2014), Brian2 (Stimberg et al., 2014), Neuron (Carnevale and Hines, 2006), NEST (Gewaltig and Diesmann, 2007), and Nevesim (Pecevski et al., 2014) therefore employ an event-based update scheme for synapses. Even though spike events at single synapses are rare, each single neuron typically receives a large amount of spikes in rapid succession due to its large number of incoming connections (in-degree). This suggests a time-driven update of neurons (**Figure 1B**). The resulting hybrid simulation scheme for neurons and synapses (Morrison et al., 2005; D'Haene et al., 2014; Krishnan et al.,

2017) is nowadays commonly used across many spiking network simulators (for a review, see Brette et al., 2007).

An event-based scheme for synapses is perfectly suitable for classical STDP rules, which only rely on a comparison between the timings of spike events. In these rules, synaptic weights formally depend on spike traces, which are continuous signals that are fully determined by spike timings of pre- and postsynaptic neurons and which can be updated at the time of spike events. Optimizations of simulations including STDP have been extensively discussed (Song et al., 2000; Ros et al., 2006; Rudolph and Destexhe, 2006; Morrison et al., 2007a) and routinely used in spiking network simulators such as Auryn (Zenke and Gerstner, 2014), Brian2 (Stimberg et al., 2014), Neuron (Carnevale and Hines, 2006), NEST (Gewaltig and Diesmann, 2007), and Nevesim (Pecevski et al., 2014) as well as in neuromorphic hardware (Pfeil et al., 2013; Serrano-Gotarredona et al., 2013; Neftci et al., 2014; Galluppi et al., 2015; Friedmann et al., 2016; Thakur et al., 2018). Some STDP variants also include the membrane potential of postsynaptic cells at the time points of presynaptic spike events as a gating variable (Brader et al., 2007; Diederich et al., 2018). At the update, these rules only require the synapse to know the current value of the postsynaptic membrane potential in addition to the pre- and postsynaptic spike time. Obtaining this value from the neuron objects is efficient to implement and already employed in Brian2 (Stimberg et al., 2014) and in a range of neuromorphic systems (Serrano-Gotarredona et al., 2013; Galluppi et al., 2015; Qiao et al., 2015; Moradi et al., 2018; Cartiglia et al., 2020).

We here focus on more complex voltage-based learning rules which not only rely on membrane potentials at the time points of spike events, but on an extended history of membrane potentials. For these rules synapses continuously require information from the postsynaptic neurons in order to update their weights (Clopath et al., 2010; Mayr and Partzsch, 2010; Brea et al., 2013; Yger and Harris, 2013; Urbanczik and Senn, 2014; Albers et al., 2016). This a priori breaks the idea behind an event-based update scheme. Therefore, previous attempts to incorporate such voltage-based plasticity in spiking network simulators resorted to time-driven synapse updates for NEST (Jordan et al., 2020b) and NEURON (see implementation of Clopath plasticity on ModelDB, Hines et al., 2004). These implementations therefore only profit from the simulation environment on the level of the implementation language, but have not been able to exploit the algorithmic optimizations and speedup of event-based synapse updates.

In this study we present an efficient archiving method for the history of postsynaptic state variables that allows for an event-based update of synapses and thus makes complex voltage-based plasticity rules compatible with state-of-the-art simulation technology for spiking neural networks. In particular, we derive two event-based algorithms that store a time-continuous or discontinuous history, respectively. These algorithms apply to plasticity rules with any dependence on post-synaptic state variables and therefore cover a large range of existing models (Brader et al., 2007; Mayr and Partzsch, 2010; Legenstein and Maass, 2011; Brea et al., 2013, 2016; Yger and Harris, 2013; Qiao et al., 2015; Albers et al., 2016; Sheik et al., 2016; Diederich et al., 2018; Sacramento et al., 2018; Cartiglia et al., 2020). We theoretically analyze advantages of the two event-driven algorithms with respect to each other and compare to a straightforward time-driven algorithm.

The presented simulation concepts are exemplified and evaluated in a reference implementation in the open source simulation code NEST (Gewaltig and Diesmann, 2007; Jordan et al., 2019). The reference implementation thereby exploits existing functionality of a scalable software platform which can be used on laptops as well as supercomputers. NEST is employed by a considerable user community and equipped with an interface to the programming language Python (Eppler et al., 2009) that is currently widely used in the field of computational neuroscience (Muller et al., 2015). It supports relevant neuron models and connection routines for the construction of complex networks. Despite this flexibility the simulation engine shields the researcher from the difficulties of handling a model description in a distributed setting (Morrison et al., 2005; Plesser et al., 2015).

To exemplify the general simulation algorithms, we here focus on the voltage-based plasticity rules by Clopath et al. (2010) and Urbanczik and Senn (2014). The two rules represent opposing ends of a family of learning rules in the amount of data required to compute weight updates. The Clopath rule by design only triggers plasticity in the vicinity of postsynaptic spike events; storing a history, which is non-continuous in time, thus becomes beneficial. In contrast, the Urbanczik-Senn rule considers noisy prediction errors based on postsynaptic membrane voltages and spikes. Such prediction errors never vanish and therefore always need to be stored to update the weights, leading to time-continuous histories. For a given span of biological time, simulations of the Urbanczik-Senn rule are therefore by design less efficient than those of the Clopath rule. However, we show that a compression of membrane potential information reduces this performance gap. Changing the learning rule to include a sparse sampling of the membrane voltage further increases efficiency and makes performance comparable to simulations with ordinary STDP.

Our study begins with a specification of the mathematical form of the learning rules that we consider (section 2.1). We distinguish between classical STDP (section 2.2) and voltage-based rules (section 2.3) and present a special case where voltage-based rules can be efficiently implemented by compressing information on the postsynaptic membrane potential. We then introduce the Clopath and the Urbanczik-Senn rule as two examples of voltage-based plasticity (sections 2.4 and 2.5). In section 3 we first contrast time- and event-driven schemes for updating synapses with voltage-based plasticity (section 3.1). Subsequently, we detail a reference implementation of the algorithms in NEST (section 3.2) and use this to reproduce results from the literature (section 3.3). After that, we examine the performance of the reference implementation for the Clopath and the Urbanczik-Senn rule (section 3.4). Conclusions from the implementation of the two rules are drawn in section 3.5, followed by a general Discussion in section 4. The technology described in the present article is available in the 2.20.1 release of the simulation software NEST as open source. The conceptual and algorithmic work is a module in our long-term collaborative

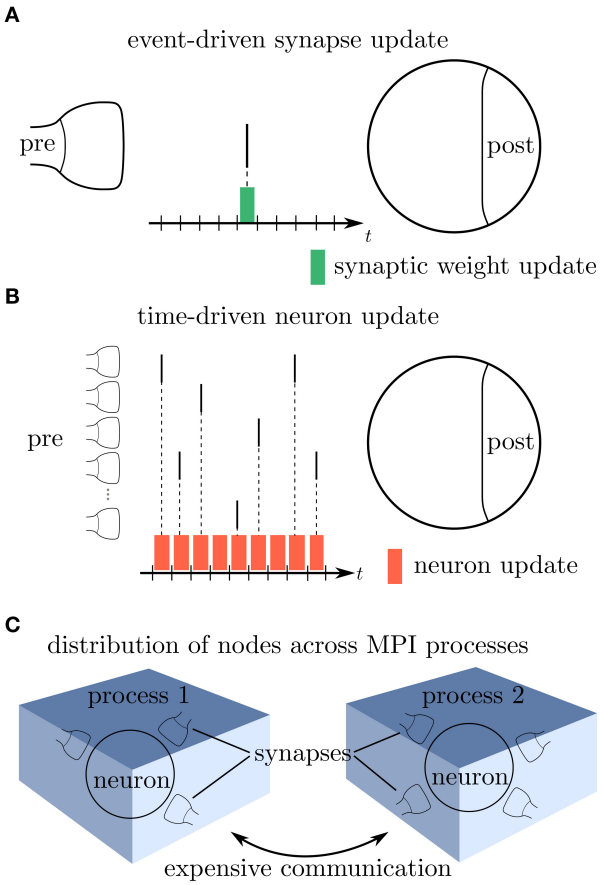


FIGURE 1 | Update schemes for neurons and synapses. **(A)** A spike crosses a synapse from the presynaptic (pre) to the postsynaptic (post) neuron. Since this is a rare event, the synaptic weight is computed only when a spike is delivered, indicated by the green bar (event-driven update). **(B)** Neurons with a large in-degree receive spikes in rapid succession which suggests a time-driven update of the neuron's state in each time step (red bars). **(C)** Since the computation of the synaptic weights requires information from the postsynaptic neuron, storing the synapses on the same compute node reduces the amount of expensive communication between compute processes.

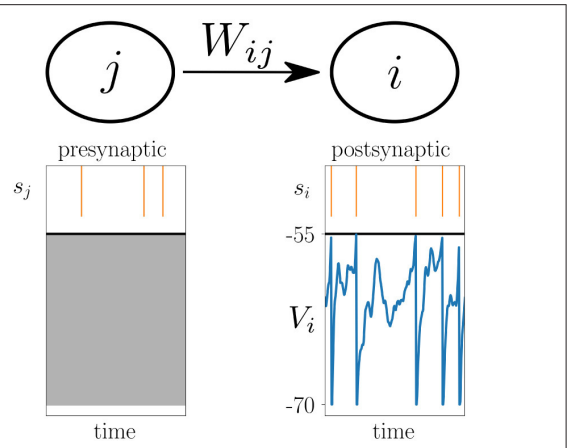


FIGURE 2 | Voltage-based plasticity rules. The change ΔW_{ij} in synaptic strength between presynaptic neuron j and postsynaptic neuron i depends on the presynaptic spike train s_j, the postsynaptic spike train s_i and the postsynaptic membrane potential V_i.

project to provide the technology for neural systems simulations (Gewaltig and Diesmann, 2007).

2. MATERIALS AND METHODS

2.1. General Structure of Learning Rules

The focus of this study are plasticity models of the general form

$$\frac{dW_{ij}(t)}{dt} = F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)), \tag{1}$$

where the change $\frac{dW_{ij}(t)}{dt}$ of the synaptic weight W_{ij} between the presynaptic neuron j and postsynaptic neuron i is given by a function F that potentially depends on the current synaptic weight $W_{ij}(t)$, as well as on $s_i^*(t), s_j^*(t), V_i^*(t)$ which are causal functionals of the postsynaptic spike train s_i, the presynaptic spike train s_j, and the postsynaptic membrane potential V_i, respectively (**Figure 2**). Causal functional here refers to $s_i^*(t)$ potentially depending on all past values $s_i(t' \leq t)$; likewise $V^*(t)$ depends on $V(t' \leq t)$. Note that for simplicity of the notation, we only show one function F on the right hand side of (1), while generally there could be a sum of multiple functions or functionals F_α, where each one depends on spike trains and membrane potentials in a different manner. Note also that F mixes information of pre- and postsynaptic neurons, while the functionals denoted by $*$ only need to take into account information of either the pre- or postsynaptic neuron. In cases where F is a functional, i.e., where F depends on the whole time course of its arguments, it can take into account an additional joint history dependence on s_i^*, s_j^*, and V_i^*. A special case, the Urbanczik-Senn learning rule, is discussed further below.

One can formally integrate (1) to obtain the weight change between two arbitrary time points t and T

$$\Delta W_{ij}(t, T) = \int_t^T dt' F(W_{ij}(t'), s_i^*(t'), s_j^*(t'), V_i^*(t')). \tag{2}$$

2.2. Spike-Timing Dependent Plasticity

In general, the integral on the right hand side of the equation cannot be calculated analytically. There is, however, a notable exception, which is the model of spike-timing dependent plasticity (STDP). This model is a form of Hebbian plasticity that relies on the exact spike times of pre- and postsynaptic neurons and ignores any effect of the postsynaptic membrane potential. The dependence on the exact spike times becomes apparent by the fact that either the pre- or postsynaptic spike functional is the spike train itself, for example

$$s_i^*(t) = s_i(t) = \sum_k \delta(t - t_i^k), \tag{3}$$

where t_i^k is the k-th spike of the i-th neuron. This yields a plasticity rule that reads (Morrison et al., 2008)

$$\frac{dW_{ij}(t)}{dt} = -f_-(W_{ij}(t))s^*_{-,i}(t)s_j(t) + f_+(W_{ij}(t))s^*_{+,j}(t)s_i(t) \quad (4)$$

with functions $f_\pm$ that model the weight dependence, and functionals $s^*_\pm(t) = (\kappa_\pm * s)(t)$ given as convolutions of spike trains with kernels $\kappa_\pm$, which in the classical STDP rule correspond to one-sided exponential decays. The appearance of the raw spike trains (delta distributions) in the differential equation of the STDP model renders the integration of the ODE trivial

$$\begin{aligned}\Delta W_{ij}(t,T) = &- \sum_{\text{spikes } k} f_-(W_{ij}(t_j^k))\kappa_{-,i}(t_j^k) \\ &+ \sum_{\text{spikes } l} f_+(W_{ij}(t_i^l))\kappa_{+,j}(t_i^l),\end{aligned} \quad (5)$$

where $t_j^k, t_i^l \in [t,T]$. An update of the synaptic weight between any two time points only requires knowledge of the weight and spike functionals at the timing of the pre- and postsynaptic spikes.

For models that do not solely rely on exact spike times, but for example on filtered versions of the spike trains, much more information is needed in order to calculate a weight update $\Delta W_{ij}(t,T)$ between any two time points. This makes the computation more involved: the synapse needs all values of $W_{ij}(t'), s_i^*(t'), s_j^*(t'), V_i^*(t')$ for $t' \in [t,T]$ to update its weight. The remainder of this study describes different approaches to this problem and their advantages and disadvantages.

2.3. Voltage-Based Plasticity

In a time-driven neuron update, the membrane potential in many simulators is computed at each simulation step $t^\alpha = \alpha \cdot h$, where h is the simulation step size and $\alpha \in \mathbb{N}$. For plasticity models that rely on the membrane potential, the time discretization of (2) therefore yields

$$\Delta W_{ij}(t,T) = \sum_{\text{steps } \alpha} \Delta W_{ij}(t^\alpha, t^{\alpha+1}), \quad (6)$$

$$\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = \int_{t^\alpha}^{t^{\alpha+1}} dt' F(W_{ij}(t'), s_i^*(t'), s_j^*(t'), V_i^*(t^\alpha)). \quad (7)$$

which, in comparison to the small sum over spikes in the STDP rule (5), contains a large sum over all time steps t^α in between time points t and T. As the membrane potential is only known at time points $t' = t^\alpha$, it generally enters (7) in a piecewise constant manner – hence the argument $V(t^\alpha)$. The synapse therefore predominantly needs information of the postsynaptic neuron in order to update its weight. Thus, in a distributed simulation framework, where neurons are split across multiple compute processes, it is beneficial to store the synapses at the site of the postsynaptic neurons in order to reduce communication (**Figure 1C**). This confirms the earlier design decision of Morrison et al. (2005) who place synapses at the site of the postsynaptic neuron to reduce the amount of data communicated by the presynaptic site.

If weight changes ΔW_{ij} depend on the synaptic weight themselves, then (7) cannot be used in practice as intermediate weights $W_{ij}(t')$ for $t^\alpha < t' < t^{\alpha+1}$ are not known. In this scenario, weight changes have to be calculated on the simulation grid with $W_{ij}(t') \to W_{ij}(t^\alpha)$ in case of a forward Euler scheme, or $W_{ij}(t') \to W_{ij}(t^{\alpha+1})$ in case of a backward Euler scheme. In the following we, for simplicity, stick to the forward Euler setting and arrive at the core computation for voltage-based plasticity rules

$$\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = \int_{t^\alpha}^{t^{\alpha+1}} dt' F(W_{ij}(t^\alpha), s_i^*(t'), s_j^*(t'), V_i^*(t^\alpha)). \quad (8)$$

Given that s_i and s_j are spike trains, the functionals s_i^* and s_j^* are obtained trivially from the kernels of their corresponding Volterra expansions. If F in addition does not depend on s_i^* and s_j^* in a too complicated manner, which is usually the case (see examples below), the integral in (8) can be calculated analytically.

2.3.1. Compression of Postsynaptic Information

The major operation of the plasticity scheme in terms of frequency and complexity is the computation of infinitesimal weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$. Since the presynaptic spike train s_j^* enters F in (8), the same postsynaptic information on s_i^* and V_i^* is used many times for very similar computations: the membrane potential trace of each neuron is effectively integrated many times. Is there a way to employ the result of the computation $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$ for neuron j for the computations $\Delta W_{ik}(t^\alpha, t^{\alpha+1})$ for other neurons $k \neq j$? In a simple setting, where F factorizes into $F(W_{ij}(t), s_i^*(t), s_j^*(t), V_i^*(t)) = s_j^*(t)\, G\left(s_i^*(t), V_i^*(t)\right)$ with $s_j^*(t) = \left(\kappa * s_j\right)(t)$ and

$$\kappa(t) = H(t)\,\frac{1}{\tau} e^{-\frac{t}{\tau}}, \quad (9)$$

defined via the Heaviside step function $H(x)$, we can make use of the property

$$s_j^*(t) = (s_j^*(t_{LS}) + \tau^{-1})\, e^{-(t-t_{LS})/\tau}, \quad (10)$$

where $t > t_{LS}$ and t_{LS} denotes the last spike time of the presynaptic neuron. In this case the weight update in between two spike events factorizes

$$\Delta W_{ij}(t_{LS}, t_S) = \underbrace{\left(s_j^*(t_{LS}) + \tau^{-1}\right)}_{=:\bar{x}_j(t_{LS})} \underbrace{\int_{t_{LS}}^{t_S} dt' e^{-(t'-t_{LS})/\tau}\, G\left(s_i^*(t'), V_i^*(t')\right)}_{=:\Delta W_i(t_{LS}, t_S)}, \quad (11)$$

where the latter integral $\Delta W_i(t_{LS}, t_S)$ is independent of the presynaptic spike train s_j^*. Moreover, ΔW_i depends on t_{LS} only via an exponential prefactor. Thus, an integral $\Delta W_i(t_1, t_2)$ over an arbitrary time interval $t_{LS} \leq t_1 < t_2 \leq t_S$ which is completely independent of any presynaptic information, can be used as a part of the whole integral $\Delta W_i(t_{LS}, t_S)$ since it can be decomposed as

$$\begin{aligned}\Delta W_i(t_{LS}, t_S) = &\ \Delta W_i(t_{LS}, t_1) + e^{-\frac{t_1 - t_{LS}}{\tau}} \Delta W_i(t_1, t_2) \\ &+ e^{-\frac{t_2 - t_{LS}}{\tau}} \Delta W_i(t_2, t_S).\end{aligned}$$

Therefore, whenever an integral of the postsynaptic quantities s_i^* and V_i^* is computed, it can be used to advance the weight update of all incoming connections and the integration only needs to be performed once. To account for the generally different last spike times t_{LS} of the incoming connections, the postsynaptic neuron stores the different $\Delta W_i\,(t_{LS}, t)$ in a so-called *compressed history*. At the time of an incoming spike event, $\Delta W_i(t_{LS}, t_{\mathrm{S}})$ can be read out by the synapse for the correct t_{LS} of that synapse and be combined with the stored presynaptic spike trace s_j^*.

2.4. Example 1: Clopath Plasticity

The Clopath rule (Clopath et al., 2010) was designed as a voltage-based STDP rule that accounts for non-linear effects of spike frequency on weight changes which had been previously observed in experiments (Sjöström et al., 2001). It does so by using the evolution of the postsynaptic membrane voltage around postsynaptic spike events instead of the postsynaptic spikes themselves. This requires a neuron model that takes into account features of membrane potential excursions near spike events, such as modified adaptive exponential integrate-and-fire (aeif) model neurons that are used in the original publication (Clopath et al., 2010, see section 5.2) or Hodgkin-Huxley (hh) neurons that are used in a NEURON reference implementation on ModelDB (Hines et al., 2004).

The plasticity rule is of the general form (1) with a sum of two different functions F_α on the right hand side. It treats long-term depression (LTD) and potentiation (LTP) of the synaptic weight in the two terms F_{LTD} and F_{LTP}, with

$$\begin{aligned} F_{\mathrm{LTD}}\left(s_j(t), V^*_{i,\mathrm{LTD}}(t)\right) &= -A_{\mathrm{LTD}}\; s_j(t)\; V^*_{i,\mathrm{LTD}}(t) \qquad (12)\\ \text{with}\quad V^*_{i,\mathrm{LTD}} &= (\bar{u}_- - \theta_-)_+\,,\\ \bar{u}_-(t) &= (\kappa_- * V_i)(t - d_s) \end{aligned}$$

and

$$\begin{aligned} F_{\mathrm{LTP}}\left(s^*_j(t), V^*_{i,\mathrm{LTP}}(t)\right) &= A_{\mathrm{LTP}}\; s^*_j(t)\; V^*_{i,\mathrm{LTP}}(t) \qquad (13)\\ \text{with}\quad s^*_j &= \kappa_s * s_j\,,\\ V^*_{i,\mathrm{LTP}} &= (\bar{u}_+ - \theta_-)_+ (V_i - \theta_+)_+\,,\\ \bar{u}_+(t) &= (\kappa_+ * V_i)(t - d_s)\,. \end{aligned}$$

Here $(x - x_0)_+ = H(x - x_0)\,(x - x_0)$ is the threshold-linear function and $H(x)$ is the Heaviside step function. A_{LTD} and A_{LTP} are prefactors controlling the relative strength of the two contributions. $\kappa_\pm$ are exponential kernels of the form (9), which are applied to the postsynaptic membrane potential, and κ_s is an exponential kernel applied to the presynaptic spike train. The time-independent parameters $\theta_\pm$ serve as thresholds below which the (low-pass filtered) membrane potential does not cause any weight change (**Figure 3**). Note that A_{LTP} can also depend on the membrane potential. This case is described in **Appendix Section 5.5**.

In a reference implementation of the Clopath rule by C. Clopath and B. Torben-Nielsen available on ModelDB (Hines et al., 2004), there is a subtle detail not explicitly addressed in the original journal article. In their implementation the authors introduce an additional delay d_s between the convolved version of the membrane potentials $\bar{u}_\pm$ and the bare one [cf. parameter d_s in (12) and (13)]. The convolved potentials are shifted backwards in time by the duration of a spike d_s (see **Supplementary Tables 1, 3**). As a result, the detailed shape of the excursion of the membrane potential during a spike of the postsynaptic neuron does not affect the LTP directly but only indirectly via the low-pass filtered version $\bar{u}_+$, see red background in **Figure 3B**. Incorporating this time shift in $\bar{u}_\pm$ is essential to reproduce the results from Clopath et al. (2010) on spike-pairing experiments.

The depression term F_{LTD} depends on the unfiltered spike train s_j. It can thus be treated analogous to ordinary STDP rules (cf. (4)ff). In particular, $V^*_{i,\mathrm{LTD}}$ only needs to be available for time points of presynaptic spikes (potentially taking into account additional delays of the connection). The potentiation term F_{LTP}, however, depends on the filtered spike train s^*_j; $V^*_{i,\mathrm{LTP}}$ consequently needs to be known also for times in between spike events.

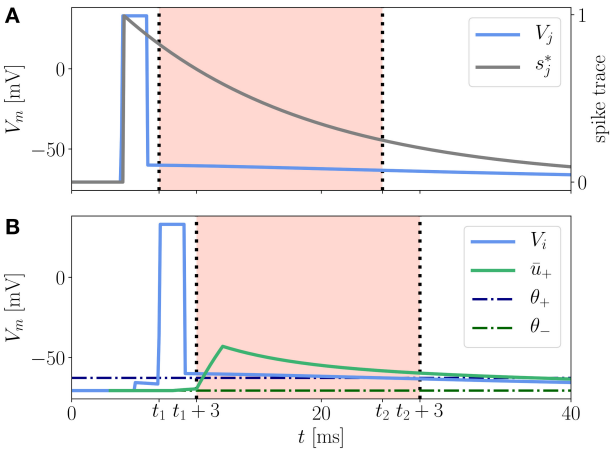


FIGURE 3 | Illustration of LTP contribution to the Clopath rule. A presynaptic neuron **(A)** and a postsynaptic neuron **(B)** emit a spike at $t_{\mathrm{sp,pre}} = 4$ ms and $t_{\mathrm{sp,post}} = 6$ ms, respectively. The presynaptic spike elicits a trace s_j^* (gray) at the synapse. The excursion of the postsynaptic membrane potential V_i (**B**, blue) elevates the low-pass filtered potential $\bar{u}_+$ (green) so that both V_i and $\bar{u}_+$ exceed the respective thresholds θ_+ (dash-dotted, dark blue) and θ_- (dash-dotted, dark green), cf. (13), between t_1 and t_2. Only within this period, shifted by $d_s = 3$ ms, which is for times $t_1 + 3\,\mathrm{ms} < t < t_2 + 3\,\mathrm{ms}$ (**B**, red background), see section 3.2.3 for details, the LTP of the synaptic weight is non-vanishing because of the threshold-linear functions in equation (13). The shift by $d_s = 3$ ms does not apply to the spike trace (**A**, red background). The rectangular shape of the spikes is achieved by a clamping of the membrane potential to $V_{\mathrm{clamp}} = 33$ mV for a period of $t_{\mathrm{clamp}} = 2$ ms.

2.5. Example 2: Urbanczik-Senn Plasticity

The Urbanczik-Senn rule (Urbanczik and Senn, 2014) applies to synapses that connect to dendrites of multicompartment model neurons. The main idea of this learning rule is to adjust the weights of dendritic synapses such that the dendrite can predict the firing rate of the soma. The dendrite expects the firing rate to

be high when the dendrite's membrane potential is elevated due to many incoming spikes at the dendrite, and to be low if there are only a few incoming spikes. Thus, for this prediction to be true, synapses that transmit a spike toward the dendrite while the firing rate of the soma is low are depressed and those that provide input while the soma's firing rate is high are facilitated. Learning can be triggered by applying a teacher signal to the neuron via somatic synapses such that the actual somatic firing deviates from the dendritic prediction.

The plasticity rule is again of the general form (1), with a functional F on the right hand side that reads

$$F[s_j^*, V_i^*] = \eta\, \kappa * \left(V_i^*\, s_j^*\right) \tag{14}$$

$$\text{with } V_i^* = \left(s_i - \phi(V_i)\right)\, h\left(V_i\right), \tag{15}$$

$$s_j^* = \kappa_s * s_j.$$

with exponential filter kernels κ and κ_s and non-linearities ϕ and h. Note that F depends on the postsynaptic spike train s_i via V_i^*. The latter can be interpreted as a prediction error, which never vanishes as spikes s_i (point process) are compared against a rate prediction $\phi(V_i)$ (continuous signal).

In order to solve (1), we need to integrate over $F[s_j^*, V_i^*]$, cf. (2). Writing down the convolution with κ explicitly, we obtain

$$\begin{aligned}\Delta W_{ij}(t,T) &= \int_t^T dt'\, F[s_j^*, V_i^*](t') \\ &= \int_t^T dt'\, \eta \int_0^{t'} dt'' \kappa\left(t'-t''\right) V_i^*\left(t''\right) s_j^*\left(t''\right). \end{aligned} \tag{16}$$

A straight forward implementation of this expression is inefficient in terms of memory usage and computations because of the two nested integrals. However, since the kernels κ and κ_s are exponentials, one can perform one of the integrations analytically (see **Appendix Section 5.1** for a derivation) to rewrite the weight change as

$$\Delta W_{ij}(t,T) = \eta\left[I_1\left(t,T\right) - I_2\left(t,T\right) + I_2\left(0,t\right)\left(1 - e^{-\frac{T-t}{\tau_\kappa}}\right)\right], \tag{17}$$

$$\text{with } I_1\left(a,b\right) = \int_a^b dt\, V_i^*\left(t\right) s_j^*\left(t\right),$$

$$I_2\left(a,b\right) = \int_a^b dt\, e^{-\frac{b-t}{\tau_\kappa}} V_i^*\left(t\right) s_j^*\left(t\right),$$

which is in line with the general formulation discussed in section 2.3.

3. RESULTS

In the following, we first discuss time- and event-driven update schemes for synapses with voltage-based plasticity. Then we present a reference implementation for the Clopath rule (Clopath et al., 2010) and the Urbanczik-Senn rule (Urbanczik and Senn, 2014) in the spiking network simulator NEST (Jordan et al., 2019). Finally, we show that these implementations reproduce results of the original works and we assess their simulation performance on a distributed computing architecture.

3.1. Time-Driven vs. Event-Driven Update Scheme for Synapses With Voltage-Based Plasticity

Let's assume in the following that t_{LS} and t_{S} denote two consecutive spike times of a presynaptic neuron j. The synaptic weight $W_{ij}(t_{\mathrm{S}})$ corresponding to the spike at time t_{S} can be obtained from the weight $W_{ij}(t_{\mathrm{LS}})$ at the time of the previous spike at t_{LS} and (6) by employing (8) to calculate the latter. As F mixes information of the pre- and postsynaptic neurons, this computation should be done in the synapse. Since there are no spikes in between t_{LS} and t_{S}, it does not matter when the synapse is performing the updates of its weight. Two possibilities are: 1) Neurons calculate their own s^* and V^* for the current time step and make it accessible to the synapse to enable direct readout and update according to (8) in every time step. This method corresponds to a time-driven update of synapses (**Figure 4A**). 2) Neurons store a history of s^* and V^* and the synapse reads out this information at t_{S}, i.e., at the time where the weight update becomes relevant for the network. This method corresponds to an event-driven update of synapses (**Figure 4B**). Both methods have their advantages and disadvantages analyzed in the following section.

3.1.1. Time-Driven Scheme

In a time-driven update scheme the information on the membrane potential is directly processed by the synapses such that only the current value of the membrane potential needs to be stored, corresponding to a membrane potential history of length $L = 1$ (**Figure 5** and **Table 1**). For a simulation of T time steps, the history needs to be manipulated $H = T$ times: the single stored value gets updated once per time step. The price that comes with the short history is that synapses need to be updated as often as neurons. This amounts to $M = K \cdot T$ function calls to synapse code for each neuron. Here K denotes the in-degree of each neuron. Each function call of synapse code causes a single computation of $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$, giving rise to in total $C = K \cdot T$ computations per neuron. The membrane potential trace is thus effectively integrated K times; once for each synapse. As both K and T are large numbers in typical simulations of plastic cortical networks, the amount of function calls and computations is therefore large in this setting. The time-driven scheme furthermore forces the execution of synapse code also at time steps where no update would be required, i.e., at time steps, where s_i^*, s_j^*, V_i^* have values for which $\Delta W_{ij}(t^\alpha, t^{\alpha+1}) = 0$. In addition, for delayed connections a history of V_i^* of length $L = d_{\max}$ of the maximal delay $d_{\max}$ measured in simulation steps needs to be stored. We here assume the delay to be on the postsynaptic side; it represents the time the fluctuations of the somatic membrane potential propagate back through the dendrites to the synapses. Therefore, F does not depend on $V_i^*(t)$, but on $V_i^*(t - d_j)$ with a delay d_j encoding the location of the synapse with presynaptic neuron j.

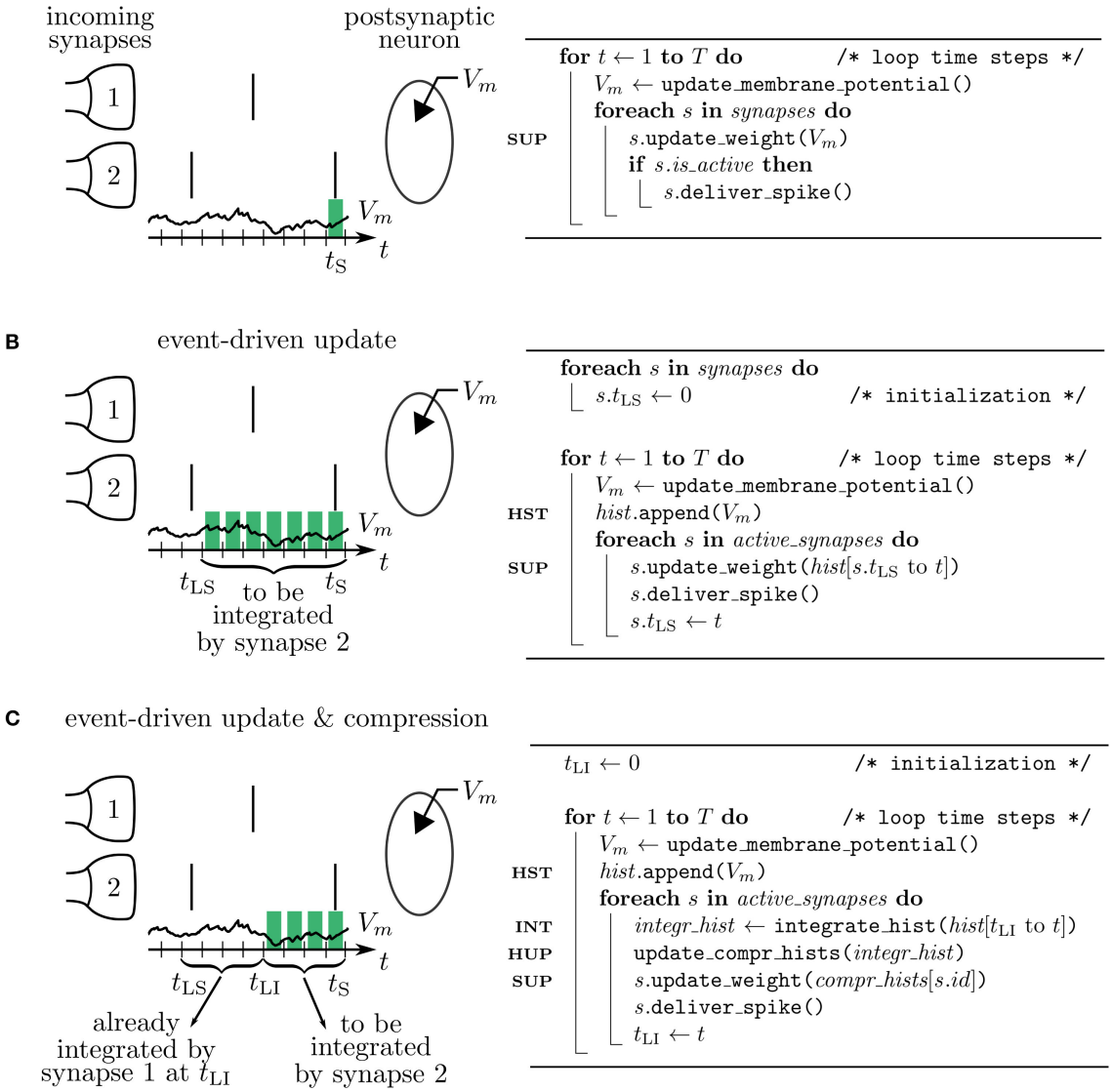


FIGURE 4 | Simulation concepts. Left: illustration of processing the postsynaptic voltage trace V_m (t) for three simulation concepts. Two incoming synapses (1 and 2) transmit spikes (black, vertical bars) to the postsynaptic neuron. Depending on the algorithm, a different number of past membrane potentials has to be stored (green blocks) so that synapse 2 can update its weight when it delivers the spike at time t_S. Right: corresponding pseudocodes. **(A)** In the time-driven update scheme the synaptic weight change is evaluated in every time step of the simulation for all the synapses. This requires only the latest value of the membrane potential to be accessible by the synapse to update its weight at t_S (see line marked SUP in pseudocode). **(B)** In the event-driven update scheme the computation of the synaptic weight change is performed only if a spike crosses the synapse. Therefore, storage of the time trace of V_m (see HST in code) from the last spike delivered by synapse 2 at t_{LS} up to the current time step t_S is needed. **(C)** In the compressed event-driven update scheme synapse 2 uses the time trace of V_m integrated from the last incoming spike at t_{LI} up to the current time step t_S (see INT in code) to complete its weight update (see SUP in code) and also to advance that of synapse 1. The preceding part of V_m from t_{LS} to t_{LI} was already integrated and applied to all incoming synapses (see HUP in code) by synapse 1 when it delivered the spike at t_{LI}.

3.1.2. Event-Driven Scheme

In an event-driven update scheme for synapses, the time trace of the membrane potential V_i^* needs to be stored until all incoming synapses have read out the information to update their weight for a given period. The storage and management of such a history can be expensive in terms of memory and runtime. In each time step, the value of the current membrane potential is appended to the history, leading to $H = T$ history manipulations for a simulation of T time steps. Assuming for simplicity a homogeneous inter-spike interval of I time steps between consecutive spikes of single neurons, we in the following showcase some qualitative history sizes. As synapses need all values of V_i^* in between two consecutive spikes, the maximum history length is $L = I$ (**Figure 5**). In case of different firing rates, I corresponds to the

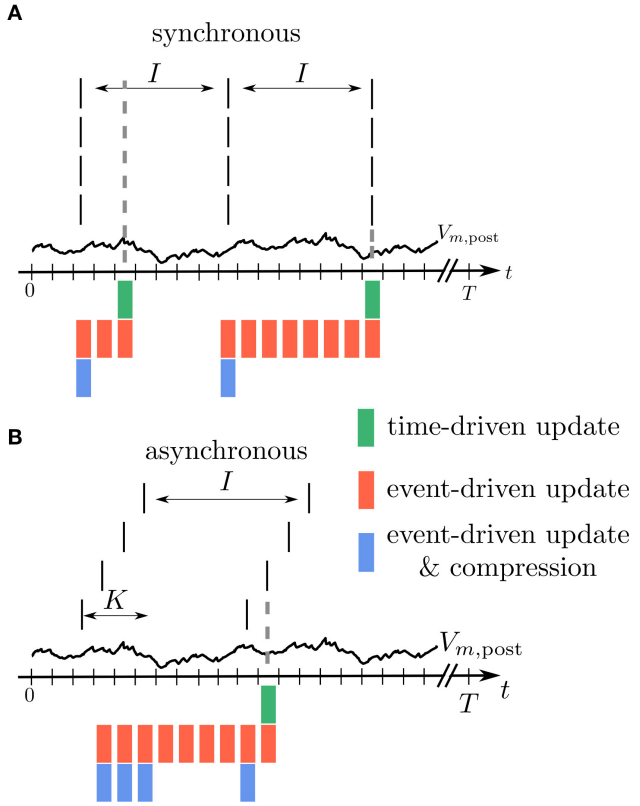


FIGURE 5 | Illustration of buffer sizes for different simulation schemes in case of fully synchronous or asynchronous spikes. **(A)** All incoming spikes arrive synchronously: In the time-driven scheme the synaptic weight is updated in every time step of the simulation, so that only the current value of $V_{m,\text{post}}$ needs to be available (green). In the event-driven scheme every synapse processes $V_{m,\text{post}}$ from the last spike to the current one. Therefore, the relevant time trace needs to be stored (red). In the compressed event-driven scheme this part of $V_{m,\text{post}}$ is processed only once and used to update the weight of all the synapses. Since the weight change is a function of the last spike time which is the same for all the synapses, only one value needs to be updated (blue). In this situation the length L of the compressed history is $i = 1$, see **Table 1**. **(B)** All incoming spikes arrive in different time bins: For the time-driven and the event-driven scheme the scenario is similar to **(A)**. For the compressed event-driven scheme the number of values that need to be updated equals the number of incoming synapses K so that $i = K$.

maximum inter-spike interval of any of the presynaptic neurons. Synapse code in this scheme is, however, only called in the event of a spike, leading to only $M = K \cdot T/I$ function calls per neuron, where T/I is the number of spikes passing a single synapse during the simulation of T time steps. The total amount of computations C of weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$ is of course unchanged with respect to the time-driven scheme; they are just split across fewer function calls ($C = M \cdot L = K \cdot T$). **Table 1** immediately shows the trade-off between memory consumption (length of history) and run time (number of function calls): the event-based scheme consumes more memory, but is faster than the time-driven scheme. Note that since a history of the membrane potential is stored anyway, this scheme is naturally applicable to connections with different delays. A further performance increase can be

TABLE 1 | Comparison of synapse update schemes.

	Time-driven	Event-driven	Event-driven & compression
History length L	1	I	i
Synapse function calls M	$K \cdot T$	$K \cdot T/I$	$K \cdot T/I$
Weight change computations C	$K \cdot T$	$K \cdot T$	T
History entry manipulations H	T	T	$K \cdot T/I \cdot i$

From the view point of a postsynaptic neuron, the table shows the maximal length of the history L, the number of function calls M of synapse code, the number of computations C of infinitesimal weight changes $\Delta W_{ij}(t^\alpha, t^{\alpha+1})$, and the number of history entry manipulations H for a simulation of T time steps, a uniform inter-spike interval I between spikes of a single presynaptic neuron, and an in-degree K for each neuron and no delays. For the event-driven compression scheme the entries show the length of the compressed history where i is the number of different spike times within the last inter-spike interval I.

achieved in plasticity rules, where weight changes only happen under certain conditions on V_i^*: if values $\Delta W_{ij}(t^\alpha, t^{\alpha+1}) \neq 0$ are rare, a non-continuous history can be stored. In such a scenario, time stamps need to be stored alongside the membrane potential to enable synapses to read out the correct time intervals (see section 3.2.3).

3.1.3. Event-Driven Compression

The event-driven compression scheme is a modified event-driven scheme that makes use of the fact that for a specific class of plasticity rules the integrated time trace of the membrane potential V_i^* can be used to advance the weight update of all incoming synapses, see section 2.3.1. Therefore, the time trace of V_i^* stored in the postsynaptic neuron only needs to extend back to the last incoming spike (denoted by t_{LI} in **Figure 4C**). This way the history of V_i^* is always short, as the total rate of incoming spikes is high in physiological network states. Due to the dependence of the weight update on the time of the last spike that crossed the synapse, the postsynaptic neuron stores the compressed history of length $L = i$, where i is the number of different spike times within the last inter-spike interval I (**Figure 5**). The compressed history is consequently never larger than the history length $L = I$ of the ordinary event-driven scheme (**Figure 5B**). For synchronous spikes where the last presynaptic spike time is the same for all synapses, the compressed history, however, contains only one entry (**Figure 5A**). Still, synapse code is executed at every spike event, giving rise to $M = K \cdot T/I$ function calls. The full membrane potential trace of length T is effectively only integrated once, amounting to in total $C = T$ infinitesimal weight change computations that are performed in batches in between any two incoming spike events (**Table 1**). The price for this is that history updates are more expensive: instead of appending a single entry in each time step, at each spike event the full compressed history is updated, giving rise to in total $H = M \cdot i = K \cdot T \cdot i/I$ history entry manipulations, as opposed to $H = T$ in the time- and ordinary event-driven schemes (**Table 1**). In practice, infinitesimal weight change computations are, however, often more costly than history updates, such that the compression algorithm achieves a performance increase (see section 3.4).

Finally, a drawback of the event-driven compression is that it relies on the fact that all synapses use the same processed membrane potential V_i^*. For distributed delays, $\Delta W_i(t_{LS}, T)$ has a dependence on the presynaptic neuron j via $V_i^*(t - d_j)$. In this case, a separate compressed history needs to be stored for every different delay of connections to the neuron.

3.2. Reference Implementation in Network Simulator With Event-Based Synapse Updates

This section describes the implementation of two example voltage-based plasticity rules by Clopath et al. (2010) and Urbanczik and Senn (2014) in a spiking neural network simulator that employs a time-driven update of neurons and an event-based update of synapses. While the naming conventions refer to our reference implementation in the simulation software NEST, the algorithms and concepts presented below are portable to other parallel spiking network simulators.

The Clopath and Urbanczik-Senn rule are chosen as widely used prototypical models of voltage-based plasticity. The differences in the two rules help to exemplify the advantages and disadvantages of the algorithms discussed in section 3.1. As originally proposed, they are implemented here for two different types of neuron models, Adex and Hodgkin-Huxley point-neurons for the Clopath rule (`aeif_psc_delta_clopath` and `hh_psc_alpha_clopath`) and two-compartment Poisson neurons (`pp_cond_exp_mc_urbanczik`) for the Urbanczik-Senn rule. Extensions to multiple dendritic compartments in the latter case are straight forward. Our implementation of `aeif_psc_delta_clopath` follows the reference implementation on ModelDB which introduced a clamping of the membrane potential after crossing the spiking threshold to mimic an action potential. Details can be found in **Appendix Section 5.2**.

The plasticity rules differ in the state variable that is being stored and its interpretation. For the Clopath rule, the stored variable is a thresholded and filtered version of the membrane potential that takes into account characteristics of membrane potential evolution within cells in the vicinity of spike events. The restriction to temporal periods around spikes suggests to implement a history that is non-continuous in time. In contrast, the Urbanczik-Senn rule uses the dendritic membrane potential to predict the somatic spiking; the resulting difference is taken as an error signal that drives learning. This error signal never vanishes and thus needs to be stored in a time-continuous history.

Finally, the proposed infrastructure for storing both continuous and non-continuous histories is generic so that it can also be used and extended to store other types of signals such as external teacher signals.

3.2.1. Exchange of Information Between Neurons and Synapses

The implementation of voltage-based plasticity rules in NEST follows the modular structure of NEST, key part of which is the separation between neuron and synapse models. This separation makes it easy for a newly added neuron model to be compatible with existing synapse models and vice versa. A downside is that information, such as values of parameters and state variables, is encapsulated within the respective objects. Simulations in NEST employ a hybrid parallelization scheme: OpenMP threads are used for intra node parallelization and the Message Passing Interface (MPI) for inter node communication. In parallel simulations, synapses are located at the same MPI process as the postsynaptic neurons (Morrison et al., 2005). Thereby, no communication between MPI processes is needed for the required transfer of information between postsynaptic neurons and synapses to compute weight changes of connections and only one spike needs to be communicated by a given source neuron for all target neurons living on the same MPI process.

The model of STDP requires synapses to access spike times of postsynaptic neurons. In order to provide a standardized transfer of this information across all neuron models that support STDP, in recent years the so-called `Archiving_Node` has been introduced as a parent class of the respective neuron models (Morrison et al., 2007a). It provides member functions to store and access spike histories. If a neuron model supports STDP, it only needs to be a child of `Archiving_Node` and contain one additional line of code, namely a call of the function `set_spiketime()`, which stores the time of outgoing spike events. We here extended this framework for voltage-based plasticity rules and enhanced the functionality of the archiving node by the member functions `write_history()`, `get_history()`, and `compress_history()` to additionally store, read out and manipulate voltage traces or other continuous signals (for Details, see **Appendix Section 5.3**). To avoid overhead for simulations with only STDP synapses, we introduced two child classes of `Archiving_Node`, `Clopath_Archiving_Node`, and `Urbanczik_Archiving_Node`, that each provide containers and functions for the specific histories required for the two plasticity rules. Neuron models that support the respective synapse model then derive from the child classes instead of the root level archiving node.

3.2.2. Delays and Min_delay Communication

All synapses implemented in NEST are so far purely event-driven. To assess the performance of the time-driven update scheme of synapses with voltage-based plasticity, we also implemented a time-driven version of the Clopath and Urbanczik-Senn synapse. Spiking network simulators exploit the delay of connections to reduce communication between compute processes (Morrison et al., 2005): Instead of sending each spike individually, spikes are buffered and sent in a batch after a certain period. The length of this period, the `min_delay`, corresponds to the minimal delay of all connections in the network. The buffering of spikes within this period is possible because the earliest time point that a spike at time t_S can affect the postsynaptic membrane potential is at $t = t_S + \text{min\_delay}$. In between t_S and t neurons are decoupled such that their state variables can be propagated forward in time independent of each other and in a batch (Morrison and Diesmann, 2008).

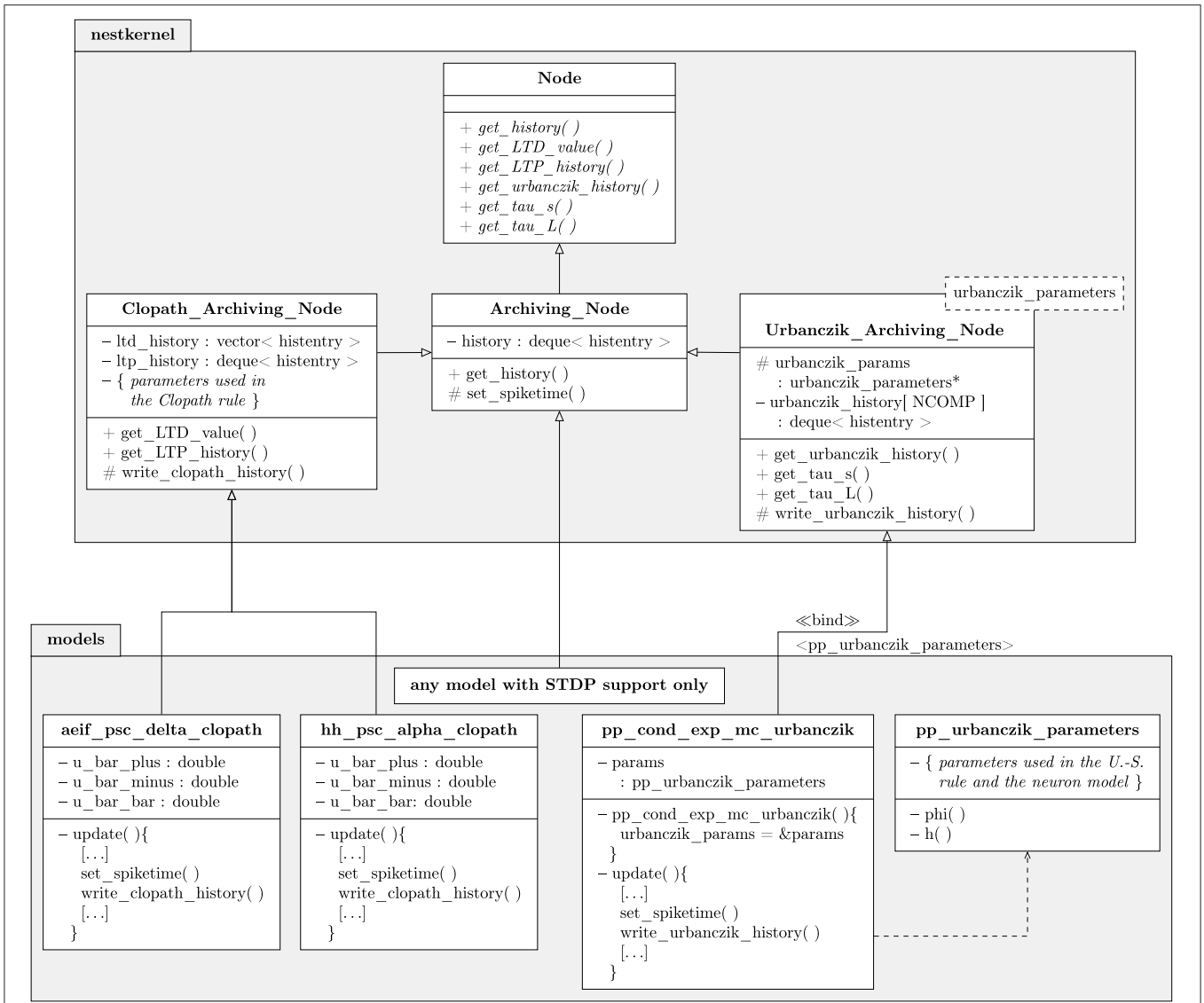


FIGURE 6 | Class diagram of NEST classes and functions. Simplified class diagram for embedding the Clopath **(Left)** and Urbanczik-Senn rule **(Right)** in the NEST infrastructure. The code is distributed across the `nestkernel` and neuron models. `nestkernel` contains the base class `Node` of all neurons models. Models that support ordinary STDP are derived from the `Archiving_Node`, models that can use the Clopath synapse (`aeif_psc_delta_clopath` and `hh_psc_alpha_clopath`) or Urbanczik-Senn synapse (`pp_cond_exp_mc_urbanczik`) are derived from the `Clopath_Archiving_Node` or the `Urbanczik_Archiving_Node`, respectively. The latter add the required functions for storing and managing the history of continuous quantities. The model `pp_cond_exp_mc_urbanczik` requires a helping class `pp_urbanczik_parameters` because the `Urbanczik_Archiving_Node` needs to access functions and parameters that are specific to the neuron model and therefore not located in the `Urbanczik_Archiving_Node` to keep its implementation more general.

We implemented the same `min_delay` update scheme for synapses, by imposing a function call to time-driven synapses in every `min_delay` period to update their synaptic weight. If `min_delay` equals the simulation step size h, this scheme corresponds to the scheme explained in section 3.1.1. Making use of the `min_delay` infrastructure of NEST speeds up simulations with time-driven synapses in the case $d > h$ as fewer function calls to synapses are needed (see section 3.4). In case of simulations with synaptic delays, the time-driven update scheme requires the storage of a history of the membrane potential of length `max_delay`.

Storing state variables in event-driven schemes is more complex as the history does not have a fixed length `max_delay`. Instead it needs to be dynamically extended and shortened. A long history can occupy a large amount of memory and its processing by the synapses becomes computationally expensive. Therefore, it is advantageous to optimize the way how information is stored and accessed and how entries that are no longer needed can be identified for deletion. For details of these optimizations in our NEST implementation, see **Appendix Section 5.3**.

As discussed in section 3.1.3, the event-based compression scheme relies on the fact that all synapses to one postsynaptic neuron employ the same V_i^*. This is not the case if delays of the corresponding connections are distributed. The compression scheme can therefore only be efficient if all delays have a fixed value. If spikes are processed and synapses are updated in a chronological order, then a well-defined segment of the history of V_i^* can be integrated and the compressed history can be updated. In NEST, spikes are, however, buffered within a period of `min_delay` before being sent and processed. Consequently, synapses are not necessarily updated in chronological order. Therefore, the event-based compression scheme can only be implemented in NEST in the case where delays equal the simulation time step. Future work may explore whether the latter restriction could be overcome by sorting all incoming spike events of a given postsynaptic neuron prior to delivery.

3.2.3. Specifics of Clopath Plasticity

We implement both an adaptive exponential integrate-and-fire neuron model (`aeif_psc_delta_clopath`) and a Hodgkin-Huxley neuron model (`hh_psc_alpha_clopath`) supporting Clopath plasticity. These implementations consider the filtered versions $\bar{u}_\pm$ of the membrane potential as additional state variables of the neuron. Thereby, they can be included in the differential equation solver of the neurons to compute their temporal evolution. Parameters of $\kappa_\pm$ consequently need to be parameters of the neuron object rather than the synapse. The same is true for the values of $\theta_\pm$; they are used in the neuron to determine whether $V^*_{i,\mathrm{LTP}}$ and $V^*_{i,\mathrm{LTD}}$ evaluate to zero, which systematically happens due to the Heaviside functions in their definitions.

The LTD mechanism is convenient to implement within the event-driven framework: when the synapse is updated at time t, it reads the values $\bar{u}_-\left(t-d\right)$ and θ_- from its target and computes the new weight. Here, d denotes the dendritic delay of the connection that accounts for the time it takes to propagate somatic membrane potential fluctuations to the synapse. The archiving node contains a cyclic buffer, also called ring buffer, that stores the history of $\bar{u}_-$ for the past `max_delay` time steps so that the synapse can access a past value of this quantity. Consequently, the LTD history is always short and can be forgotten in a deterministic fashion.

The computation of the weight change due to LTP requires the evaluation of the integral over $V^*_{i,\mathrm{LTP}}(t)$. The latter is stored in the archiving node as a vector whose elements are objects that contain three values: the corresponding time t, the value of $V^*_{i,\mathrm{LTP}}$ and an access counter that initially is set to zero.

3.2.3.1. Time-Driven Update

For simulations with homogeneous delays equal to the simulation time step, the history of $V^*_{i,\mathrm{LTP}}$ always contains only a single value as it is read out in every time step by all synapses. For larger delays, the history is of length `max_delay`, and each synapse reads out a segment of length `min_delay`, increasing the access counter of the corresponding entries by one. For the last synapse that requests a certain segment, the access counter then equals the in-degree K, which is the criterion to delete the corresponding entries from the history. Although for simplicity done in our reference implementation, the time-driven scheme does not require us to store the time stamp t of each history entry. The overhead of this additional number is, however, negligible.

3.2.3.2. Event-Driven Update

In event-driven schemes, the history of $V^*_{i,\mathrm{LTP}}$ dynamically grows and shrinks depending on the spikes of presynaptic neurons. Since many values of $V^*_{i,\mathrm{LTP}}$ are zero, it is beneficial to only store the non-zero values. In this case, a time stamp of each entry is required to assign values of the non-continuous history of $V^*_{i,\mathrm{LTP}}$ to their correct times. In case of the uncompressed scheme, when a synapse j is updated at time t_S of a spike, it requests the part of the history between the last spike t_{LS} and the current spike t_S (minus the dendritic delay, see **Appendix Section 5.3**) from the archiving node. This history segment is then integrated in synapse j and used for its weight update. Each synapse thus integrates the history $V^*_{i,\mathrm{LTP}}$ anew (section 3.1.2). For the compressed scheme, the history of $V^*_{i,\mathrm{LTP}}$ is integrated between the last incoming spike at t_{LI} and the current spike at t_{S} inside the archiving node. Using this newly integrated time trace, the weight of synapse j is updated and the compressed history for all other last spike times is advanced. Afterwards the history of $V^*_{i,\mathrm{LTP}}$ is deleted. Thereby, $V^*_{i,\mathrm{LTP}}$ is only integrated once for all synapses.

In any case, the integrated history of $V^*_{i,\mathrm{LTP}}$ needs to be combined with the presynaptic spike trace s_j^*. The latter is easily computed analytically inside the synapse because it is an exponential decay of the corresponding value at the time of the last spike. At the end of the update process the trace is increased by τ_s^{-1} to account for the trace of the current spike, where τ_s is the time constant of the kernel κ_s.

3.2.4. Specifics of Urbanczik-Senn Plasticity

Following the original publication (Urbanczik and Senn, 2014), we implement a Poisson spiking neuron model (`pp_cond_exp_mc_urbanczik`) supporting Urbanczik-Senn plasticity. One peculiarity of this model is that the gain function ϕ that translates the membrane potential into a firing rate also enters the plasticity rule through V^*. Therefore ϕ as well as its parameters need to be known by the neuron and the synapse. Creating an additional helper class (`pp_urbanczik_parameters`) as a template argument for the corresponding archiving node (`Urbanczik_Archiving_Node`) and neuron model (`pp_cond_exp_mc_urbanczik`) solves this problem (**Figure 6**): it contains all parameters and functions required by both classes. As explained in section 2.5, the representation (17) is more beneficial for implementing the Urbanczik-Senn rule than that of (16). The first two integrals in (17) only extend from t to T; history entries for times smaller than t are not needed and can be deleted after the corresponding update. The dependence on the full history back until 0 arising from the convolution with κ is accumulated in the last term in (17), which the synapse computes with the help of storing one additional value $I_2\left(0,t\right)$. At the end of a weight update this value is overwritten by the new value $I_2\left(0,T\right)=e^{-\frac{T-t}{\tau_\kappa}}I_2\left(0,t\right)+I_2\left(t,T\right)$ which is then used in the next update. Either the synapse (time- and event-driven

update) or the archiving node (event-driven compression) compute the two integrals I_1 and I_2 but in all cases the archiving node stores the history of $V_i^*(t)$.

3.3. Reproduction of Results in Literature

The reference implementation of the Clopath plasticity reproduces the results from Clopath et al. (2010) on the frequency dependence of weight changes in spike-pairing experiments and the emergence of bidirectional connections in small all-to-all connected networks (**Figure 7**). The setup of the spike-pairing experiment in **Figure 7A** consists of two neurons connected via a plastic synapse. The pre- and postsynaptic neuron are forced to spike with a time delay of Δt multiple times which leads to a change in synaptic weight that depends on the frequency of the spike pairs (**Figure 7B**). The setup of the small network is shown in **Figure 7C**. The weights of the plastic synapses within the recurrently connected excitatory population are initialized all to the same value. At the end of the simulation during which the network receives a time varying input, some pairs of neurons show strong bidirectional connections (**Figure 7D**). See **Appendix Section 5.4** for details on the setup of both experiments as implemented in NEST.

The basic use of the Urbanczik-Senn rule in NEST is exemplified in **Figure 8** which shows the reproduction of a simple learning experiment from the original publication (Urbanczik and Senn, 2014). Here the neuron is supposed to transform spike patterns in the input to the dendritic compartment into a sinusoidal modulation of the somatic membrane potential. This target potential is determined by an external teaching signal during learning. Via minimizing the error between the dendritic prediction of the somatic membrane potential and the actual somatic membrane potential, weights of dendritic synapses are organizing such that the neuron can produce the desired membrane potential. There is, however, no stop-learning region in the Urbanczik-Senn rule (for a modified version, see Cartiglia et al., 2020): The error never vanishes completely which causes weights to keep changing despite an overall good approximation of the target signal. Details of the experiment and NEST setup can be found in **Appendix Section 5.6**.

3.4. Performance of the Reference Implementations

3.4.1. Clopath Plasticity

In order to evaluate the performance of the implementation of the Clopath rule in NEST, in a weak-scaling setup, we simulate excitatory-inhibitory networks of increasing size, but fixed in-degree K. As we expect the performance to critically depend on the number of synapses, we examine two scenarios: a small in-degree $K = 100$, and a rather large in-degree $K = 5,000$. While the first case might be suitable for small functional networks, the latter in-degree represents a typical number for cortical networks. Further details on network and simulation parameters are given in **Supplementary Table 5**. As a reference, we also simulate the same network with STDP synapses, which require much fewer computations as they rely solely on spike times. To achieve the same network state, that is the same spikes, for the different connectivity rules, we impose the weights to stay constant across time by setting learning rates to zero. This way all computations for weight changes are being performed, but just not applied. This has the additional advantage that reasonable asynchronous irregular network states are simple to find based on predictions for static synapses (Brunel, 2000).

The Clopath rule has originally been proposed for connections without delays (Clopath et al., 2010). Therefore, we first evaluate its performance in this setting (delay equals simulation time step), which is, however, not the natural setting for a simulator like NEST that makes use of delays to speed up communication between compute processes. The first observation is that, as expected, simulations with Clopath synapses are slower than those with ordinary STDP (**Figure 9**). Given the update of synapses in every simulation step, the time-driven scheme for Clopath synapses is much slower than the event-driven scheme (**Figure 9A**). The difference becomes larger the more synapses there are (**Figure 9B**). Introducing a delay leads to fewer function calls to synapses (once every `min_delay`) and therefore increases the speed of the time-driven scheme (**Figure 9C**). Its simulation times, however, remain much above the event-driven scheme. This comparison illustrates the benefit of event-driven updates for Clopath synapses.

How does compression of the history change the picture? As discussed in section 3.1.3, compression has the advantage of not integrating the membrane potential history for each synapse separately. A downside of the event-based compression is that it requires storing one history entry for each last spike time of presynaptic neurons. For large in-degrees, this history is therefore longer than the history of $V^*_{i,\mathrm{LTP}}$, which we implemented as non-continuous for the Clopath rule. Consequently, the event-based compression scheme only outperforms the ordinary event-driven scheme for small in-degrees (**Figure 9A**), but not for large in-degrees (**Figure 9B**). Given that the compression can only be implemented in NEST for connections with delay equal to the resolution of the simulation (see section 3.2.2), the method of choice is therefore the ordinary event-driven scheme (section 3.1.2). Although a bit slower, its run-time is on the same order of magnitude as the ordinary STDP synapse, with similar weak-scaling behavior (**Figure 10**). The additional computations with respect to STDP result in a constant overhead.

Another advantage of having short non-continuous histories is that searching the history at readout is fast. A simple linear iteration scheme is therefore even faster than a binary search (**Figure 9D**) because the latter search requires an additional list of presynaptic spike times (see **Appendix Section 5.3**) which is unnecessary overhead in this scenario. As a result the ordinary event-driven scheme with linear history iteration is the most general and efficient scheme and therefore integrated into NEST 2.20.1 (Jordan et al., 2019).

3.4.2. Urbanczik-Senn Plasticity

The Urbanczik-Senn rule, in its original version, does not account for delays in connections (Urbanczik and Senn, 2014). As for the Clopath rule, we therefore first evaluate its performance for connections with delays that equal the simulation time step. We compare the results to networks with ordinary STDP

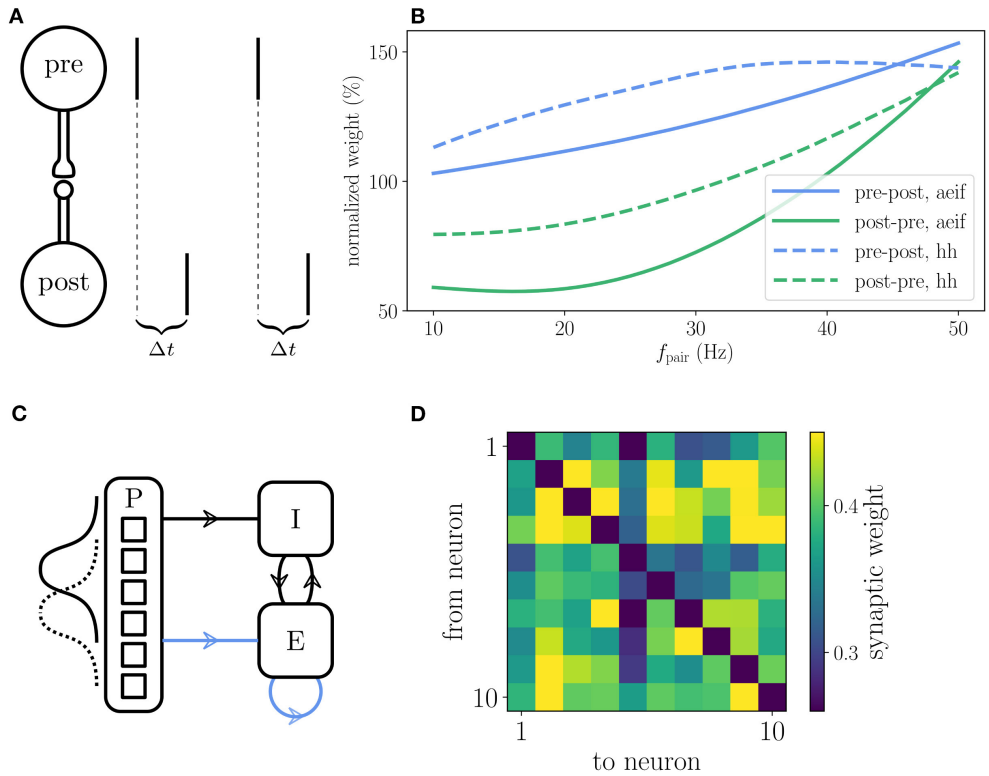


FIGURE 7 | Reproduction of results with Clopath rule. **(A)** Setup of the spike pairing experiment. Two neurons ("pre" and "post") that are connected by a plastic synapse receive input so that they spike one after another with a delay Δt. The change of the synaptic weight is computed according to the Clopath rule as a function of the frequency f_{pair} with which the spike pairs are induced. **(B)** Result of the spike pairing experiment. The relative change of the synaptic weight after five spike pairs as a function of f_{pair} is shown for two different neuron models (aeif: solid curves, Hodgkin-Huxley: dashed curves). The blue curves represent a setup where the postsynaptic neuron fires after the presynaptic one (pre-post, $\Delta t = 10$ ms) and the green curves represent the opposite case (post-pre, $\Delta t = -10$ ms). This panel corresponds to Figure 2B of Clopath et al. (2010). **(C)** Setup of the network that produces strong bidirectional couplings. The network consists of an inhibitory (I) and an excitatory (E) population which receive Poisson spike trains (P) as an external input. The firing rate of the latter is modulated with a Gaussian shape whose center is shifted every 100 ms. The external input connections to the excitatory population are plastic as well as the connections within the excitatory population (indicated by blue arrows). **(D)** Synaptic weights of the all-to-all connected excitatory neurons after the simulation of the network. Strong bidirectional couplings can be found, e.g., between neurons 2 and 3, 2 and 9, and 4 and 7. The setup of this experiment is similar to that shown in Figure 5 of Clopath et al. (2010). A more detailed description of the two experiments can be found in **Appendix Section 5.4**.

synapses, again setting all learning rates to zero to maintain the same network state across different types of plasticity. Naturally, the processing of the membrane potential information makes the Urbanczik-Senn plasticity less efficient to simulate than networks with ordinary STDP synapses (**Figure 11**). Note that the absolute numbers of simulation times are not directly comparable to simulations with Clopath plasticity (**Figure 9**) as network sizes are smaller here (**Supplementary Table 5**). Networks with small and large in-degrees behave qualitatively similar: given the long continuous history that needs to be stored and read out, the event-driven scheme does not significantly outperform the time-driven scheme (**Figures 11A,B**). In the network with small in-degree, the time-driven scheme is even slightly faster (**Figure 11A**). This behavior reverses for large in-degrees as the number of synapse calls grows stronger than the length of the history (**Figure 11B**). However, given that the length of the history is so critical in this rule, the compression algorithm can in both cases achieve a significant increase in performance (**Figures 11A,B**). This performance increase is larger the smaller the in-degree, as the compressed history becomes shorter (**Figure 11A**). Due to current NEST specifics (see section 3.2.2), the compression algorithm cannot be used in settings with delays that are larger than the simulation time step (**Figure 11C**): Here, as expected, the time-driven scheme becomes faster than in the $d = h$ case, but it is in general still comparable in performance to the event-driven scheme. The latter is therefore the method of choice for simulations with delayed connections; for zero-delay connections, the compression algorithm performs best. Whether the history readout is done via linear iteration or via computing positions of history entries has no significant impact on the performance (**Figure 11D**). Therefore, the simple linear iteration is integrated in NEST 2.20.1.

We furthermore employ a weak-scaling setup with excitatory-inhibitory networks of increasing size and fixed in-degree $K = 5,000$ (**Figures 12A,B**, and **Supplementary Table 5**). Apart

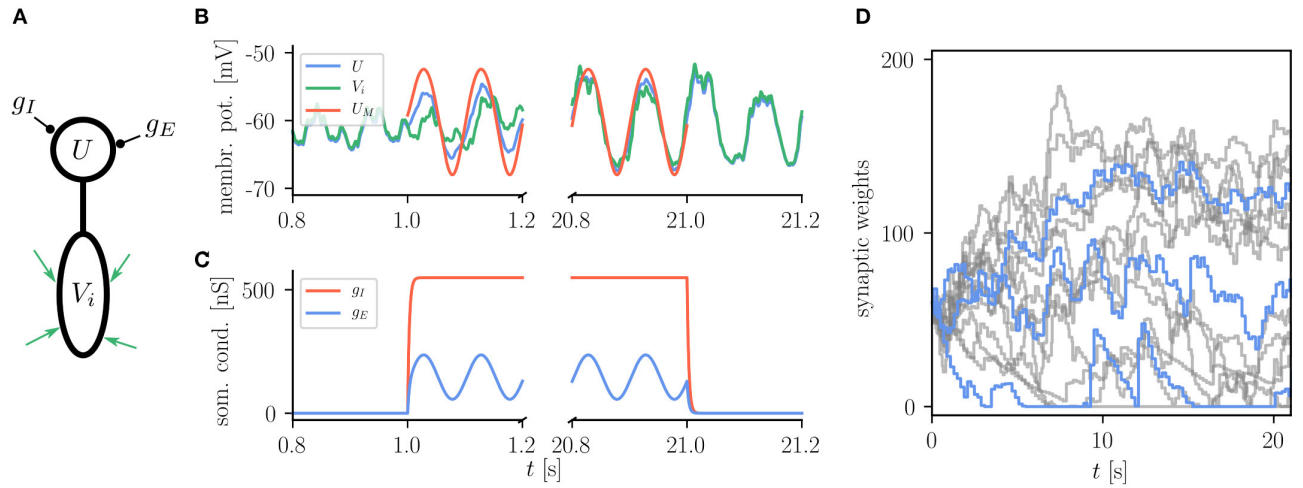


FIGURE 8 | Reproduction of results with Urbanczik-Senn rule. **(A)** Setup of a simple learning task using the Urbanczik-Senn plasticity rule. The somatic conductances g_I and g_E of a two-compartment neuron are modulated such that they induce a teaching signal with sinusoidal shape. The dendrite receives a repeating spike pattern as an input via plastic synapses (green arrows). **(B)** The synapses adapt their weights so that the somatic membrane potential U (blue) and the dendritic prediction V_i (green) follow the matching potential U_M (red) after learning. **(C)** Excitatory (g_E) and inhibitory (g_I) somatic conductances that produce the teaching signal. **(A,B)** correspond to Figure 1 of Urbanczik and Senn (2014). **(D)** Temporal evolution of the synaptic weights during learning. For the sake of better overview, only a subset of weights is shown (gray) with three randomly chosen time traces highlighted in blue. Synapses in NEST fulfill Dale's principle which means that a weight update cannot convert an excitatory into an inhibitory synapse and vice versa giving rise to the rectification at zero.

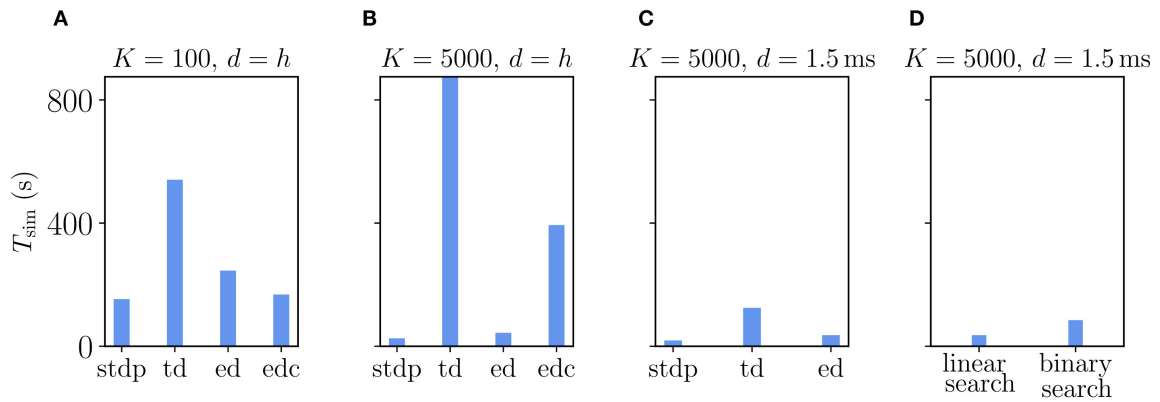


FIGURE 9 | Comparison of simulation times T_{sim} for excitatory-inhibitory networks with different implementations of the Clopath plasticity in NEST. Simulation times exclude network building and only account for updates of the dynamical state of the system. The following implementations are shown: "stdp": standard implementation of STDP synapse, "td": time-driven implementation of Clopath synapse, "ed": event-driven scheme as included in NEST 2.20.1, "edc": event-driven compression. **(A)** Network of size $N = 1.92 \cdot 10^6$ with small in-degree $K = 100$ and all synapses having a delay d equal to the resolution of the simulation $h = 0.1$ ms. **(B)** Network of size $N = 1.54 \cdot 10^5$ with large in-degree $K = 5,000$ and $d = h$. **(C)** Same network as in **(B)** but $d = 1.5$ ms (for $d > h$ "edc" not compatible with NEST, see section 3.1.3). In **(A–C)** both "ed" and "edc" use linear search of the history and access counters, see **Appendix Section 5.3**. **(D)** Comparison between "ed"-implementations using linear search and direct computation of the position, see **Appendix Section 5.3.0.2**. All simulations use 768 threads distributed over 32 compute nodes each running one MPI process. Further parameters as in **Supplementary Table 5**.

from a constant offset, the scaling of simulation time T_{sim} for updating neurons and synapses is similar for Urbanczik, static and STDP synapses. With increasing network size N and proportionally increasing number of MPI processes, T_{sim} rises only slightly (**Figure 12B**), indicating almost ideal weak-scaling behavior. The constant offset in T_{sim} is larger than for Clopath synapses as the Urbanczik-Senn rule requires longer histories of membrane potentials and a more extensive history management.

3.5. Conclusions

The analyses of the Clopath and the Urbanczik-Senn plasticity as prototypical examples for rules that rely on storage of discontinuous vs. continuous histories show that the former are much faster to simulate, in particular for large networks that require distributed computing. For discontinuous histories, the event-driven scheme is most generally applicable and efficient, which makes corresponding rules easy to integrate into modern simulators with event-based synapses. The performance gap

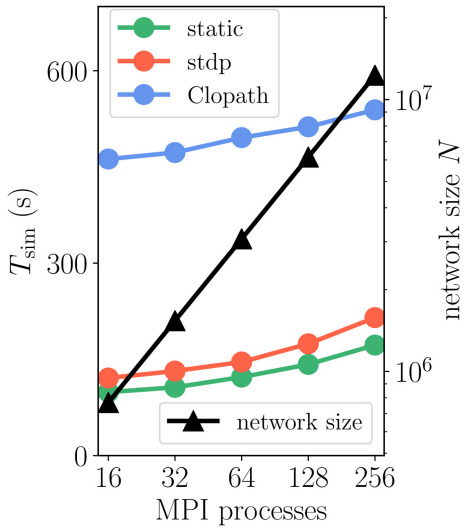


FIGURE 10 | Scaling of simulation time T_{sim} with network size for 2 *s* of biological time: Clopath plasticity. Weak scaling: computational resources (horizontal axis) increase proportionally to network size *N* (black curve and triangles, right vertical axis). Event-driven scheme for Clopath rule (blue) compared to static (green) and STDP synapse (red). Network and simulation parameters as in **Supplementary Table 5** with in-degree $K = 5,000$. For all simulations each compute node runs one MPI process with 24 threads.

between the different rules should be kept in mind in the design of new learning rules. Furthermore, it is worthwhile to test modifications of existing learning rules to decrease the amount of stored information.

For illustration, we here test a spike-based alternative to the original Urbanczik-Senn rule, where we replace the rate prediction $\phi(V_i(t))$ in V^* of (15) by a noisy estimate, which we generate by a non-homogeneous Poisson generator with rate $\phi(V_i(t))$, see **Appendix Section 5.7**. The prediction error then results in a comparison of somatic and dendritic spikes, s_i and s_i^{dend}, respectively; it is therefore purely based on point processes. In terms of storage and computations, the rule thereby becomes similar to ordinary STDP [cf. (5)]. This becomes apparent in the weak-scaling experiment in **Figure 12C**, which shows that the modification of the learning rule results in a speedup of a factor 10 to 30 arriving essentially at the same run time as the ordinary STDP rule.

When changing learning rules to improve the efficiency of an implementation, the question is in how far the modified rule, in our example including the noisy estimate of the dendritic prediction, still fulfills the functionality that the original rule was designed for. Generally, without control of the error any simulation can be made arbitrarily fast. Therefore, Morrison et al. (2007b) define efficiency as the wall-clock time required to achieve a given accuracy. We test in **Figure 13** whether the dynamics is still robust enough to achieve proper learning and function in the reproduced task of **Figure 8**. The learning works as well as in the original Urbanczik-Senn rule. However, given the simplicity of the chosen task, this result may not generalize to other more natural tasks. We leave a more detailed investigation of this issue to future studies. The basic exploration here, however, illustrates how taking into account the efficiency of implementations can guide future development of learning rules to make them practically usable for large-scale simulations of brain networks.

4. DISCUSSION

This work presents efficient algorithms to implement voltage-based plasticity in modern neural network simulators that rely on event-based updates of synapses (for a review, see Brette et al., 2007). This update scheme restricts function calls of synapse code to time points of spike events and thereby improves performance in simulations of biologically plausible networks, where spike events at individual synapses are rare and the total number of synapses is large compared to the number of neurons. While our framework has no restrictions on the postsynaptic voltage-dependence of the learning rule, a particular focus of this work is on those plasticity rules, where synapses rely on an extended history of membrane potentials and therefore continuous information of state variables of postsynaptic cells to update their strength. This dependence naturally suggests a time-driven update of synapses. Instead, we here propose an efficient archiving of voltage traces to enable event-based synapse updates and detail two schemes for storage, read out and post-processing of time-continuous or discontinuous information. We show their superior performance with respect to time-driven update both theoretically and with a reference implementation in the neural network simulation code NEST for the rules proposed in Clopath et al. (2010) and Urbanczik and Senn (2014).

Event-driven update schemes for voltage-based plasticity come at the expense of storing possibly long histories of a priori continuous state variables. Such histories not only require space in memory but they also affect the runtime of simulations, which we focus on here. The time spent for searching and post-processing the history to calculate weight updates increases with increasing length, and these operations have to be done for each synapse. Therefore, in addition to an ordinary event-driven scheme, we devised a compression scheme that becomes superior for long histories as occurring in the Urbanczik-Senn rule. In particular for networks with small in-degrees or synchronous spiking, the compression scheme results in a shorter history. It further reduces the total amount of computations for weight changes by partially re-using results from other synapses thereby avoiding multiple processing of the history. For short histories as occurring in the Clopath rule, the compression results in unnecessary overhead and an increase in history size as one entry per last presynaptic spike time needs to be stored instead of a discontinuous membrane potential around sparse postsynaptic spike events. We here, for simplicity, contrasted time- and event-driven update schemes. However, further work could also investigate hybrid schemes, where synapses are not only updated at spike events, but also on a predefined and coarse time grid to avoid long histories and corresponding extensive management. A similar mechanism is used in Kunkel et al. (2011) to implement a

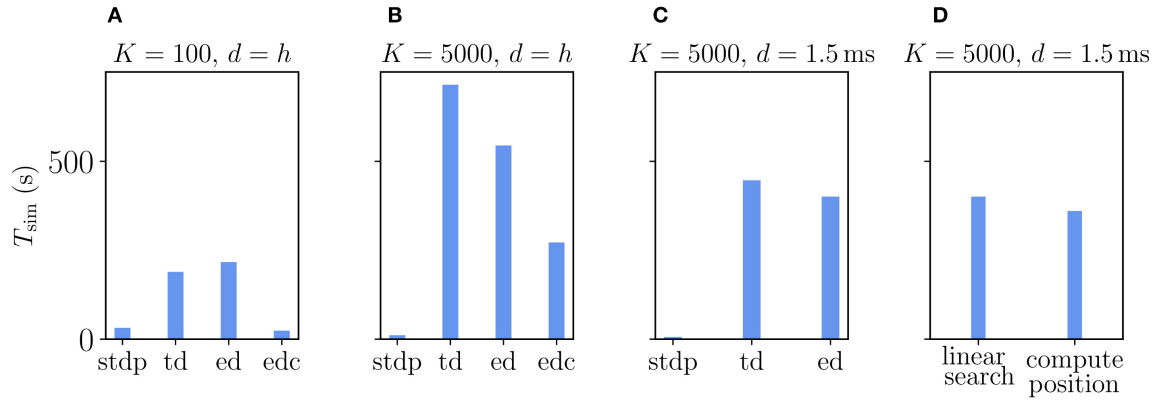


FIGURE 11 | Comparison of simulation times T_{sim} for excitatory-inhibitory networks with different implementations of the Urbanczik-Senn plasticity in NEST. The following implementations are shown: "stdp": standard implementation of STDP synapse in NEST, "td": time-driven implementation of Urbanczik-Senn synapse, "ed": event-driven scheme, edc": event-driven compression. **(A)** Network of size $N = 3.84 \cdot 10^5$ with small in-degree $K = 100$ and all synapses having a delay d equal to the resolution of the simulation $h = 0.1$ ms. **(B)** Network of size $N = 3.84 \cdot 10^4$ with large in-degree $K = 5,000$ and $d = h$. **(C)** Same network as in **(B)** but $d = 1.5$ ms (for $d > h$ "edc" not compatible with NEST, see section 3.1.3). In **(A–C)** both "ed" and "edc" use linear search of the history and the access counters, see **Appendix Section 5.3**. **(D)** Comparison between "ed"-implementations using linear search and direct computation of the position, see **Appendix Section 5.3.0.2**. All simulations use 768 threads distributed over 32 compute nodes each running one MPI process. Details on network parameters in **Supplementary Table 5**.

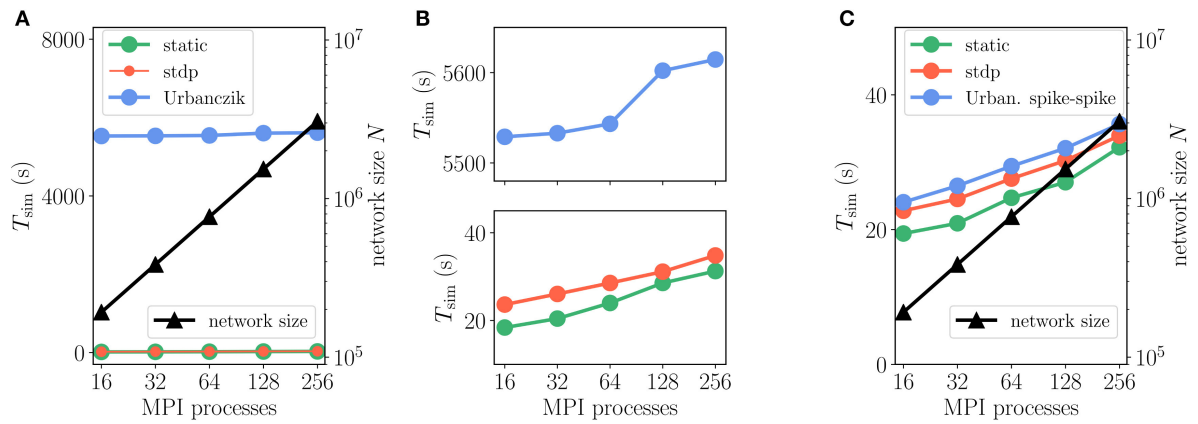


FIGURE 12 | Scaling of simulation times T_{sim} with network size for 2 *s* of biological time: Urbanczik-Senn plasticity. Same weak scaling as in **Figure 10**. **(A)** Event-driven Urbanczik-Senn rule (blue) compared to static (green) and STDP synapse (red). On the scale of the vertical axis the red curve (STDP synapses) falls on top of the green curve (static synapses), indicated by finer line width and marker size of the former. **(B)** Same simulation time data as in **(A)** but with a smaller range on the vertical axis. Upper panel: enlargement of Urbanczik-Senn data. Lower panel: enlargement of data for static and STDP synapses. **(C)** Spike-spike version of the Urbanczik-Senn rule compared to static and STDP synapse. Network and simulation parameters as in **Supplementary Table 5** with in-degree $K = 5,000$. For all simulations each compute node runs one MPI process with 24 threads.

normalization of synaptic weights. The corresponding technical details can be found in Kunkel (2015, ch. 5.2).

The storage and management of the history as well as complex weight change computations naturally reduce the performance of simulations with voltage-based plasticity in comparison to static or STDP synapses. The latter only require information on spike times which is much less data compared to continuous signals. Nevertheless, given that the Clopath rule is based on thresholded membrane potentials and consequently short, discontinuous histories, the performance and scaling of the event-driven algorithms is only slightly worse than for ordinary STDP. Time-driven implementations cannot employ this model feature and update weights also in time steps where no adjustment would be required, leading to significantly slower simulations. The performance gain of using event-driven schemes is less pronounced for the Urbanczik-Senn rule as, by design, histories are typically long. In this case, the compression scheme naturally yields better results in terms of runtime. Our own modification of the Urbanczik-Senn rule only requires storage of sparsely sampled membrane potentials, giving rise to the same performance as STDP. Generally, an algorithm is faster if it requires fewer computations. However, opportunities for vectorization and cache efficient processing,

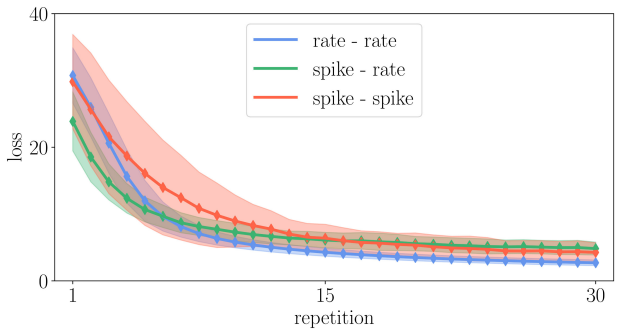


FIGURE 13 | Comparison of learning curves in the experiment described in **Appendix Section 5.6** for different variants of the Urbanczik-Senn plasticity rule. The loss is averaged over 128 trials of different input patterns. Solid curves denote the mean value and the shaded area the corresponding standard deviation of the loss.

outside of the scope of the present manuscript, may change the picture.

We here chose the Clopath and the Urbanczik-Senn rule as two prototypical models of voltage-based plasticity. While both rules describe a voltage dependence of weight updates, their original motivation as well as their specific form are different: The Clopath rule refines standard STDP models to capture biologically observed phenomena such as frequency dependence of weight changes (Sjöström et al., 2001). For this it is sufficient to take into account membrane potential traces in the vicinity of spike events, leading to storage of time-discontinuous histories in our implementation. In contrast, the Urbanczik-Senn rule is functionally inspired by segregating dendritic and somatic compartments of cells and using the difference between somatic output and dendritic prediction as a teacher signal for dendritic synapses. The teacher signal is by construction never vanishing, imposing the need to store a time-continuous history. The original publications of both rules had a great and long-lasting impact on the field. The Clopath rule has been used in a variety of studies (Clopath and Gerstner, 2010; Ko et al., 2013; Litwin-Kumar and Doiron, 2014; Sadeh et al., 2015; Bono and Clopath, 2017; Maes et al., 2020), partly in modified versions which are, however, still compatible with the here presented simulation algorithms. The same holds for the Urbanczik-Senn rule (Brea et al., 2016; Sacramento et al., 2018).

The current implementation, which is published and freely available in NEST 2.20.1, supports an adaptive exponential integrate-and-fire and a Hodgkin-Huxley neuron model for the Clopath rule. The former is used in the original publication (Clopath et al., 2010) and the latter appears on ModelDB (Hines et al., 2004) in code for the Clopath rule for the NEURON simulator (Hines and Carnevale, 2001). For the Urbanczik-Senn rule, NEST currently supports the two-compartment Poisson model neuron of the original publication (Urbanczik and Senn, 2014). A three-compartment version as used in Sacramento et al. (2018) or other models are straight forward to integrate into the current simulation framework. However, with voltage-based plasticity rules, borders between neurons and synapses become blurred as these rules often depend on specifics of the employed neuron models rather than only spike times as for standard STDP. Consequently, archiving nodes might need to have specific functionalities, which, in light of the zoo of existing neuron models, could easily lead to a combinatorial explosion of code. These problems can in future be overcome with automatic code generation using NESTML that only creates and compiles code that is needed for the specified model simulations (Plotnikov et al., 2016).

While the here presented implementation refers to the neural network simulator NEST (Gewaltig and Diesmann, 2007), the proposed algorithms and simulation infrastructure are compatible with any network simulator with event-driven update of synapses, such as for example, NEURON (Lytton et al., 2016, cf. ch. 2.4) and Brian2 (Stimberg et al., 2014). Furthermore, applicability is not restricted to the Clopath and Urbanczik-Senn rule, but the framework can be adapted to any other learning rule that relies on state variables of postsynaptic neurons. State variables hereby not only encompass membrane potentials such as for example, in the LCP rule by Mayr and Partzsch (2010), the Convallis rule by Yger and Harris (2013), the voltage-triple rule by Brea et al. (2013), the MPDP rule by Albers et al. (2016), the voltage-gated learning rules by Brader et al. (2007), Sheik et al. (2016), Qiao et al. (2015), Diederich et al. (2018), and Cartiglia et al. (2020), or the branch-specific rule by Legenstein and Maass (2011), but also, for example, firing rates of stochastic neuron models or rate models (Brea et al., 2016; Sacramento et al., 2018), or other learning signals (Neftci et al., 2017; Bellec et al., 2020). The infrastructure in NEST allows for the storage of time-continuous and discontinuous histories and therefore poses no restrictions on the dependence of the learning rule on the postsynaptic state variables. The here developed machinery could be also used to store external teacher signals that are provided to model neurons by stimulation devices mimicking brain or environmental components not explicitly part of the model. Since synapses are located at the compute process of the postsynaptic neuron, readout of state variables from presynaptic neurons comes with large costs in simulations on distributed computing architectures and is therefore not considered here. Due to specifics of the present NEST code in spike delivery, the event-driven compression proposed here is only applicable in NEST for delays that equal the simulation time step. Such a restriction can be readily overcome in a simulation algorithm that performs a chronological update of synapses.

In general, one has to distinguish two types of efficiency in the context of simulating plastic networks: Firstly, the biological time it takes the network to learn a task by adapting the weights of connections. Secondly, the wall-clock time it takes to simulate this learning process. Both times crucially depend on the employed plasticity rule. In this study, we focus on the wall-clock time and argue that this can be optimized by designing learning rules that require storing only minimal information on postsynaptic state variables. Ideally, the plasticity rule contains unfiltered presynaptic or postsynaptic spike trains to reach the same performance as in ordinary STDP simulations. This amounts to synapses requiring postsynaptic state variables only at the time of spike

events. The Clopath and Urbanczik-Senn rule capture the dependence of synaptic weights on the postsynaptic membrane potential in a phenomenological manner. The dependence on the voltage history observed in biological synapses (Artola et al., 1990; Ngezahayo et al., 2000) is an indirect effect mediated by receptors and channels with voltage-dependent dynamics (see Clopath et al., 2010, and references therein). Modeling these complex dynamics explicitly and evolving corresponding state variables in the postsynaptic neurons would enable a readout of these quantities only at spike times and thereby remove the need to maintain explicit histories. If phenomenological rules, however, need to capture the pre- and post-spike dynamics of postsynaptic membrane potentials explicitly, it is beneficial to make use of thresholds as in the example of the Clopath rule as these yield short, time-discontinuous histories. Reducing the amount of information available for synapses to adjust their weights can in general slow down the learning. We present a modification of the Urbanczik-Senn rule where the dendritic prediction of the somatic firing contains an additional sampling step with Poisson spike generation. This modification significantly reduces the simulation time. For the here presented simple task, learning speed is largely unaffected, but generally a performance decrease is to be expected when error signals become more noisy. Therefore, there is a trade-off between learning speed and simulation speed, which should be considered in the design process of new learning rules. Complex voltage-based plasticity rules have been simplified and turned into voltage-gated learning rules to make them compatible with event-based synapse updates: Cartiglia et al. (2020) propose a modification of the Urbanczik-Senn rule underlying the model in Sacramento et al. (2018). This simplification only requires postsynaptic membrane potentials at the time of spike events, which makes the rule much more efficient to simulate and applicable to neuromorphic hardware. Bono and Clopath (2017) simplify the Clopath rule in an analogous fashion to allow for its event-based simulation in the spiking network simulator Brian2, see documentation at https://brian2.readthedocs.io/en/stable/examples/frompapers.Clopath_et_al_2010_homeostasis.html (Stimberg et al., 2014). Our general framework supports systematic testing of such simplifications in terms of simulation performance and functionality.

For the plasticity rules by Clopath et al. (2010) and Urbanczik and Senn (2014), we present a highly scalable reference implementation that is published and freely available in NEST 2.20.1. The parallelism of the NEST implementation enables simulations of plastic networks of realistic size on biologically plausible time scales for learning. The field of computational neuroscience recently entered a new era with the development of large-scale network models (Markram et al., 2015; Schmidt et al., 2018; Billeh et al., 2020). Emulating the dynamics of cortical networks, such models are so far restricted to static connections. We here provide simulation algorithms for plasticity mechanisms that are required for augmenting such complex models with functionality. It is our hope that incorporating both biologically and functionally inspired plasticity models in a single simulation engine fosters the exchange of ideas between communities toward the common goal of understanding system-level learning in the brain.

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: Stapmanns et al. (2021).

CODE AVAILABILITY

The reference implementation for the event-driven update scheme of synapses with Clopath and Urbanczik-Senn plasticity was reviewed by the NEST initiative and is publicly available in NEST 2.20.1. The PyNEST code for model simulations and Python scripts for the analysis and results are fully available at: Stapmanns et al. (2021).

AUTHOR CONTRIBUTIONS

JS and JH wrote the simulation code, the plotting scripts, and performed the NEST simulations for the HPC performance measurements. JS and DD worked out the details of the theoretical analysis of the different algorithms. JS was supervised by MH, MD, and DD. JH was supervised by MB. All authors jointly did the conceptual work and wrote the paper.

FUNDING

This project has received funding from the Helmholtz Association Initiative and Networking Fund under project number SO-092 (Advanced Computing Architectures, ACA) and the European Union's Horizon 2020 research and innovation programme under grant agreements No 785907 (HBP SGA2) and No 945539 (HBP SGA3). Partly supported by the Helmholtz young investigator group VH-NG-1028. All network simulations carried out with NEST (http://www.nest-simulator.org). This work was supported by the Jülich-Aachen Research Alliance Center for Simulation and Data Science (JARA-CSD) School for Simulation and Data Science (SSD).

ACKNOWLEDGMENTS

We thank Claudia Clopath and Wulfram Gerstner for explaining details of their reference implementation and the underlying biological motivation. Moreover, we thank Hedyeh Rezaei and Ad Aertsen for suggesting to implement the Clopath rule in NEST and Charl Linssen, Alexander Seeholzer, Renato Duarte for carefully reviewing our implementation. Finally we thank Walter Senn, Mihai A. Petrovici, Laura Kriener, and Jakob Jordan for fruitful discussions on the Urbanczik-Senn rule and our proposed spike based version. We further gratefully acknowledge the computing time on the supercomputer JURECA (Jülich Supercomputing Centre , 2015) at Forschungszentrum Jülich

granted by firstly the JARA-HPC Vergabegremium (provided on the JARA-HPC partition, jinb33) and secondly by the Gauss Centre for Supercomputing (GCS) (provided by the John von Neumann Institute for Computing (NIC) on the GCS share, hwu12).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2021.609147/full#supplementary-material

REFERENCES

Albers, C., Westkott, M., and Pawelzik, K. (2016). Learning of precise spike times with homeostatic membrane potential dependent synaptic plasticity. *PLoS ONE* 11:e0148948. doi: 10.1371/journal.pone.0148948

Artola, A., Bröcher, S., and Singer, W. (1990). Different voltage dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex. *Nature* 347, 69–72. doi: 10.1038/347069a0

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* 11:3625. doi: 10.1038/s41467-020-17236-y

Bi, G., and Poo, M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Billeh, Y. N., Cai, B., Gratiy, S. L., Dai, K., Iyer, R., Gouwens, N. W., et al. (2020). Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex. *Neuron* 106, 388–403.e18. doi: 10.1016/j.neuron.2020.01.040

Bono, J., and Clopath, C. (2017). Modeling somatic and dendritic spike mediated plasticity at the single neuron and network level. *Nat. Commun.* 8, 1–17. doi: 10.1038/s41467-017-00740-z

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* 19, 2881–2912. doi: 10.1162/neco.2007.19.11.2881

Brea, J., Gaal, A. T., Urbanczik, R., and Senn, W. (2016). Prospective coding by spiking neurons. *PLoS Comput. Biol.* 12:e1005003. doi: 10.1371/journal.pcbi.1005003

Brea, J., Senn, W., and Pfister, J.-P. (2013). Matching recall and storage in sequence learning with spiking neural networks. *J. Neurosci.* 33, 9565–9575. doi: 10.1523/JNEUROSCI.4098-12.2013

Brette, R., and Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.* 94, 3637–3642. doi: 10.1152/jn.00686.2005

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* 23, 349–398. doi: 10.1007/s10827-007-0038-6

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comput. Neurosci.* 8, 183–208. doi: 10.1023/a:1008925309027

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

Cartiglia, M., Haessig, G., and Indiveri, G. (2020). "An error-propagation spiking neural network compatible with neuromorphic processors," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Genova: IEEE), 84–88.

Clopath, C., Büsing, L., Vasilaki, E., and Gerstner, W. (2010). Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nat. Neurosci.* 13, 344–352. doi: 10.1038/nn.2479

Clopath, C., and Gerstner, W. (2010). Voltage and spike timing interact in stdp–a unified model. *Front. Synap. Neurosci.* 2:25. doi: 10.3389/fnsyn.2010.00025

Davison, A., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., et al. (2008). PyNN: a common interface for neuronal network simulators. *Front. Neuroinformatics* 2:11. doi: 10.3389/neuro.11.011.2008

D'Haene, M., Hermans, M., and Schrauwen, B. (2014). Toward unified hybrid simulation techniques for spiking neural networks. *Neural Comput.* 26, 1055–1079. doi: 10.1162/NECO_a_00587

Diederich, N., Bartsch, T., Kohlstedt, H., and Ziegler, M. (2018). A memristive plasticity model of voltage-based stdp suitable for recurrent bidirectional neural networks in the hippocampus. *Sci. Rep.* 8, 1–12. doi: 10.1038/s41598-018-27616-6

Djurfeldt, M., Davison, A. P., and Eppler, J. M. (2014). Efficient generation of connectivity in neuronal networks from simulator-independent descriptions. *Front. Neuroinformatics* 8:43. doi: 10.3389/fninf.2014.00043

Djurfeldt, M., Hjorth, J., Eppler, J. M., Dudani, N., Helias, M., Potjans, T. C., et al. (2010). Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics* 8, 43–60. doi: 10.1007/s12021-010-9064-z

Eppler, J. M., Helias, M., Muller, E., Diesmann, M., and Gewaltig, M. (2009). PyNEST: a convenient interface to the NEST simulator. *Front. Neuroinformatics* 2:12. doi: 10.3389/neuro.11.012.2008

Friedmann, S., Schemmel, J., Grübl, 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

Galluppi, F., Lagorce, X., Stromatias, E., Pfeiffer, M., Plana, L. A., Furber, S. B., et al. (2015). A framework for plasticity implementation on the spinnaker neural architecture. *Front. Neurosci.* 8:429. doi: 10.3389/fnins.2014.00429

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature* 383, 76–78. doi: 10.1038/383076a0

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics. From single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press.

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

Hahne, J., Helias, M., Kunkel, S., Igarashi, J., Bolten, M., Frommer, A., et al. (2015). A unified framework for spiking and gap-junction interactions in distributed neuronal network simulations. *Front. Neuroinformatics* 9:22. doi: 10.3389/fninf.2015.00022

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv [Preprint]*. arXiv:1412.5567v2.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. New York, NY: John Wiley & Sons.

Hines, M. L., and Carnevale, N. T. (2001). NEURON: a tool for neuroscientists. *Neuroscientist* 7, 123–135. doi: 10.1177/107385840100700207

Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. (2004). ModelDB: A database to support computational neuroscience. *J. Comput. Neurosci.* 17, 7–11. doi: 10.1023/B:JCNS.0000023869.22017.2e

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 1527–1554. doi: 10.1162/neco.2006.18.7.1527

Jordan, J., Helias, M., Diesmann, M., and Kunkel, S. (2020a). Efficient communication in distributed simulations of spiking neuronal networks with gap junctions. *Front. Neuroinformatics* 14:12. doi: 10.3389/fninf.2020.00012

Jordan, J., Ippen, T., Helias, M., Kitayama, I., Sato, M., Igarashi, J., et al. (2018). Extremely scalable spiking neuronal network simulation code: from laptops to exascale computers. *Front. Neuroinformatics* 12:2. doi: 10.3389/fninf.2018.00002

Jordan, J., Mørk, H., Vennemo, S. B., Terhorst, D., Peyser, A., Ippen, T., et al. (2019). *NEST 2.18.0*. doi: 10.5281/zenodo.2605422

Jordan, J., Schmidt, M., Senn, W., and Petrovici, M. A. (2020b). Evolving to learn: discovering interpretable plasticity rules for spiking networks. *arXiv [Preprint]*. arXiv:2005.14149v3.

Jülich Supercomputing Centre (2015). *JUQUEEN: IBM Blue Gene/Q® Supercomputer System at the Jülich Supercomputing Centre*.

Ko, H., Cossell, L., Baragli, C., Antolik, J., Clopath, C., Hofer, S. B., et al. (2013). The emergence of functional microcircuits in visual cortex. *Nature* 496, 96–100. doi: 10.1038/nature12015

Krishnan, J., Porta Mana, P., Helias, M., Diesmann, M., and Di Napoli, E. A. (2017). Perfect detection of spikes in the linear sub-threshold dynamics of point neurons. *Front. Neuroinformatics* 11:75. doi: 10.3389/fninf.2017.00075

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural information Processing Systems* (Tahoe, CA), 1097–1105.

Kumar, N. M., and Gilula, N. B. (1996). The gap junction communication channel. *Cell* 84, 381–388. doi: 10.1016/S0092-8674(00)81282-9

Kunkel, S. (2015). *Simulation technology for plastic neuronal networks on high- performance computers* (Doctoral dissertation). FreiDok plus Universitätsbibliothek Freiburg, University of Freiburg, Freiburg, Germany. doi: 10.6094/UNIFR/10419

Kunkel, S., Diesmann, M., and Morrison, A. (2011). Limits to the development of feed-forward structures in large recurrent neuronal networks. *Front. Comput. Neurosci.* 4:160. doi: 10.3389/fncom.2010.00160

Lecun, Y. (1985). "Une procedure d'apprentissage pour reseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks)," in *Proceedings of Cognitiva 85* (Paris), 599–604.

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

Legenstein, R., and Maass, W. (2011). Branch-specific plasticity enables self-organization of nonlinear computation in single neurons. *J. Neurosci.* 31, 10787–10802. doi: 10.1523/JNEUROSCI.5684-10.2011

Litwin-Kumar, A., and Doiron, B. (2014). Formation and maintenance of neuronal assemblies through synaptic plasticity. *Nat. Commun.* 5, 1–12. doi: 10.1038/ncomms6319

Lytton, W. W., Seidenstein, A. H., Dura-Bernal, S., McDougal, R. A., Schürmann, F., and Hines, M. L. (2016). Simulation neurotechnologies for advancing brain research: parallelizing large networks in NEURON. *Neural Comput.* 28, 2063–2090. doi: 10.1162/neco_a_00876

Maes, A., Barahona, M., and Clopath, C. (2020). Learning spatiotemporal signals using a recurrent spiking network that discretizes time. *PLoS Comput. Biol.* 16:e1007606. doi: 10.1371/journal.pcbi.1007606

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213–215. doi: 10.1126/science.275.5297.213

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., et al. (2015). Reconstruction and simulation of neocortical microcircuitry. *Cell* 163, 456–492. doi: 10.1016/j.cell.2015.09.029

Mayr, C. G., and Partzsch, J. (2010). Rate and pulse based plasticity governed by local synaptic state variables. *Front. Synapt. Neurosci.* 2:33. doi: 10.3389/fnsyn.2010.00033

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2018). 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

Morrison, A., Aertsen, A., and Diesmann, M. (2007a). Spike-timing dependent plasticity in balanced random networks. *Neural Comput.* 19, 1437–1467. doi: 10.1162/neco.2007.19.6.1437

Morrison, A., and Diesmann, M. (2008). "Maintaining causality in discrete time neuronal network simulations," in *Lectures in Supercomputational Neurosciences: Dynamics in Complex Brain Networks*, eds P. B. Graben, C. Zhou, M. Thiel, and J. Kurths (Berlin; Heidelberg: Springer), 267–278.

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike-timing. *Biol. Cybern.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Morrison, A., Mehring, C., Geisel, T., Aertsen, A., and Diesmann, M. (2005). Advancing the boundaries of high connectivity network simulation with distributed computing. *Neural Comput.* 17, 1776–1801. doi: 10.1162/0899766054026648

Morrison, A., Straube, S., Plesser, H. E., and Diesmann, M. (2007b). Exact subthreshold integration with continuous spike times in discrete-time neural network simulations. *Neural Comput.* 19, 47–79. doi: 10.1162/neco.2007.19.1.47

Muller, E., Bednar, J. A., Diesmann, M., Gewaltig, M.-O., Hines, M., and Davison, A. P. (2015). Python in neuroscience. *Front. Neuroinformatics* 9:11. doi: 10.3389/fninf.2015.00011

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., and Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* 7:272. doi: 10.3389/fnins.2013.00272

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

Ngezahayo, A., Schachner, M., and Artola, A. (2000). Synaptic activity modulates the induction of bidirectional synaptic changes in adult mouse hippocampus. *J. Neurosci.* 20, 2451–2458. doi: 10.1523/JNEUROSCI.20-07-02451.2000

Parker, D. (1985). *Learning Logic*. Technical Report TR-47.

Pecevski, D., Kappel, D., and Jonke, Z. (2014). Nevesim: event-driven neural simulation framework with a python interface. *Front. Neuroinformatics* 8:70. doi: 10.3389/fninf.2014.00070

Pfeil, T., Grübl, A., Jeltsch, S., Müller, E., Müller, P., Petrovici, M. A., et al. (2013). Six networks on a universal neuromorphic computing substrate. *Front. Neurosci.* 7:11. doi: 10.3389/fnins.2013.00011

Plesser, H., Diesmann, M., Gewaltig, M.-O., and Morrison, A. (2015). "NEST: the neural simulation tool," in *Encyclopedia of Computational Neuroscience*, eds D. Jaeger and R. Jung (New York, NY: Springer), 1849–1852.

Plotnikov, D., Blundell, I., Ippen, T., Eppler, J. M., Morrison, A., and Rumpe, B. (2016). "NESTML: a modeling language for spiking neurons," in *Modellierung 2016 Conference, Vol. 254 of* LNI (Bonn: Bonner Köllen Verlag), 93–108

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

Ros, E., Carrillo, R., Ortigosa, E. M., Barbour, B., and Agís, R. (2006). Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics. *Neural Comput.* 18, 2959–2993. doi: 10.1162/neco.2006.18.12.2959

Rudolph, M. and Destexhe, A. (2006). Event-based simulation strategy for conductance-based synaptic interactions and plasticity. *Neurocomputing* 69, 1130–1133. doi: 10.1016/j.neucom.2005.12.059

Rumelhart, E. D., Geoffrey, H. E., and Ronald, W. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536. doi: 10.1038/323533a0

Sacramento, J., Costa, R. P., Bengio, Y., and Senn, W. (2018). "Dendritic cortical microcircuits approximate the backpropagation algorithm," in *Advances in Neural Information Processing Systems* (Montreal, QC), 8721–8732.

Sadeh, S., Clopath, C., and Rotter, S. (2015). Emergence of functional specificity in balanced networks with synaptic plasticity. *PLoS Comput. Biol.* 11: e1004307. doi: 10.1371/journal.pcbi.1004307

Schmidt, M., Bakker, R., Shen, K., Bezgin, G., Diesmann, M., and van Albada, S. J. (2018). A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLoS Comput. Biol.* 14:e1006359. doi: 10.1371/journal.pcbi.1006359

Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013). Stdp and stdp variations with memristors for spiking neuromorphic learning systems. *Front. Neurosci.* 7:2. doi: 10.3389/fnins.2013.00002

Sheik, S., Paul, S., Augustine, C., and Cauwenberghs, G. (2016). "Membrane-dependent neuromorphic learning rule for unsupervised spike pattern detection," in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Shanghai: IEEE), 164–167.

Sjöström, P., Turrigiano, G., and Nelson, S. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* 32, 1149–1164. doi: 10.1016/S0896-6273(01)00542-6

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Song, S., Sjöström, P., Reigl, M., Nelson, S., and Chklovskii, D. (2005). Highly nonrandom features of synaptic connectivity in local cortical circuits. *PLoS Biol.* 3:e68. doi: 10.1371/journal.pbio.0030068

Stapmanns, J., Hahne, J., Helias, M., Bolten, M., Diesmann, M., and Dahmen, D. (2021). Event-based update of synapses in voltage-based learning rules. Zenodo. doi: 10.5281/zenodo.4565188

Stimberg, M., Goodman, D., Benichoux, V., and Brette, R. (2014). Equation-oriented specification of neural models for simulations. *Front. Neuroinformatics* 8:6. doi: 10.3389/fninf.2014.00006

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. *Front. Neurosci.* 12:891. doi: 10.3389/fnins.2018.00891

Urbanczik, R., and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron* 81, 521–528. doi: 10.1016/j.neuron.2013.11.030

Watts, L. (1994). "Event-driven simulation of networks of spiking neurons," in *Advances in Neural Information Processing Systems* (Denver, CO), 927.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Cambridge, MA: Harvard University.

Yger, P., and Harris, K. D. (2013). The convallis rule for unsupervised learning in cortical networks. *PLoS Comput. Biol.* 9:e1003272. doi: 10.1371/journal.pcbi.1003272

Zenke, F., and Gerstner, W. (2014). Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Front. Neuroinformatics* 8:76. doi: 10.3389/fninf.2014.00076

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.

5. APPENDIX

5.1. Analytical Integration in Urbanczik-Senn Rule

To derive (17) it is convenient to first investigate $\Delta W_{ij}(0,t)$ and $\Delta W_{ij}(0,T)$ and then compute $\Delta W_{ij}(t,T) = \Delta W_{ij}(0,T) - \Delta W_{ij}(0,t)$. Assuming that the simulation starts at $t = 0$, the weight change from the start to time t is given by

$$\begin{aligned}\Delta W_{ij}(0,t) &= \eta \int_0^t dt' \int_0^{t'} dt'' \kappa\left(t'-t''\right) V_i^*\left(t''\right) s_j^*\left(t''\right) \\ &= \eta \int_0^t dt'' \int_{t''}^t dt' \, \kappa\left(t'-t''\right) V_i^*\left(t''\right) s_j^*\left(t''\right) \\ &= \eta \int_0^t dt'' \left[\tilde{\kappa}\left(t-t''\right) - \tilde{\kappa}\left(0\right)\right] V_i^*\left(t''\right) s_j^*\left(t''\right) \\ &= \eta \left[-I_2\left(0,t\right) + I_1\left(0,t\right)\right]\end{aligned}$$

where we exchanged the order of integration from the first to the second line. In the third line we introduced $\tilde{\kappa}(t)$ defined by $\kappa(t) = \frac{\partial}{\partial t}\tilde{\kappa}(t)$ and in the fourth line we defined the two integrals

$$\begin{aligned}I_1\left(t_1,t_2\right) &= -\int_{t_1}^{t_2} dt' \, \tilde{\kappa}\left(0\right) V_i^*\left(t'\right) s_j^*\left(t'\right), \\ I_2\left(t_1,t_2\right) &= -\int_{t_1}^{t_2} dt' \, \tilde{\kappa}\left(t-t'\right) V_i^*\left(t'\right) s_j^*\left(t'\right).\end{aligned}$$

In case of the Urbanczik-Senn rule

$$\tilde{\kappa}\left(t\right) = -e^{-\frac{t}{\tau_\kappa}}$$

which implies the identities

$$\begin{aligned}I_1\left(t_1,t_2+\Delta t\right) &= I_1\left(t_1,t_2\right) + I_1\left(t_2,t_2+\Delta t\right), \\ I_2\left(t_1,t_2+\Delta t\right) &= e^{-\frac{t_2-t_1}{\tau_\kappa}} I_2\left(t_1,t_2\right) + I_2\left(t_2,t_2+\Delta t\right),\end{aligned}$$

which we use to write the weight change from t to T as

$$\begin{aligned}\Delta W_{ij}(t,T) &= \Delta W_{ij}(0,T) - \Delta W_{ij}(0,t) \\ &= \eta \left[-I_2\left(0,T\right) + I_1\left(0,T\right) + I_2\left(0,t\right) - I_1\left(0,t\right)\right] \\ &= \eta \left[I_1\left(t,T\right) - I_2\left(t,T\right) + I_2\left(0,t\right)\left(1 - e^{-\frac{T-t}{\tau_\kappa}}\right)\right].\end{aligned}$$

This is the the result (17).

5.2. Voltage Clamping of the Adaptive Exponential Integrate-and-Fire Model

For the Clopath rule the change of the synaptic weight strongly depends on the excursion of the membrane potential V_m around a spike of the postsynaptic neuron which causes $\bar{u}_\pm$ to cross the respective thresholds $\theta_\pm$ so that (12) and (13) yield non-vanishing results. Within the original neuron model Brette and Gerstner (2005) u is reset immediately after it reached the spiking threshold so that the shape of the action potential is not described accurately. In our NEST implementation of `aeif_psc_delta_clopath` we adapted the approach of the reference implementation on ModelDB (Hines et al., 2004) and introduced a clamping of u to a fixed value V_{clamp} for a period of t_{clamp} before it is reset. The reference implementation is restricted to a simulation resolution of exactly 1 ms and sets u to two different values for the two subsequent simulation steps after a spike. To be independent of the resolution of the simulation, the implementation in NEST uses a constant V_{clamp}. In the simulations we set t_{clamp} to 2 ms and V_{clamp} to 33 mV. These values are chosen to match the behavior of the reference implementation.

5.3. History Management

There are three points that need to be considered in the context of history management: First, which information needs to be stored. Second, how to search and read out the history. Third, how to identify and remove information that is no longer needed. The first and third point mainly affect memory usage, while the second point impacts the simulation time as searching within shorter histories is faster.

There are four different histories to which our considerations apply. The one to store the membrane potential V_i^*, the compressed history $\Delta W_i(t_{LS},T)$ used only for the compressed event-driven scheme, the history to store the spike times s_i of the postsynaptic neuron (also used for ordinary STDP), and finally one might need a history that stores the last spike time for every incoming synapse (see below for details).

5.3.0.1. Adding Information to the History

This paragraph concerns only the history that stores the time trace of V_i^*. In every time step of the simulation, neurons call the protected function `write_history()` of the archiving node and pass the current value of the (low-pass filtered) membrane potential. The archiving node then computes the derived quantities V_i^* or combinations of V_i^* and s_i^*, and saves them in the history, which is of type `vector`. It is more efficient to do the computations inside the archiving node and not in the synapse for two reasons: Firstly, the computation is done only once and then used for all incoming synapses. This way no direct exchange of information between different synapses is required. Secondly, the archiving node does not need to store the raw membrane potentials before readout, but can directly store the derived quantities V_i^*, which reduces the memory consumption, especially in cases where only a non-continuous history is needed.

5.3.0.2. Readout of Information From the History

Let's assume t_{LS} and t_S be the times of the last and the current spike of a synapse. At time t_S that synapse then needs to request a part from $t_1 = t_{LS} - d$ to $t_2 = t_S - d > t_1$ of the history that ranges from $t_{start} < t_1$ to $t_{end} > t_2$. This part is shifted with respect to the spike times by a delay d which models the time of signal propagation from the postsynaptic soma back to the synapse. The software framework NEST of our reference implementation uses only one variable to represent the delay from synapse to soma and the delay in the opposite direction. Consequently, when a spike arrives at the synapse of the postsynaptic neuron, the synapse sees a membrane potential from the past. In case every time step of the simulation adds a new entry to the history, one can easily compute the positions of the entries corresponding to $t_{1/2}$ by

just knowing t_{start} and t_{end}. As pointed out in section 3.2 this is the case for the Urbanczik-Senn plasticity rule. If the history is not continuous in time, like in case of the Clopath rule, this scheme is not applicable. Instead, we add a time stamp s as an additional variable to each entry and search for those with the smallest/greatest s within the interval (t_1, t_2) using e.g., a linear or a binary search. Searching for the positions that define the start and the end of the requested interval is slower than computing them directly. Nevertheless, a non-continuous history can lead to a large acceleration of simulations as we discussed in case of the Clopath rule (section 3.4.1). Here, only values of the membrane potential in the vicinity of a spike of the postsynaptic neuron are needed so that neglecting the majority of values in between leads to a non-continuous history but saves memory.

Technically, the archiving node contains a function called `get_history()` which expects two iterators `start` and `finish` and two times t_1 and t_2. When executed, the function sets the iterators to point to the correct entries of the history of the postsynaptic neuron corresponding to t_1 and t_2, respectively. Having received the correct position of the pointers, the synapse evaluates the integral (6). In the event-driven compression scheme, the integration (11) is not done inside the synapse but inside the `archiving_node`. The reason for this is that the compressed history $\Delta W_i(t_{LS}, t_S)$, which is updated in case of an incoming spike, is stored inside the `archiving_node.` This way no exchange of information is needed. The synapse only triggers the updating process by calling the function `compress_history()` of the `archiving_node`. Internally, the `archiving_node` can use `get_history()` to obtain the part of the history that has to be integrated. Even though the linear search a priori might seem less efficient than a binary search or direct computation of the position, it turns out that it has an advantage in that it iterates consecutively over the history entries which can be employed to identify data no longer needed. Therefore, especially for short histories a simple iteration that comes without any overhead is fastest (see section 3.4.1).

5.3.0.3. Removing Information From the History

To prevent the history from occupying an unnecessary amount of memory, it is crucial to have a mechanism to delete those entries that have been used by all incoming synapses. The simplest implementation to identify these entries is to add one additional variable to each entry called *access counter* initialized to zero when the entry is created. When a synapse requests a part from t_1 to t_2 of the history, the algorithm iterates over all entries $t_1 < t < t_2$ and increases the access counters by one. After the update of the synaptic weight all entries whose access counters are equal to the number of incoming synapses are deleted. This scheme can be combined easily with a linear search starting the iteration from the oldest entry of the history.

For long histories a linear search is inefficient and should be replaced by a binary search or direct computation of positions if applicable. To avoid iteration within long histories, we replace access counters by a vector that stores the last spike time t_{LS} for every incoming synapse. If a synapse delivers a spike, it updates its entry in that vector by replacing t_{LS} by the time stamp of the current spike. After each weight update, searching the vector for the smallest t_{LS} allows us to safely remove all membrane potentials with time stamps $t < \min(\{t_{LS,i}\})$. In practice, we can further improve this mechanism with two technical details. Firstly, n incoming spikes with the same time stamp can share the same entry t_{LS} which we then have to provide with a counter that goes down from n to zero in steps of one whenever one of the n synapses sends a new spike for a time $t > t_{LS}$. Secondly, we can avoid the search for the smallest t_{LS} by making sure that the entries t_{LS} are in chronological order. This can be easily achieved if a synapse does not update its entry in the vector but removes it and appends a new one at the end of the vector.

5.4. Implementation of Experiments Using Clopath Rule

5.4.1. Spike Pairing Experiment

The setup of the spike pairing experiment from Clopath et al. (2010) presented in **Figures 7A,B** consists of two neurons connected via a plastic synapse. The pre- and postsynaptic neuron are forced to spike with a time delay of Δt by stimulation with `spike_generators` at times $t_{pre}^{(i)} = t^{(i)}$ and $t_{post}^{(i)} = t^{(i)} + \Delta t$, respectively. A positive time shift $\Delta t > 0$ refers to the presynaptic neuron spike before the postsynaptic one (pre-post pairing, solid lines in **Figure 7**) and vice versa. Spike pairs $\left(t_{pre}^{(i)}, t_{post}^{(i)}\right)$ are induced with frequency $f_{pair} = \frac{1}{t^{(i+1)} - t^{(i)}}$ and the weight change of the synapse is measured after a set of five pairs. In our simulation using NEST the presynaptic neuron is modeled as a `parrot_neuron` and the postsynaptic neuron is either of type `aeif_psc_delta_clopath` or `hh_psc_alpha_clopath`. In NEST `parrot_neurons` are model neurons that emit a spike whenever they receive one. In this setup they are required because devices like `spike_generators` support only static synapses in NEST so that we cannot connect the postsynaptic neuron directly to the `spike_generator` via a plastic synapse. The initial weight of the `clopath_synapse` connecting the two neurons is given by w_{init}. In this experiment we use the Clopath rule with fixed amplitude A_{LTD}. A list with all the parameters can be found in **Supplementary Table 1**.

5.4.2. Emergence of Strong Bidirectional Couplings

In this experiment after Clopath et al. (2010), a small network of N_I inhibitory and N_E excitatory neurons subject to an external input develops strong bidirectional couplings between neurons of the excitatory population. The input is given by N_p Poisson spike trains with rates

$$f_p^{(j)} = A_p e^{-\frac{(j-\mu_p)^2}{2\sigma_p^2}} + c_p, \tag{18}$$

where $j = 1, \ldots, N_p$. The center μ_p of the Gaussian is drawn randomly from a set s_p of possible values and a new value is drawn after each time interval t_μ. The total number of intervals is N_μ. In our simulation with NEST we used `aeif_psc_delta` model neurons with the same parameters (cf. **Supplementary Table 3**) for both the inhibitory and the excitatory population. The

simulation is divided into N_μ intervals between which the rates of the N_p `poisson_generators` are set according to (18). The `poisson_generators` are connected in a one-to-one manner to N_p `parrot_neurons` which in turn are connected to the network. The details of the latter connectivity can be found in **Supplementary Table 2**. In NEST a `poisson_generator` that is connected to several target model neurons generates an independent Poisson spike train for each of these neurons. Thus, the intermediate step via `parrot_neurons` is required to provide neurons in the network with common Poisson inputs. Moreover, a direct connection from a device like a `poisson_generator` to a model neuron via a plastic synapse is not possible in NEST. In this experiment, the Clopath rule with homeostasis (time dependent prefactor for LTD, cf. section 5.5) is used. **Figure 7C** shows the weights of the synapses among the excitatory population after the simulation.

5.5. Implementation of Homeostasis $A_{\mathrm{LTD}}(\bar{\bar{u}})$

For the network simulations presented in Clopath et al. (2010), the authors use a sightly modified version of the Clopath rule defined in (12): The constant factor A_{LTD} is replaced by a voltage dependent term

$$A_{LTD}\left(\bar{\bar{u}}\right) = A_{LTD}\left(\frac{\bar{\bar{u}}}{u_{ref}}\right)^2$$

to take into account homeostatic processes. The quantity $\bar{\bar{u}}$ is a temporal average of the quantity $\bar{u}_-(t)$ over a time window of $T = 1\ s$ and u_{ref} is a reference value. An exact temporal average requires storing the time trace of $\bar{u}_-(t)$ for the entire interval T. This would cancel the advantage of keeping only a sparse history as discussed in 3.2.3.2 where storage of time traces is needed only in the vicinity of spikes. Therefore, deviating from the original work by Clopath et al. (2010), we implement an additional low-pass filtering $\bar{\bar{u}}(t) = (\kappa_{low} * \bar{u}_-)(t)$ with an exponential kernel $\kappa_{low}(t) = H(t)\exp\left(-t/\bar{\bar{\tau}}\right)$ instead. Like $\bar{u}_\pm$, $\bar{\bar{u}}$ is passed as an additional state variable to the solver.

5.6. Implementation of Experiment Using Urbanczik-Senn Rule

In the simulation experiment shown in **Figure 8** the dendrite of a conductance-based two-compartment model neuron receives a spike pattern of duration T as an input via plastic synapses. The pattern consists of N_p independent Poisson spike trains with a firing rate f_p. For learning, the pattern is repeated N_{rep} times. Dendritic synapses adapt their weights so that after learning the somatic membrane potential U and the dendritic prediction V_w^* follow a matching potential U_M. The latter is created by somatic input via two `spike_generators` that are connected via a static excitatory or inhibitory connection, respectively. Both spike generators send spikes in every simulation step. Inhibitory input spikes have a constant weight to generate a constant somatic inhibitory conductance g_I. Excitatory spikes have a modulated weight to generate a periodic excitatory conductance g_E. The input to the dendritic compartment is provided by N_p `spike_generators` each of which is connected to one `parrot_neuron` which in turn is connected to the dendrite via a plastic `urbanczik_synapse`. The intermediate parrot_neurons are required since in NEST the spike_generators can have only static synapses as outgoing connections. The spike times of the `spike_generators` are set to repeatedly generate the spike pattern created before the start of the actual simulation. The neuron's state variables are read out by a `multimeter` and the synaptic weights by a `weight_recorder`.

5.7. Experiment With Modified Version of the Urbanczik-Senn Rule

The weight change of the Urbanczik-Senn rule as presented in section 2.5 in line with the original publication is driven by the prediction error

$$V_i^* = \left(s_i - \phi(V_i)\right) h(V_i),$$

where s_i is the somatic spike train and V_i the dendritic prediction of the somatic membrane potential U_i. Instead of integrating over the difference between the spike train and the rate $\phi(V_i)$ (spike-rate), one can derive two variants

$$V_i^* = \left(s_i - s_i^{dend}\right) h(V_i) \quad (spike - spike) \quad and$$
$$V_i^* = \left(\phi(U_i) - \phi(V_i)\right) h(V_i) \quad (rate - rate).$$

In the first one (spike-spike) we replaced the dendritic rate prediction by a noisy realization s_i^{dend} using an inhomogeneous Poisson process with rate $\phi(V_i)$. In the second one (rate-rate) the somatic spike train is replaced by the rate of the underlying Poisson process which is computed by applying the rate function ϕ to the somatic potential U_i. The learning of a matching potential U_M as described in section 3.3 also works in these two cases. **Figure 13** shows the learning curve for all three variants of the Urbanczik-Senn rule. The loss is defined as the average mismatch between U_i and U_M averaged over one period T_p of the input pattern

$$\frac{1}{T_p}\int dt\left(U(t) - U_M(t)\right)^2.$$

The decrease of the loss as a function of the pattern repetitions has a similar shape for all three variants with a significantly higher variance in case of the spike-spike version.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 12 October 2021
doi: 10.3389/fnins.2021.699632

Emulation of Astrocyte Induced Neural Phase Synchrony in Spin-Orbit Torque Oscillator Neurons

Umang Garg[1,2], *Kezhou Yang*[1] *and Abhronil Sengupta*[1]*

[1] School of Electrical Engineering and Computer Science, Department of Materials Science and Engineering, The Pennsylvania State University, University Park, PA, United States, [2] Department of Electronics and Instrumentation Engineering, Birla Institute of Technology and Science, Pilani, India

Astrocytes play a central role in inducing concerted phase synchronized neural-wave patterns inside the brain. In this article, we demonstrate that injected radio-frequency signal in underlying heavy metal layer of spin-orbit torque oscillator neurons mimic the neuron phase synchronization effect realized by glial cells. Potential application of such phase coupling effects is illustrated in the context of a temporal "binding problem." We also present the design of a coupled neuron-synapse-astrocyte network enabled by compact neuromimetic devices by combining the concepts of local spike-timing dependent plasticity and astrocyte induced neural phase synchrony.

Keywords: neuromorphic computing, magnetic tunnel junction, astrocytes, Spintronics, spiking neural networks

OPEN ACCESS

Edited by:
Anup Das,
Drexel University, United States

Reviewed by:
Arash Ahmadi,
Carleton University, Canada
Kazuki Nakada,
Hiroshima City University, Japan
Debanjan Bhowmik,
Indian Institute of Technology Delhi, India

***Correspondence:**
Abhronil Sengupta
sengupta@psu.edu

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

Received: *23 April 2021*
Accepted: *25 August 2021*
Published: *12 October 2021*

Citation:
Garg U, Yang K and Sengupta A (2021) Emulation of Astrocyte Induced Neural Phase Synchrony in Spin-Orbit Torque Oscillator Neurons. Front. Neurosci. 15:699632. doi: 10.3389/fnins.2021.699632

1. INTRODUCTION

Neuromorphic engineering is emerging to be a disruptive computing paradigm in recent times driven by the unparalleled efficiency of the brain at solving cognitive tasks. Brain-inspired computing attempts to emulate various aspects of the brain's processing capability ranging from synaptic plasticity mechanisms, neural spiking behavior to *in-situ* memory storage in the underlying hardware substrate and architecture. The work presented in this article is guided by the observation that current neuromorphic computing architectures have mainly focused on emulation of bio-plausible computational models for neuron and synapse—but have not focused on other computational units of the biological brain that might contribute to cognition.

Over the past few years, there has been increasing evidence that glial cells, and in particular, astrocytes play an important role in multitude of brain functions (Allam et al., 2012). It is estimated that glia form ∼50% of the human brain cells (Möller et al., 2007) and participate by modulating the neuronal firing behavior, though unable to discharge electrical impulses of their own. Indeed, these glial-cells work in coordination with neural assemblies, to enable information processing in the human brain and performing incisive operations. Astrocytes hold the recipe to potentiate or suppress neurotransmitter activity within networks and are responsible for phenomenon like synchronous network firing (Fell and Axmacher, 2011; Wade et al., 2011) and self-repair mechanisms (Wade et al., 2012; Rastogi et al., 2020). It is therefore increasingly important to capture the dynamics of such ensembles, a step toward realizing more sophisticated neuromimetic machines and ultimately enabling cognitive electronics.

Recently, there has been extensive literature reporting astrocyte computational models and their impact on synaptic learning (De Pittà et al., 2012; Manninen et al., 2018). Continuing these fundamental investigations to decode neuro-glia interaction, there have been recent neuromorphic implementations of astrocyte functionality in analog and digital Complementary Metal Oxide

Semiconductor (CMOS) hardware (Möller et al., 2007; Irizarry-Valle and Parker, 2015; Naeem et al., 2015; Ranjbar and Amiri, 2017; Karimi et al., 2018; Faramarzi et al., 2019). For instance, analog CMOS circuits capturing the neural-glial transmitter behavior have been demonstrated (Joshi et al., 2011; Irizarry-Valle et al., 2013; Irizarry-Valle and Parker, 2015; Lee and Parker, 2016). There is also increasing interest in low-complexity FPGA implementation of the astrocyte computation models (Nazari et al., 2015; Ranjbar and Amiri, 2016, 2017; Karimi et al., 2018; Faramarzi et al., 2019). However, the primary focus has been on a brain-emulation perspective, i.e., implementing astrocyte computational models with high degree of detail in the underlying hardware.

On the other hand, recent advances in emerging post-CMOS technologies like phase change materials, resistive memories, ferromagentic, and ferroelectric materials (Jo et al., 2010; Kuzum et al., 2011; Ramakrishnan et al., 2011; Jackson et al., 2013; Sengupta and Roy, 2017; Saha et al., 2021), among others have resulted in the development of electronic device structures that can reproduce various biomimetic characteristics at low operating voltages through their intrinsic physics. However, while there has been extensive work on exploring post-CMOS technologies for mimicking bio-realistic computations due to the prospects of low-power and compact hardware design, they have been only studied from standalone neuron/synapse perspective. Emulation of the neuron-astrocyte crosstalk using bio-mimetic devices has largely been neglected, and no such literature exists hitherto, to the best of our knowledge. This work is therefore an effort to bridge this gap and, specifically, elucidates the emulation of transient synchronous activity resulting from neural-glial interactions by utilizing spin-orbit torque induced phase synchronization of spintronic oscillator neurons. It is worth mentioning here that we abstract the neuron functionality as a non-linear oscillator, in agreement with prior neuroscience and computational models (Jaeger and Haas, 2004). Emulation of astrocyte induced neural phase synchrony through the intrinsic physics of spintronic devices will be critical to enable the next generation of resource constrained cognitive intelligence platforms like robotic locomotion (Polykretis et al., 2020). This work also presents an important addition to the wide variety of next-generation computational paradigms like associative computing, vowel-recognition, physical reservoir computing among others (Fan et al., 2015; Torrejon et al., 2017; Romera et al., 2018, 2020; Riou et al., 2019; Tsunegi et al., 2019), being implemented using spin-torque oscillator devices.

2. NEUROSCIENCE BACKGROUND

The human brain houses multiple-independent local neuronal groups which perform dedicated computations in relevance to their assigned tasks. Besides this general uncorrelated activity of neurons, multiple neural spiking data recordings reveal that the independent signals from these neural assemblies frequently coalesce in time to generate a synchronous output (Fries, 2005; Fell and Axmacher, 2011). Multiple reports on the cause of such patterns now provide compelling evidence that astrocytes are the agents of this phenomenon (Fellin et al., 2004; Wade et al., 2011). Astrocytes modulate the concentration of neurotransmitters like glutamate inside the synaptic clefts in response to its internal Calcium (Ca^{2+}) oscillations (Newman, 2003; Garbo et al., 2007). A single astrocyte spans tens of thousands of synapses, where units called microdomains (concentrated Ca^{2+} stores within the astrocyte) monitor the activity for a group of neurons and perform subsequent chemical actions (Volterra and Meldolesi, 2005; Haydon and Carmignoto, 2006). The astrocyte-derived glutamate binds to extrasynaptic NMDAR (N-methyl-D-aspartate) receptor channels, and induce Slow-inward Currents (SIC) in the post-synaptic membrane. SICs are attributed to triggering a simultaneous response in different synapses with high timing precision, and its large amplitude and slow-decay rate provide an increased timescale for the correlated activity (Fellin et al., 2004; Wade et al., 2011). The astrocytic units influencing synapses, can act both independently or in coordination enabling long-distance indirect signaling among independent neuronal groups. Furthermore, an increased intensity of synaptic activity can trigger multiple astrocytes to share their information through their gap-junctions and elicit coherent behaviors among different uncorrelated neuronal networks. We in this paper do not discriminate among the two signaling processes. Thus, the two astrocytes shown in **Figure 1A** for different sub-networks can also imply microdomains within a single astrocyte. These units control the synchronization signal to networks A and B. **Figure 1A** captures the biological perspective of such a system which controls the neural synchronization among neurons present in these different sub-networks. Sub-networks A and B each consist of three different neurons, which in-turn generate oscillatory outputs. The temporal profiles, shown in **Figure 1B**, depict the neuron outputs before and after synchronization is initiated by Astrocyte 1 in the network A. Interested readers are referred to Wade et al. (2011) for details on the astrocyte computational models. It is worth mentioning here that unlike CMOS implementations that are able to implement computational models with a high degree of detail, emerging device based implementations usually focus on mimicking key aspects of the neurosynaptic functionality necessary from computing perspective since the exact behavior is governed by the intrinsic device physics. In this work, we primarily consider emulating the neural phase synchrony effect of astrocytes and evaluate it in the context of a temporal information binding application.

3. ASTROCYTIC SYNCHRONIZATION EMULATION

3.1. Device Basics

In this work, we utilize Magnetic Tunnel Junctions (MTJs) (Julliere, 1975) as the core hardware primitive to mimic neural oscillations. The MTJ consists of two ferromagnetic layers (pinned layer and free layer) with a spacer oxide layer in between. The direction of magnetization of the pinned layer (PL) is fixed, while that of the free layer (FL) can be manipulated by external stimuli (spin current/magnetic field). The MTJ

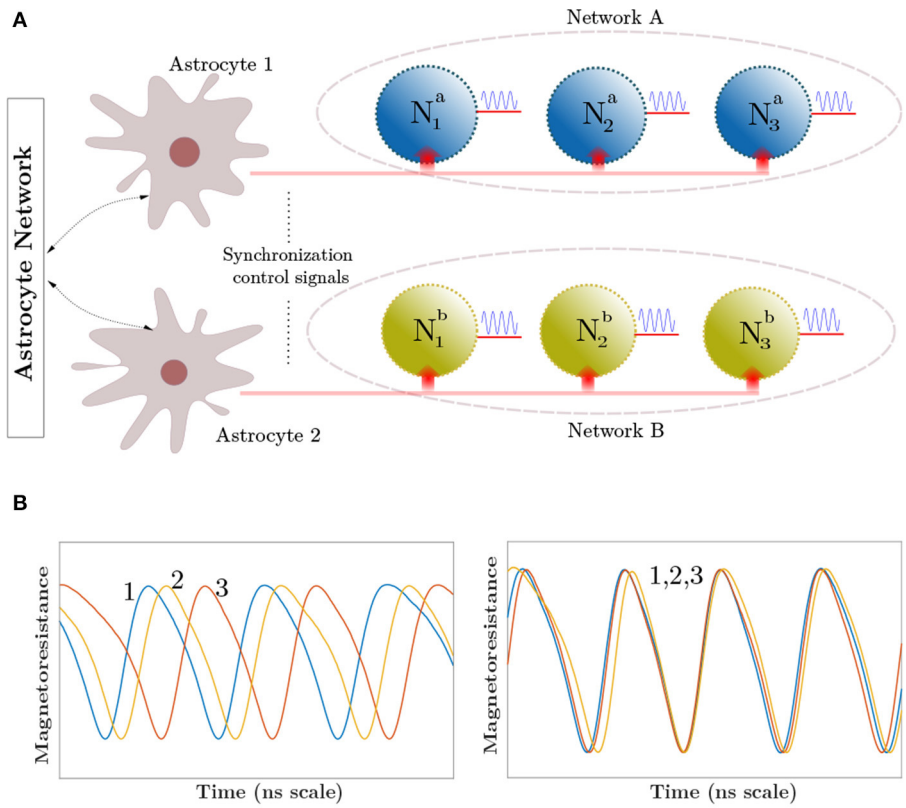


FIGURE 1 | (A) Top-level network depicting the synchronization control by astrocytic injection. Astrocytes share information among their glial network. **(B)** The curves show the synchronized and unsynchronized outputs of Neurons 1–3 in Network A depending on the astrocyte input.

stack exhibits a varying resistance depending on the relative magnetic orientations of the PL and the FL. The extreme resistive states are referred to as the parallel (P) and anti-parallel (AP) states depending on the relative FL magnetization. The magnetization dynamics of the FL can be modeled by Landau-Lifshitz-Gilbert-Slonczewski (LLGS) equation with stochastic thermal noise (Sengupta and Roy, 2017):

$$\frac{d\hat{m}}{dt} = -\gamma(\hat{m} \times H_{eff}) + \alpha(\hat{m} \times \frac{d\hat{m}}{dt}) + \frac{1}{qN_s}(\hat{m} \times I_s \times \hat{m}) \quad (1)$$

In Equation (1), $\hat{m}$ is the unit vector representing the magnetization direction of FL, H_{eff} is the effective magnetic field including thermal noise (Scholz et al., 2001), demagnetization field and external magnetic field, γ is the gyromagnetic ratio, α is Gilbert's damping ratio, I_s is the spin current, q is the electronic charge, and $N_s = \frac{M_s V}{\mu_B}$ is the number of spins in free layer of volume V (M_s is saturation magnetization and μ_B is Bohr magneton). If the magnitude of spin current and external magnetic field are chosen appropriately such that the damping due to the effective magnetic field is compensated, a steady procession of the FL magnetization can be obtained. It is worth mentioning here that the intrinsic magnetization dynamics in Equation (1) is used to model the oscillator dynamics. Other variants of oscillatory behavior can be achieved by modified spin device structures (Matsumoto et al., 2019).

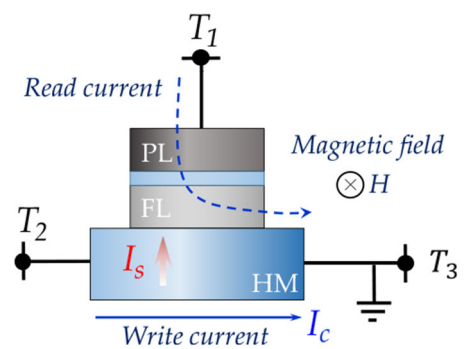


FIGURE 2 | Spin-orbit torque device undergoes oscillation due to applied external magnetic field, H, and charge current, I_c. Note that the directions of both the magnetic field and magnetic anisotropy are in-plane.

In order to achieve decoupled output oscillator readout and astrocyte injection induced phase coupling, we utilize a three terminal device structure, as shown in **Figure 2**, in which a nanomagnet with in-plane magnetic anisotropy lies on top of a heavy metal (HM) layer with high spin-orbit

TABLE 1 | MTJ device simulation parameters.

Parameters	Value
Ferromagnet area, A_{FM}	40 × 100 nm
HM thickness, t_{HM}	3 nm
Energy barrier, E_b	62.76 kT
Saturation magnetization, M_s	$\frac{10^7}{4\pi}$ A/m
Spin-hall angle, θ_{SH}	0.3
Spin-flip length, λ_{sf}	1.4 nm
Gilbert damping factor, α	0.03
External magnetic field, H	750 Oe
TMR ratio, *TMR*	200%
Temperature, T	300 K

coupling. Due to spin-Hall effect (Hirsch, 1999), a transverse spin current is injected into the MTJ FL by charge current, I_c, flowing through the HM between terminals T2 and T3. The relation between spin current I_s and charge current I_c is,

$$I_s = \theta_{SH}\frac{A_{FM}}{A_{HM}}\left(1 - \text{sech}\left(\frac{t_{HM}}{\lambda_{sf}}\right)\right)I_c \tag{2}$$

where, A_{FM} and A_{HM} are the FM and HM cross-sectional areas respectively, θ_{SH} is the spin-Hall angle (Hirsch, 1999), t_{HM} is the HM thickness and λ_{sf} is the spin-flip length. Note that an in-plane magnetic field, H, is also applied to achieve sustained oscillation. The MTJ state is read using the current sensed through terminal T1. The device simulation parameters are tabulated in **Table 1** and are based on typical experimental measurements reported in literature (Fan et al., 2015). However, the conclusions presented in this study are not specific to these parameters. Experimental demonstration of injection locked spin-torque oscillators have been achieved (Rippard et al., 2005, 2013; Georges et al., 2008; Demidov et al., 2014). It is worth mentioning here that we assume all the devices are magnetically isolated and sufficiently spaced such that dipolar coupling is negligible (Yogendra et al., 2017). We also consider that the generated charge current in the HM layer due to FL magnetic precession via the Inverse spin-Hall effect (ISHE) is not dominant enough to impact the phase coupling phenomena. While recent studies have shown that the ISHE modulated current alone, without any amplification, is not sufficient to impact phase locking (Elyasi et al., 2015), such effects can be also overcome by limiting the number of oscillators sharing a common HM substrate.

3.2. Phase Synchronization of MTJ Oscillator Neurons

The electrical analog of **Figure 1A** is shown in **Figure 3**, where the MTJs represent the oscillatory neurons present in a particular network. The neurons share a HM layer which acts as the common substrate for the driving astrocyte signal. The current flowing through the HM has two components—a DC current input which determines the free-running frequency of the oscillator and a radio-frequency signal which represents the astrocyte input. **Figure 4A** highlights the oscillation characteristics of the MTJ. The DC current controls the precession frequency in absence of other inputs. This DC input is analogous to the external stimulus determining the frequency of neuron oscillation in a particular network. In the absence of the RF signal, all the neurons oscillate at the same frequency (dependent on stimulus magnitude or DC current) but out-of-phase due to thermal noise. Upon the application of the external RF astrocyte signal, the device oscillation locks in phase and frequency to this input. Higher peak-to-peak amplitude of the astrocyte locking signal increases the locking range of the device. It is worth mentioning here that the locking frequency of neurons in a particular network is dependent on the stimulus and astrocytes only induce phase locking. Therefore, the alternating astrocyte signal flowing through the HM layer can be generated from a separate astrocyte device that is driven by the corresponding DC input of the network, thereby ensuring independent phase and frequency control. The astrocyte device is interfaced with a Reference MTJ and a voltage-to-current converter to drive the alternating current signal through the common HM layer. The Reference MTJ state is fixed to the AP state (by ensuring that the read supply voltage, $V_{DD} = 0.65V$ is not high enough to write the MTJ state) and forms a resistive divider with the oscillating Astrocyte MTJ resistance. Therefore, the gate voltage of the interfaced PMOS transistor, $V_G = \frac{R_A}{R_A+R_{REF}}V_{DD}$ where R_A is the Astrocyte MTJ resistance and R_{REF} is the Reference MTJ resistance, also varies accordingly, which in turn, modulates the current flowing through the common HM layer proportionally.

In order to evaluate the degree of phase synchronization in presence of thermal noise, we consider two MTJ devices lying on top of a common HM layer at room temperature. Cross-correlation metric is evaluated for the two MTJ output signals to measure the similarity among them as a function of displacement of one relative to the other. Considering two time-domain functions $x(t)$ and $y(t)$, whose power spectrum density (PSD) is given by $S_{xx}(\omega)$ and $S_{yy}(\omega)$, respectively, their cross-correlation is defined by:

$$R_{xy}(\tau) = (x \star y)(\tau) = \int_{-\infty}^{\infty} \overline{x(t-\tau)}y(t)\,dt \tag{3}$$

where, $\overline{x(t)}$ represents the complex conjugate of $x(t)$ and τ denotes the lag parameter. Further, cross-power spectral density (CPSD) is defined as the Fourier transformation of cross-spectrum in (3) and is given by:

$$S_{xy}(\omega) = \int_{-\infty}^{\infty} R_{xy}(t)e^{-j\omega t}\,dt \tag{4}$$

S_{xy} comprises of both magnitude and phase ($\angle$) information at different frequencies present in $[\omega]$ vector. When two signals are phase synchronized, the cross-spectrum phase vector becomes zero, indicating high correlation.

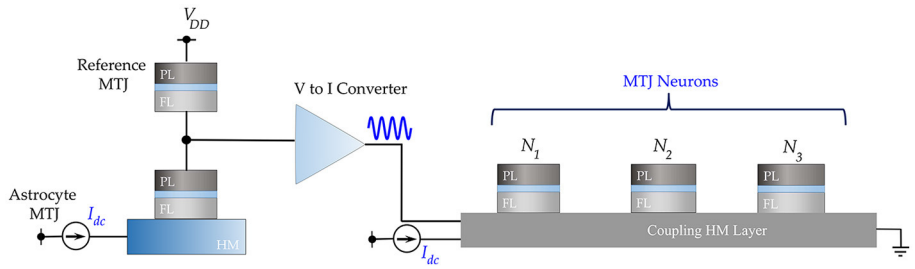


FIGURE 3 | Electrical emulation of astrocyte induced neural synchrony is shown where an astrocyte device drives an alternating current through a common HM substrate to phase-lock the MTJ oscillator neurons.

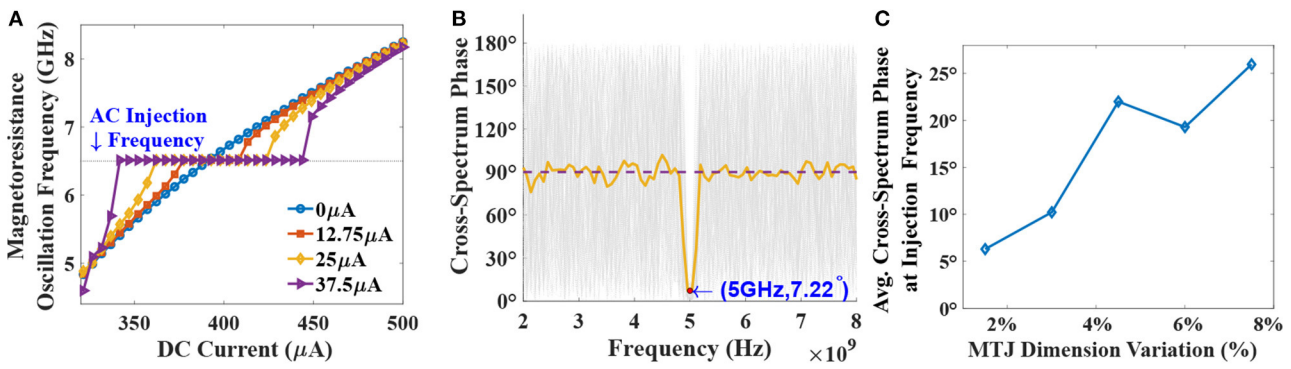


FIGURE 4 | (A) Oscillator frequency plotted against the DC current input to the device. Higher AC amplitudes lead to increased DC locking range at the injected RF signal of 6.5 GHz frequency. **(B)** Cross-spectrum phase for 100 independent stochastic LLGS simulations of two noisy MTJ neurons, under RF injection of 5*GHz*. Average CPSD phase indicates tight phase-coupling at the required frequency with un-correlated activity at other frequencies. **(C)** Average cross-spectrum phase at the injection frequency accounting for device dimension variations.

Such a property is highlighted in **Figure 4B** where 100 independent stochastic-LLGS simulations are performed for two neuronal devices placed on a common HM layer with a 5 GHz injected RF current. Cross-spectrum phase at the injection frequency, i.e., 5 GHz converges close to zero. Average cross-spectrum phase is also shown in the plot depicting tight phase-coupling between the neurons at the injection frequency. Notably, a sharp reduction of average phase offset to just 7.22° at 5 GHz is observed compared to 90° for other frequencies, thereby establishing the robustness of the synchronization scheme. Additionally, the impact of non-idealities like device dimension variations on the phase coupling phenomena is evaluated in **Figure 4C**. The results are reported for 50 independent Monte-Carlo simulations with variation in both the length and width of the MTJ. Each Monte-Carlo simulation consisted of 50 stochastic LLGS simulation for the average cross-spectrum phase calculation. The phase correlation between the device oscillations remains reasonably high even with 7.5% variation in both length and width dimensions of the MTJ. Related discussions on oscillator dynamics with respect to perturbative current and correspondence of the results with the Kuramoto model for oscillator synchronization is provided in the **Supplementary Material**.

4. BINDING PROBLEM

4.1. Problem Formulation

Next, we discuss a renowned problem which is envisioned to be solved by neural synchronous activity. Amongst the most intriguing themes of neuro-psychological studies is the "binding problem" (BP) (Feldman, 2013; Fields et al., 2014). It concerns with how different attributes of sensory information are encoded, processed, and perceived for decision-making by the human brain circuits. With a now widely accepted viewpoint of distributive computing and segregated processing for different features (especially visual) and later integration into a unified percept via re-entrant connections (Milner, 1974; Bartels and Zeki, 2006), we have progressed further toward understanding cognition. Primate brains have evolved to continuously assimilate the voluminous perceptive information available in their social setting and find a best fit for the primate's goals in the quickest manner. This training and growth, although very crucial in most situations—sometimes also leads to "misbinding" (Whitney, 2009). In particular, optical illusions, such as shown in **Figure 5A**, exploit the feature patterns ingrained in the human visual percept, causing misbinding. The figure is a bistable portrait of an elephant, or an overlap of two (seemingly) possible

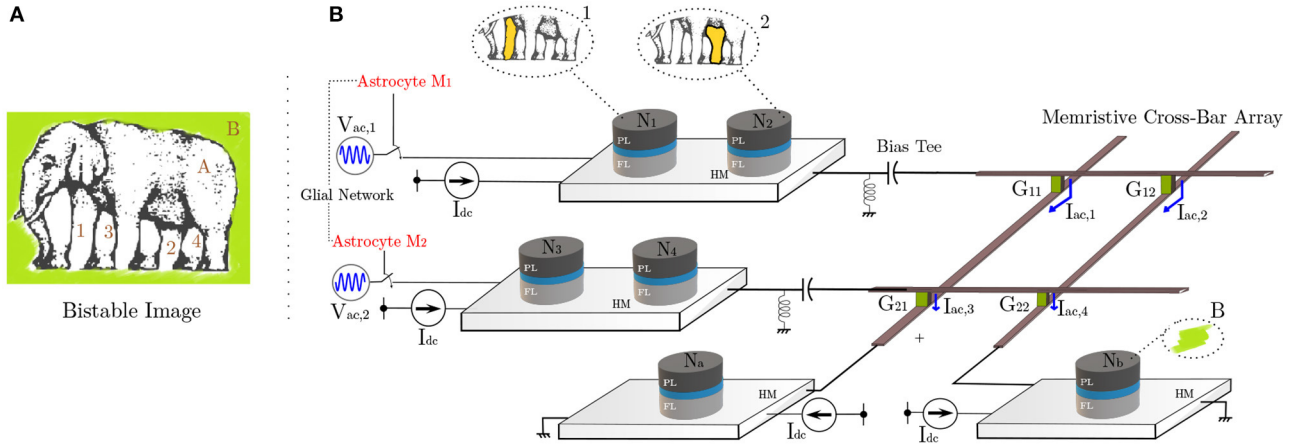


FIGURE 5 | (A) The optical illusion induces confusion in the viewer concerning association among different apparent limbs with the body and the background (*Courtesy of Roger Shepard's "L'egsistential paradox"*) (Shepard, 1990). **(B)** MTJ system architecture depicting hierarchical organization of neurons. The illustrated binding problem is mapped to this hardware with one possible interpretation shown. The connection between different neuron layers is implemented by the memristive cross-bar array with initially untuned synaptic weights. Unsupervised STDP learning rule causes the weights to evolve, making the network to finally elicit synchronous responses post-training.

interpretations, obtained by associating different body parts to other features of the image. For instance, the labels 1 and 2 can be viewed associated with the body (A), while 3 and 4 to the background (B) to paint one such possible interpretation. The other interpretation can be visualized if the roles A and B are reversed. For an in-depth discussion, interested readers are directed to Hasz and Miller (2013) and Ignatov et al. (2017). In this work, we do not address the clustering mechanism of labels 1–2 and 3–4. This labeling and identification can be potentially attributed to the agent's visual attention. In particular, attention captures the most relevant information present in a space-time lapse by masking (filtering) off the distractor areas, while performing feature labeling of the cropped scene (Kosiorek et al., 2017). Assuming that attention performs the role of spatio-temporal integration among such multiple attributes captured by a visual scene, synchronous activity in the neurons is considered as the underlying mechanism in brain to create a coherent episode of perception, and perhaps cognition. Indeed, it is now becoming more evident that cognitive processes like attention and behavioral efficiency elicit targeted synchronous activity in different brain regions tuned to responding toward different spatial and featural attributes of the attended sensory input (Ward, 2003; Womelsdorf and Fries, 2007).

4.2. Hardware Mapping

In order to correlate our spin-orbit torque oscillator phase synchronization due to astrocyte injection locking in the context of "temporal binding," we consider a network as shown in **Figure 5B**. Adhering to the currently prominent view of hierarchical organization in the neural assemblies, spin-torque neurons N_1, N_2, N_3, N_4 here are dedicated to processing simple attributes, while N_a and N_b after receiving inputs from previous layers perform complex feature processing corresponding to the assigned task. In reference to potential processing applications like cognitive feature binding, each spin-orbit torque neuron in the network represents the corresponding feature in the elephant's bistable image, previously shown in **Figure 5A**. All neuronal devices are mounted atop a HM with $I_{\rm dc} = 420\mu A$ DC drive ($f_{\rm free} = 7.05$ GHz). The network utilizes two different injection signals with the same frequency of 7.05 GHz with 180^o phase difference (corresponding to the two different interpretations/configurations of the bistable image). Here, we use two RF voltage sources, namely $V_{\rm ac1}$ and $V_{\rm ac2}$ with amplitude of 250 mV. The connection between the two neuron layers is achieved by means of a resistive synaptic cross-bar array. We combine the concepts of bio-inspired unsupervised Spike-Timing Dependent Plasticity (STDP) (Bi and Poo, 1998) and astrocyte induced neural phase synchrony to automatically enable the network to learn to elicit such behavioral patterns, on the fly. The developed system sets off from an unlearnt state where all neurons have an independent response and remain unsynchronized in phase. However, upon system activation (and consequently astrocyte RF injection), the architecture eventually learns to bind the different possible configurations for the visual scene through phase correlation to either $V_{\rm ac1}$ or $V_{\rm ac2}$. It is to be noted that neurons N_1, N_2, N_3, N_4 comprise of pre-neurons while $N_{\rm a}$ and $N_{\rm b}$ are post neurons, separated by the resistive cross-bar array. Ultimately, a tight phase and frequency locking is observed among a particular pair of pre-neurons (N_1, N_2, and N_3, N_4) and post-neurons (N_a and N_b). Due to random thermal fluctuations, the devices can converge to either of the two possible configurations for the bistable image, thereby illustrating the concept of optical illusion. The work can potentially pave the way for efficient hardware realization of coupled neuron-synapse-astrocyte networks enabled by compact neuromimetic devices.

4.3. Learning Phase Correlation

The premise for triggering the synchronous activity via astrocyte is accredited to the sensory attention as discussed before, and can be mapped in our proposed system to the amplitude of RF injection signal. Similar to better binding observed with increased attention, larger amplitudes lead to improved neural coupling. The strength of each input current to N_a and N_b is controlled by the synaptic conductances $G_{11} - G_{22}$ of the memristive cross-bar array as shown in **Figure 5B**. Implementation of such cross-bar arrays with *in-situ* STDP learning has been previously explored for spintronic devices (Sengupta et al., 2016; Sengupta and Roy, 2017) and other post-CMOS technologies (Jo et al., 2010; Kuzum et al., 2011; Saha et al., 2021). It is worth mentioning here that each cross-connection also features a prior filtering "bias tee" to eliminate any possible DC current interactions among different devices. The DC paths of the bias tee are terminated to ground, while the AC signals get passed on to the cross-bar for coupling. Elaborating, the input AC current to the j_{th} post-neuronal device (considering HM resistance to be considerably lower in comparison to the synaptic resistances at each cross-point) can be described by Equation (5) as:

$$I_{ac,N_j}(t) = \sum_i G_{ij}.V_i(t) \tag{5}$$

We now elucidate how our proposed architecture captures the essence of the optical illusion problem, shown in **Figure 5**, in reference frame of an observer. Specifically, the system should be able to adapt and converge to one of the possible interpretation discussed above. In particular, biologically inspired unsupervised STDP principles are used to train the programmable synaptic conductances ($G_{11} - G_{22}$) in the cross-bar architecture for this purpose. The STDP weight (conductance) update equations are given by: $\Delta w = \eta_+ w \exp(\frac{-\Delta t}{\tau_+})$ (for $\Delta t > 0$) and $\Delta w = \eta_+ w \exp(\frac{\Delta t}{\tau_+})$ (for $\Delta t < 0$), where η_+ and τ_+ are learning hyperparameters, Δw is the synaptic weight update and Δt is the timing difference between the spikes corresponding to the selected post- and pre-neuron. The positive learning window ($\Delta t > 0$) update occurs whenever a post-neuron fires while the negative learning window ($\Delta t < 0$) update occurs at a pre-neuron firing event. It is worth pointing out here that we use a symmetric STDP learning rule in this work, i.e., the synaptic weight is potentiated for both the positive and negative learning windows. This is in contrast to the more popular asymmetric STDP observed in glutamatergic synapses (Bi and Poo, 1998), typically used in neuromorphic algorithms (Diehl and Cook, 2015). While symmetric STDP has also been observed in GABAergic synapses (Woodin et al., 2003), further neuroscience insights are required to substantiate the exact underlying mechanisms and cause of this plasticity. Asymmetric STDP is useful in application domains requiring temporal ordering of spikes, i.e., a pre-synaptic neuron spike will trigger a post-neuron spike. However, for our scenario, a temporal correlation is crucial irrespective of the sequence, which is enabled by the symmetric STDP behavior. Implementation of symmetric STDP in memristive cross-bar arrays can be easily achieved by proper waveform engineering of the programming voltage applied across the synapses (Serrano-Gotarredona et al., 2013; Sengupta et al., 2016). The cross-bar resistances are considered to have an ON/OFF resistance ratio of 10. The different input spike trains are derived from each device's magnetoresistance (MR) where a spike is triggered when the MR crosses its mean-value of 2 $K\Omega$. Because N_1 (N_3) and N_2 (N_4) share a common HM, either of them can be used to extract the pre-neuron spikes during the weight update period. Besides STDP, a lateral inhibition effect (Diehl and Cook, 2015) is utilized. Whenever a spike occurs for any pre-neuron (post-neuron), the corresponding row (column) weights of the array are potentiated. However, the remaining rows (columns) are depressed proportionately. The lateral inhibition weight update equations are given by: $\Delta w = -\eta_- w \exp(\frac{-\Delta t}{\tau_-})$ (for $\Delta t > 0$) and $\Delta w = -\eta_- w \exp(\frac{\Delta t}{\tau_-})$ (for $\Delta t < 0$), where η_- and τ_- are learning hyperparameters, Δw is the synaptic weight update and Δt is the timing difference corresponding to the symmetric STDP weight update for the row or column which experiences weight potentiation. The lateral inhibition scheme is a simple extension of the synaptic programming voltage waveform engineering used in prior work (Indiveri et al., 2011; Serrano-Gotarredona et al., 2013; Sengupta et al., 2016). During the learning phase, this lateral inhibition effect causes the neuron under study to start responding selectively toward a specific configuration. This, in turn, enables the network to later converge to one of the interpretations for **Figure 5A**, as mentioned previously. The network simulation parameters are outlined in **Table 2**. The tabulated time-constants are measured with respect to the time-step for LLG simulation.

TABLE 2 | Learning simulation parameters.

Parameters	Value
Time-step for LLG simulation	0.1 ps
STDP learning rate, η_+	0.25
STDP time constant, τ_+	5
Inhibition learning rate, η_-	0.15
Inhibition time constant, τ_-	5
Maximum synapse resistance in cross-bar array	25 $k\Omega$

4.4. Simulation Results

The net currents for devices A and B, evolving through time, is portrayed for one of the simulations in **Figures 6A,B** respectively. Meanwhile, the corresponding synaptic resistances for the network are plotted in **Figure 6C** to elucidate the learning process discussed previously. The learning phase for the simulation is plotted as a function of timestep of the LLG simulation of the MTJ devices (0.1 ps). Observing the temporal profiles, an interesting deduction can be formulated, confirming that the different post-neurons get dominantly locked to different injection frequencies. The two sinusoids, being initially out of phase and adding up in comparable amounts for post-neurons, result in very low net currents. But, as the learning progresses, it becomes clear that one of the frequency gets dominant for a particular post-neuron, and thus the envelope tends to flatten in

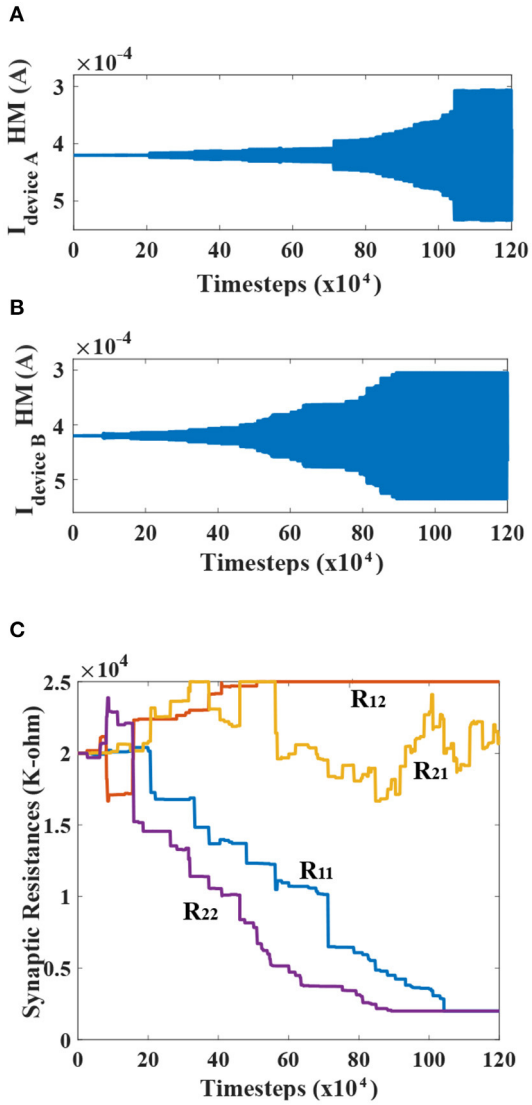


FIGURE 6 | (A,B) The temporal evolution profiles of the net currents (DC+AC) flowing through the heavy metal for devices A and B are shown. The increasing AC amplitudes about the mean DC value can be seen. The relatively flattened envelopes post-learning suggest that the post-neuron devices are dominantly locked to one of the frequencies. **(C)** Temporal evolution of the cross-bar resistances during the learning process is shown.

the end. It is worth mentioning here that the synaptic learning simulation in this work was performed from an algorithmic standpoint in a technology agnostic fashion. Depending on the underlying synapse technology, prior proposals for peripheral design for STDP learning needs to be considered (Serrano-Gotarredona et al., 2013; Sengupta et al., 2016). Since the focus of this article is on the MTJ neural synchrony aspect, we did not consider any specific synaptic device programming delay constraint (which is reflected in the instantaneous state changes of the synaptic connection strengths in **Figure 6C**). In reality, from a system design perspective, we need to have interleaved synaptic device state update phases that do not interfere with the neuron oscillation behavior (for instance, through decoupled write-read phases of three-terminal synaptic devices; Sengupta et al., 2016). The convergence was also not affected with reduced programming resolution of the synaptic connections (4-bits), thereby indicating resiliency to quantization (Hu et al., 2021).

Cohesing to one of the percept should surmise of a random event to provide equal chance for any of the two possible configurations to develop. Indeed, it is observed in our network that the synchronization occurs for random first and second layer neurons, post-training. Such a phenomenon can be accredited to the natural thermal fluctuations in our system, which tend to perturb the MTJ device's periodic nature. **Figures 7A,B**, respectively, depict the FFTs and cross-spectrum phase for various devices in the network for one such possible configuration upon learning termination. Specifically, cross-spectrum phases for device-pairs 1 & A (blue curve), 1 & 3 (yellow curve), and 1 & B (green curve) in **Figure 7B** are plotted to highlight that device 1, 2, and A get locked in phase at the injection frequency (7.05 GHz) while being completely out of phase with devices 3, 4, and B for the considered configuration.

Figure 8 plots the temporal profile of device magnetoresistance (MR) for N_1, N_2, and N_a devices in the top panel, along with MR of N_3, N_4, and N_b devices shown in the bottom panel. Initially all neuronal devices, albeit operating at the same free-running frequency ($f_{\text{free}} = 7.05$ GHz), elicit un-correlated phases, and hence temporal spike response due to devices' inherent thermal noise. After the astrocyte AC signal injection and STDP learning commences, it is observed that the devices N_1 (N_3) and N_2 (N_4) achieve a gradual coherent phase along with device N_a (N_b), getting locked to the respective injection signal, as can be clearly seen in the right panels. The subsequent cross-correlation phase at the 7.05 GHz injection frequency post-synchronization averages to 1.6232° for the three-possible temporal profile pairs among N_1, N_2, and N_a ($N_1 \star N_2$: 0.88°, $N_2 \star N_a$: 2.136°, and $N_1 \star N_a$: 1.856°). Likewise, N_3, N_4, and N_b after learning, achieve an average cross-phase of 1.848°. Bio-physically equivalent, this can be interpreted as a tight correlation among the attributes 1, 2, and A, corresponding to one of the interpretations of the bistable image. Finally, an increasing phase-mismatch is visible in neuronal outputs of all devices if the synchronization is revoked by the astrocyte, and the devices revert to their uncorrelated original free running frequency. This can be attributed to a diverted attention toward the sensory modal-input features leading to the impairment in correlated activity.

5. DISCUSSION

Even though this work proves to be a good preliminary framework for emulating such brain-like functions, more investigation is required for decoding the neural code in such processes along with integrating these insights in Artificial Intelligence (AI) systems. For instance, selectivity bias toward some features among the myriad available sensory information, and, reductionism (down-streaming) of such higher-level modal inputs to local neuronal groups in the hierarchical structure,

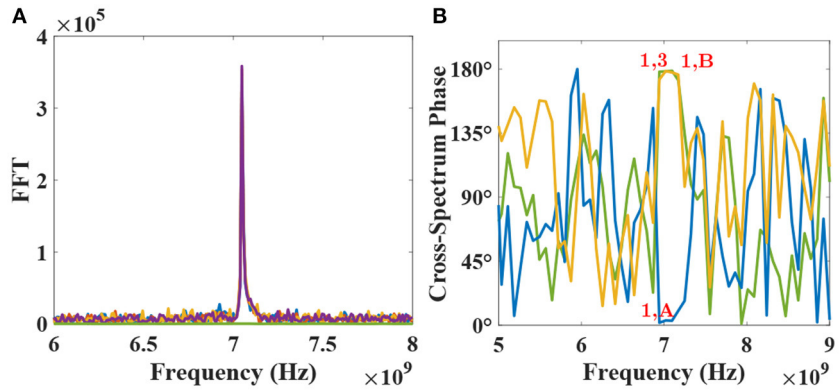


FIGURE 7 | (A) FFT plots for all devices for one of the two possible configurations are shown post-learning. **(B)** Cross-spectrum phase for devices-pairs 1–3 (178.53°), 1-A (3.35°), and 1-B (178.3°) are plotted to show the phase-locking nature of the network post-learning at the injection frequency of 7.05 GHz.

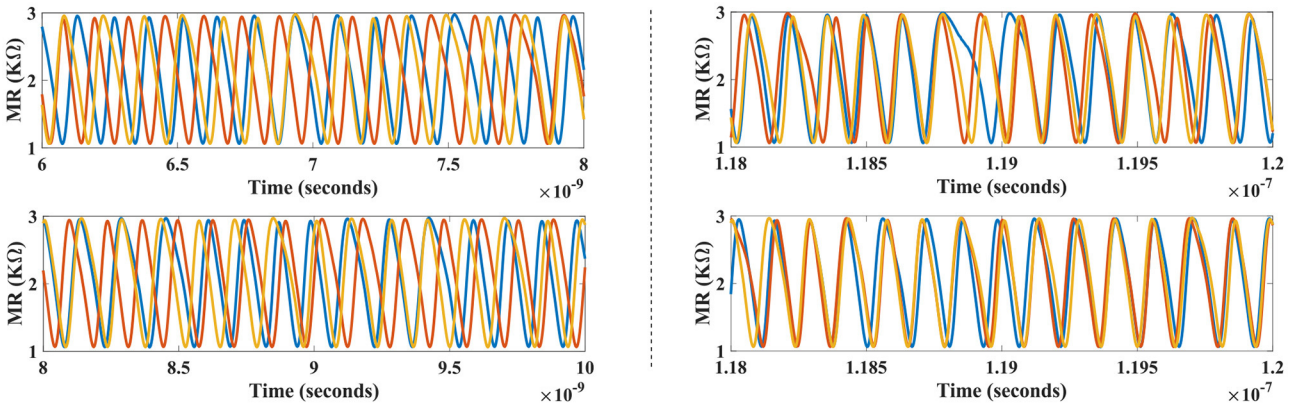


FIGURE 8 | Temporal profile for the devices in the network (shown in **Figure 5B**) before **(left)** and after synchronization **(right)** are depicted for one particular configuration. Astrocyte functionality activates the synchronous regime causing learning to occur and subsequently coherent neural patterns are achieved for this configuration (a stochastic event). Devices N_1, N_2, and N_a (top-right panel) lock to injection signal with $\phi = 0°$, while devices N_3, N_4, and N_b reveal concerted neural patterns in conjunction to $\phi = 180°$ injection signal (bottom-right panel).

is poorly understood. There have been some efforts to study such processes using a reverse approach, where robots like Darwin VIII, inspired by the re-entrant neuroanatomy and synaptic plasticity, are developed and trained on visual mode data (Seth et al., 2004). In agreement with our work, they show synchronous activity binds different representative features of the detected object. Incorporating such connections in our system can be explored to further bridge the gap between real cortical networks and the respective inspired models. Supported by both neuroscience research and AI hardware developments, coupled astrocyte-neuron network architectures can potentially pave the way for a new generation of artificial cognitive-intelligence.

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 contributed equally to the writing of the paper, developing the concepts, and performing the simulations.

FUNDING

The work was supported in part by the National Science Foundation grant nos. BCS #2031632, ECCS #2028213, and CCF #1955815.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2021.699632/full#supplementary-material

REFERENCES

Allam, S. L., Ghaderi, V. S., Bouteiller, J.-M. C., Legendre, A., Nicolas, A., Greget, R., et al. (2012). A computational model to investigate astrocytic glutamate uptake influence on synaptic transmission and neuronal spiking. *Front. Comput. Neurosci.* 6:70. doi: 10.3389/fncom.2012.00070

Bartels, A., and Zeki, S. (2006). The temporal order of binding visual attributes. *Vis. Res.* 46, 2280–2286. doi: 10.1016/j.visres.2005.11.017

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. doi: 10.1523/JNEUROSCI.18-24-10464.1998

De Pittá, M., Volman, V., Berry, H., Parpura, V., Volterra, A., and Ben-Jacob, E. (2012). Computational quest for understanding the role of astrocyte signaling in synaptic transmission and plasticity. *Front. Comput. Neurosci.* 6:98. doi: 10.3389/fncom.2012.00098

Demidov, V., Ulrichs, H., Gurevich, S., Demokritov, S., Tiberkevich, V., Slavin, A., et al. (2014). Synchronization of spin hall nano-oscillators to external microwave signals. *Nat. Commun.* 5, 1–6. doi: 10.1038/ncomms4179

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

Elyasi, M., Bhatia, C. S., and Yang, H. (2015). Synchronization of spin-transfer torque oscillators by spin pumping, inverse spin hall, and spin hall effects. *J. Appl. Phys.* 117:063907. doi: 10.1063/1.4907914

Fan, D., Maji, S., Yogendra, K., Sharad, M., and Roy, K. (2015). Injection-locked spin hall-induced coupled-oscillators for energy efficient associative computing. *IEEE Trans. Nanotechnol.* 14, 1083–1093. doi: 10.1109/TNANO.2015.2471092

Faramarzi, F., Azad, F., Amiri, M., and Linares-Barranco, B. (2019). A neuromorphic digital circuit for neuronal information encoding using astrocytic calcium oscillations. *Front. Neurosci.* 13:998. doi: 10.3389/fnins.2019.00998

Feldman, J. (2013). The neural binding problem(s). *Cogn. Neurodyn.* 7, 1–11. doi: 10.1007/s11571-012-9219-8

Fell, J., and Axmacher, N. (2011). The role of phase synchronization in memory processes. *Nat. Rev. Neurosci.* 12, 105–118. doi: 10.1038/nrn2979

Fellin, T., Pascual, O., Gobbo, S., Pozzan, T., Haydon, P. G., and Carmignoto, G. (2004). Neuronal synchrony mediated by astrocytic glutamate through activation of extrasynaptic nmda receptors. *Neuron* 43, 729–743. doi: 10.1016/j.neuron.2004.08.011

Fields, R. D., Araque, A., Johansen-Berg, H., Lim, S.-S., Lynch, G., Nave, K.-A., et al. (2014). Glial biology in learning and cognition. *Neuroscientist* 20, 426–431. doi: 10.1177/1073858413504465

Fries, P. (2005). A mechanism for cognitive dynamics: neuronal communication through neuronal coherence. *Trends Cogn. Sci.* 9, 474–480. doi: 10.1016/j.tics.2005.08.011

Garbo], A. D., Barbi, M., Chillemi, S., Alloisio, S., and Nobile, M. (2007). Calcium signalling in astrocytes and modulation of neural activity. *Biosystems* 89, 74–83. doi: 10.1016/j.biosystems.2006.05.013

Georges, B., Grollier, J., Darques, M., Cros, V., Deranlot, C., Marcilhac, B., et al. (2008). Coupling efficiency for phase locking of a spin transfer nano-oscillator to a microwave current. *Phys. Rev. Lett.* 101:017201. doi: 10.1103/PhysRevLett.101.017201

Hasz, B., and Miller, P. (2013). *Storing autoassociative memories through gamma-frequency binding between cell assemblies of neural oscillators* (Thesis). Brandeis University, Waltham, MA, United States

Haydon, P. G., and Carmignoto, G. (2006). Astrocyte control of synaptic transmission and neurovascular coupling. *Physiol. Rev.* 86, 1009–1031. doi: 10.1152/physrev.00049.2005

Hirsch, J. (1999). Spin Hall effect. *Phys. Rev. Lett.* 83:1834. doi: 10.1103/PhysRevLett.83.1834

Hu, S., Qiao, G., Chen, T., Yu, Q., Liu, Y., and Rong, L. (2021). Quantized STDP-based online-learning spiking neural network. *Neural Comput. Appl.* 1–16. doi: 10.1007/s00521-021-05832-y

Ignatov, M., Ziegler, M., Hansen, M., and Kohlstedt, H. (2017). Memristive stochastic plasticity enables mimicking of neural synchrony: memristive circuit emulates an optical illusion. *Sci. Adv.* 3:e1700849. doi: 10.1126/sciadv.1700849

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

Irizarry-Valle, Y., and Parker, A. C. (2015). An astrocyte neuromorphic circuit that influences neuronal phase synchrony. *IEEE Trans. Biomed. Circ. Syst.* 9, 175–187. doi: 10.1109/TBCAS.2015.2417580

Irizarry-Valle, Y., Parker, A. C., and Joshi, J. (2013). "A CMOS neuromorphic approach to emulate neuro-astrocyte interactions," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, TX, 1–7. doi: 10.1109/IJCNN.2013.6707076

Jackson, B. L., Rajendran, B., Corrado, G. S., Breitwisch, M., Burr, G. W., Cheek, R., et al. (2013). Nanoscale electronic synapses using phase change devices. *ACM J. Emerg. Technol. Comput. Syst.* 9:12. doi: 10.1145/2463585.2463588

Jaeger, H., and Haas, H. (2004). Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science* 304, 78–80. doi: 10.1126/science.1091277

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

Joshi, J., Parker, A. C., and Tseng, K. (2011). "An *in-silico* glial microdomain to invoke excitability in cortical neural networks," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, Rio de Janeiro 681–684. doi: 10.1109/ISCAS.2011.5937657

Julliere, M. (1975). Tunneling between ferromagnetic films. *Phys. Lett. A* 54, 225–226. doi: 10.1016/0375-9601(75)90174-7

Karimi, G., Ranjbar, M., Amirian, M., and Shahim-aeen, A. (2018). A neuromorphic real-time VLSI design of Ca2+ dynamic in an astrocyte. *Neurocomputing* 272, 197–203. doi: 10.1016/j.neucom.2017.06.071

Kosiorek, A., Bewley, A., and Posner, I. (2017). "Hierarchical attentive recurrent tracking," in *Advances in Neural Information Processing Systems 30*, eds I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Long Beach, CA: Curran Associates, Inc.), 3053–3061.

Kuzum, D., Jeyasingh, R. G., Lee, B., and Wong, H.-S. P. (2011). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* 12, 2179–2186. doi: 10.1021/nl201040y

Lee, R. K., and Parker, A. C. (2016). "A CMOS circuit implementation of retrograde signaling in astrocyte-neuron networks," in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Shanghai 588–591. doi: 10.1109/BioCAS.2016.7833863

Manninen, T., Havela, R., and Linne, M.-L. (2018). Computational models for calcium-mediated astrocyte functions. *Front. Comput. Neurosci.* 12:14. doi: 10.3389/fncom.2018.00014

Matsumoto, R., Lequeux, S., Imamura, H., and Grollier, J. (2019). Chaos and relaxation oscillations in spin-torque windmill spiking oscillators. *Phys. Rev. Appl.* 11:044093. doi: 10.1103/PhysRevApplied.11.044093

Milner, P. M. (1974). A model for visual shape recognition. *Psychol. Rev.* 81, 521–535. doi: 10.1037/h0037149

Möller, C., Lücke, J., Zhu, J., Faustmann, P. M., and von der Malsburg, C. (2007). Glial cells for information routing? *Cogn. Syst. Res.* 8, 28–35. doi: 10.1016/j.cogsys.2006.07.001

Naeem, M., McDaid, L. J., Harkin, J., Wade, J. J., and Marsland, J. (2015). On the role of astroglial syncytia in self-repairing spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2370–2380. doi: 10.1109/TNNLS.2014.2382334

Nazari, S., Faez, K., Amiri, M., and Karami, E. (2015). A digital implementation of neuron-astrocyte interaction for neuromorphic applications. *Neural Netw.* 66, 79–90. doi: 10.1016/j.neunet.2015.01.005

Newman, E. A. (2003). New roles for astrocytes: regulation of synaptic transmission. *Trends Neurosci.* 26, 536–542. doi: 10.1016/S0166-2236(03)00237-6

Polykretis, I., Tang, G., and Michmizos, K. P. (2020). "An astrocyte-modulated neuromorphic central pattern generator for hexapod robot locomotion on Intel's loihi," in *International Conference on Neuromorphic Systems 2020*, Oak Ridge, TN 1–9. doi: 10.1145/3407197.3407205

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

Ranjbar, M., and Amiri, M. (2016). Analog implementation of neuron-astrocyte interaction in tripartite synapse. *J. Comput. Electron*. 15, 311–323. doi: 10.1007/s10825-015-0727-8

Ranjbar, M., and Amiri, M. (2017). On the role of astrocyte analog circuit in neural frequency adaptation. *Neural Comput. Appl*. 28, 1109–1121. doi: 10.1007/s00521-015-2112-8

Rastogi, M., Lu, S., Islam, N., and Sengupta, A. (2020). On the self-repair role of astrocytes in STDP enabled unsupervised SNNs. *Front. Neurosci*. 14:603796. doi: 10.3389/fnins.2020.603796

Riou, M., Torrejon, J., Garitaine, B., Abreu Araujo, F., Bortolotti, P., Cros, V., et al. (2019). Temporal pattern recognition with delayed-feedback spin-torque nano-oscillators. *Phys. Rev. Appl*. 12:024049. doi: 10.1103/PhysRevApplied.12.024049

Rippard, W., Pufall, M., and Kos, A. (2013). Time required to injection-lock spin torque nanoscale oscillators. *Appl. Phys. Lett*. 103:182403. doi: 10.1063/1.4821179

Rippard, W. H., Pufall, M. R., Kaka, S., Silva, T. J., Russek, S. E., and Katine, J. A. (2005). Injection locking and phase control of spin transfer nano-oscillators. *Phys. Rev. Lett*. 95:067203. doi: 10.1103/PhysRevLett.95.067203

Romera, M., Talatchian, P., Tsunegi, S., Abreu Araujo, F., Cros, V., Bortolotti, P., et al. (2018). Vowel recognition with four coupled spin-torque nano-oscillators. *Nature* 563, 230–234. doi: 10.1038/s41586-018-0632-y

Romera, M., Talatchian, P., Tsunegi, S., Yakushiji, K., Fukushima, A., Kubota, H., et al. (2020). Binding events through the mutual synchronization of spintronic nano-neurons. *arXiv [Preprint] arXiv:2001.08044*.

Saha, A., Islam, A., Zhao, Z., Deng, S., Ni, K., and Sengupta, A. (2021). Intrinsic synaptic plasticity of ferroelectric field effect transistors for online learning. *arXiv preprint arXiv:2107.13088*.

Scholz, W., Schrefl, T., and Fidler, J. (2001). Micromagnetic simulation of thermally activated switching in fine particles. *J. Magn. Magn. Mater*. 233, 296–304. doi: 10.1016/S0304-8853(01)00032-4

Sengupta, A., Banerjee, A., and Roy, K. (2016). Hybrid spintronic-cmos spiking neural network with on-chip learning: devices, circuits, and systems. *Phys. Rev. Appl*. 6:064003. doi: 10.1103/PhysRevApplied.6.064003

Sengupta, A., and Roy, K. (2017). Encoding neural and synaptic functionalities in electron spin: a pathway to efficient neuromorphic computing. *Appl. Phys. Rev*. 4:041105. doi: 10.1063/1.5012763

Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013). STDP and STDP variations with memristors for spiking neuromorphic learning systems. *Front. Neurosci*. 7:2. doi: 10.3389/fnins.2013.00002

Seth, A. K., McKinstry, J. L., Edelman, G. M., and Krichmar, J. L. (2004). Visual Binding through reentrant connectivity and dynamic synchronization in a brain-based device. *Cereb. Cortex* 14, 1185–1199. doi: 10.1093/cercor/bhh079

Shepard, R. N. (1990). *Mind Sights: Original Visual Illusions, Ambiguities, and Other Anomalies, With a Commentary on the Play of Mind in Perception and Art*. New York, NY: W H Freeman; Times Books; Henry Holt & Co.

Torrejon, J., Riou, M., Araujo, F. A., Tsunegi, S., Khalsa, G., Querlioz, D., et al. (2017). Neuromorphic computing with nanoscale spintronic oscillators. *Nature* 547:428. doi: 10.1038/nature23011

Tsunegi, S., Taniguchi, T., Nakajima, K., Miwa, S., Yakushiji, K., Fukushima, A., et al. (2019). Physical reservoir computing based on spin torque oscillator with forced synchronization. *Appl. Phys. Lett*. 114:164101. doi: 10.1063/1.5081797

Volterra, A., and Meldolesi, J. (2005). Astrocytes, from brain glue to communication elements: the revolution continues. *Nat. Rev. Neurosci*. 6, 626–640. doi: 10.1038/nrn1722

Wade, J., McDaid, L., Harkin, J., Crunelli, V., and Kelso, S. (2012). Self-repair in a bidirectionally coupled astrocyte-neuron (an) system based on retrograde signaling. *Front. Comput. Neurosci*. 6:76. doi: 10.3389/fncom.2012.00076

Wade, J. J., McDaid, L. J., Harkin, J., Crunelli, V., and Kelso, J. A. S. (2011). Bidirectional coupling between astrocytes and neurons mediates learning and dynamic coordination in the brain: a multiple modeling approach. *PLoS ONE* 6:e29445. doi: 10.1371/journal.pone.0029445

Ward, L. M. (2003). Synchronous neural oscillations and cognitive processes. *Trends Cogn. Sci*. 7, 553–559. doi: 10.1016/j.tics.2003.10.012

Whitney, D. (2009). Neuroscience: toward unbinding the binding problem. *Curr. Biol*. 19, R251-R253. doi: 10.1016/j.cub.2009.01.047

Womelsdorf, T., and Fries, P. (2007). The role of neuronal synchronization in selective attention. *Curr. Opin. Neurobiol*. 17, 154–160. doi: 10.1016/j.conb.2007.02.002

Woodin, M. A., Ganguly, K., and Poo, M.-m. (2003). Coincident pre-and postsynaptic activity modifies gabaergic synapses by postsynaptic changes in CL- transporter activity. *Neuron* 39, 807–820. doi: 10.1016/S0896-6273(03)00507-5

Yogendra, K., Liyanagedera, C., Fan, D., Shim, Y., and Roy, K. (2017). Coupled spin-torque nano-oscillator-based computation: a simulation study. *ACM J. Emerg. Technol. Comput. Syst*. 13, 1–24. doi: 10.1145/3064835

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

HYPOTHESIS AND THEORY
published: 21 December 2021
doi: 10.3389/fnins.2021.750748

BrainFreeze: Expanding the Capabilities of Neuromorphic Systems Using Mixed-Signal Superconducting Electronics

Paul Tschirhart [1]* *and Ken Segall* [1,2]

[1] *Advanced Technology Laboratory, Northrop Grumman, Linthicum, MD, United States,* [2] *Department of Physics and Astronomy, Colgate University, Hamilton, NY, United States*

Superconducting electronics (SCE) is uniquely suited to implement neuromorphic systems. As a result, SCE has the potential to enable a new generation of neuromorphic architectures that can simultaneously provide scalability, programmability, biological fidelity, on-line learning support, efficiency and speed. Supporting all of these capabilities simultaneously has thus far proven to be difficult using existing semiconductor technologies. However, as the fields of computational neuroscience and artificial intelligence (AI) continue to advance, the need for architectures that can provide combinations of these capabilities will grow. In this paper, we will explain how superconducting electronics could be used to address this need by combining analog and digital SCE circuits to build large scale neuromorphic systems. In particular, we will show through detailed analysis that the available SCE technology is suitable for near term neuromorphic demonstrations. Furthermore, this analysis will establish that neuromorphic architectures built using SCE will have the potential to be significantly faster and more efficient than current approaches, all while supporting capabilities such as biologically suggestive neuron models and on-line learning. In the future, SCE-based neuromorphic systems could serve as experimental platforms supporting investigations that are not feasible with current approaches. Ultimately, these systems and the experiments that they support would enable the advancement of neuroscience and the development of more sophisticated AI.

Keywords: neuromorphic, architecture, superconducting, mixed-signal, spiking

OPEN ACCESS

Edited by:
Yoeri van de Burgt,
Eindhoven University of Technology, Netherlands

Reviewed by:
Yang Cindy Yi,
Virginia Tech, United States
Maryam Parsa,
George Mason University, United States

***Correspondence:**
Paul Tschirhart
paul.tschirhart@ngc.com

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

Received: *12 August 2021*
Accepted: *08 November 2021*
Published: *21 December 2021*

Citation:
Tschirhart P and Segall K (2021) BrainFreeze: Expanding the Capabilities of Neuromorphic Systems Using Mixed-Signal Superconducting Electronics. Front. Neurosci. 15:750748. doi: 10.3389/fnins.2021.750748

1. INTRODUCTION

Superconducting electronics (SCE) has many characteristics that make it a natural fit for implementing neuromorphic systems. This work will explore how such a system might be constructed using a combination of digital and analog SCE. The unique collection of capabilities enabled by SCE neuromorphic systems has the potential to provide solutions to some of the most difficult problems facing AI research in the future. In particular, the efficiency and speed of SCE architectures could help to address the compute challenges facing AI development. Similarly, the biological fidelity that is possible in SCE-based neural circuits may also prove to be a valuable source of future capabilities as researchers seek to incorporate novel functionality into their neural networks.

The amount of computation required to train more sophisticated AI applications is one of the most significant problems facing the development of these applications. As machine learning applications have advanced in terms of capabilities, their training requirements have also greatly increased. In **Figure 1** we can see that over the last decade, the amount of compute required to train new machine learning applications has grown much faster than the performance improvement in computing hardware. If this trend continues, then training times for future machine learning applications will become prohibitively long and eventually untenable.

As more advanced machine learning applications are developed, understanding and incorporating more complex system dynamics will likely be required to provide novel functionality. Inspiration for the form and function of these dynamics could potentially come from biological brains as was the case with the very concept of neural networks (Sompolinsky, 2014). However, efforts to understand the purpose of biological neural dynamics are limited by the significant computational requirements of accurate models of biological neurons. Current approaches generally struggle to simultaneously support the complexity, scale, and length of the experiments that would ideally explore these dynamics.

These trends suggest that a paradigm shift is needed in the area of neuromorphic computing in order to address both the need for novel functionality and the need for improved support for training. Existing neuromorphic approaches provide a set of capabilities that include scalability, programmability, on-line learning support, complex soma models, efficiency, and accelerated simulation timescales. However, to the best of our knowledge, no architecture has been able to provide all of these capabilities simultaneously. Instead, each design has been optimized for one or more of the capabilities at the cost of others (Furber, 2016). Enabling all of these capabilities simultaneously would be a significant step in the development of neuromorphic systems that will meet the future needs of machine learning applications and computational neuroscience experiments. Superconducting electronics has the potential to support all of these capabilities simultaneously but serious challenges need to be overcome with either architecture or device solutions in order to realize that potential.

A successful superconducting architecture needs to be scalable and programmable. To satisfy these requirements, we propose a mixed-signal approach that combines superconducting digital logic with superconducting analog neuron circuits. To investigate the suitability of possible SCE mixed-signal architectures, this work presents a series of trade-off studies using numerical analysis based on measurements and designs from previously demonstrated circuits. It is important to note that this work represents an early feasibility study and that additional research is required to fully develop the architecture and ideas discussed here. This work will hopefully motivate future work in this area

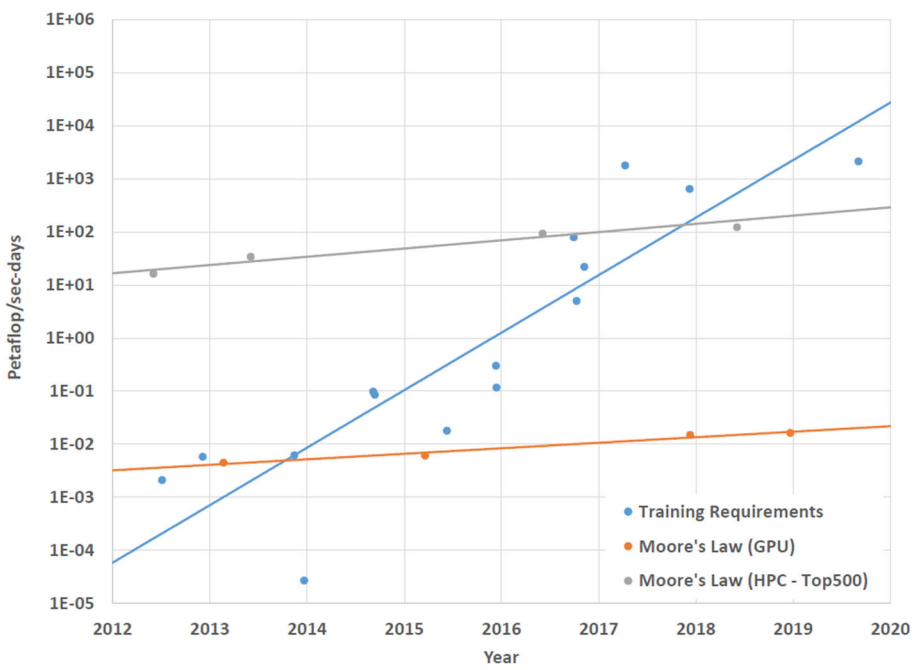


FIGURE 1 | This graph shows that compute requirements for training novel AI applications is growing at a much faster rate than improvements in compute performance. This figure expands on the analysis presented in Amodei et al. (2018). The AI application data is from Hinton et al. (2012), Mnih et al. (2013), Simonyan and Zisserman (2014), Sutskever et al. (2014), Zeiler and Fergus (2014), Szegedy et al. (2015), Amodei et al. (2016), He et al. (2016), Wu et al. (2016), Zoph and Le (2016), Chollet (2017), Krizhevsky et al. (2017), Silver et al. (2017b), Silver et al. (2017a), and Vinyals et al. (2019). The GPU data is the maximum theoretical single precision floating point operations per second for the NVIDIA Titan, TitanX, TitanV, and TitanRTX GPUs. The HPC data is the maximum theoretical double precision floating point operations per second for the Sequoia, Tianhe-2, TaihuLight, and Summit supercomputers.

that will result in the development of efficient, fast, and scalable neuromorphic systems that are also programmable, biologically suggestive, and capable of supporting on-line learning. Such systems could provide a critical platform that is needed for future AI development as well as computational neuroscience research.

The main contributions of this paper are:

- A description of a novel mixed-signal superconducting neuromorphic architecture
- A detailed analysis of the design trade-offs and feasibility of mixed-signal superconducting neuromorphic architectures
- A comparison of the proposed system with other state-of-the-art neuromorphic architectures
- A discussion of the potential of superconducting neuromorphic systems in terms of on-line learning support and large system scaling.

2. BACKGROUND

2.1. Neuromorphic State-of-the-Art

Neuromorphic computers are designed to replicate the structures and behaviors of the biological brain using analog or digital hardware or a mixture of both. By mimicking biology, neuromorphic computers are able to incorporate characteristics that support novel applications and research. There have been many different approaches to building these sorts of systems each with its own priorities and innovations. Five large-scale neuromorphic systems that we considered to be representative of the current state-of-the-art are compared in **Table 1**. This comparison is intended to be a brief sample of the current state of the field rather than a comprehensive review. As a result, there are other compelling approaches that are not included in this comparison.

It is worth noting that it is possible make use of novel technologies, such as phase change memory (PCM) or Memristors to efficiently implement the functionality of different portions of the neuron (Ebong and Mazumder, 2012; Soudry et al., 2015; Sebastian et al., 2018). However, a review of the literature failed to locate an example large scale implementation of these approaches that provided the relevant details and metrics that would be needed evaluate it relative to the other approaches considered here. As a result, these sorts of approaches are not considered in the comparison in **Table 1** which is focused on large scale implementations of spiking neuromorphic systems.

The primary characteristic that the approaches in **Table 1** have in common is that they are designed to enable scaling to millions of neurons and billions of synapses. To achieve that scale several critical design decisions must be made regarding the complexity of the neuron model, the number and resolution of synapses to allow per neuron in the design, and the interconnect scheme used to communicate between the neurons. These decisions, in turn, affect how many neurons the system can reasonably accommodate, the power that is consumed, and the speed with which the system can update the neuron models. In some cases these decisions also limit or preclude functionality such as on-line learning (Benjamin et al., 2014).

Perhaps the most fundamental design decision of neuromorphic architectures is how much detail to support in the neuron model. It is possible to implement biologically relevant models using semiconductor electronics but this approach typically requires prohibitively complex circuit designs that have reduced yield probabilities and scaling limitations (Arthur and Boahen, 2011). An example of this trade off can be seen by comparing IBM's TrueNorth which used a completely digital design to implement a million LIF neurons on a single chip (Merolla et al., 2014) and BrainScales which used a mixed signal approach to implement a more detailed neuron model in a wafer scale system (Schemmel et al., 2010; Meier, 2015). In the case of TrueNorth, the neuron model is power and area efficient but does not capture many biological details in a single instance (Cassidy et al., 2013b). BrainScales, on the other hand, uses a more complex model but the power used by the system is significant with 1.3 W required for just 512 neurons (Furber, 2016). This is roughly 1000–10,000x more power per neuron than is needed by large neuromorphic systems that use simplified neuron models (Furber, 2016).

Another interesting design decision concerns the interconnect scheme used to enable communication between the neurons. Aspects of these schemes, such as whether the messaging is unicast or multicast and what network topology is employed, ultimately determine the bandwidth and latency characteristics of the network. These decisions also affect the complexity of the routers required to implement the network. SpiNNaker is an approach that devotes a lot of effort to solving the networking problems posed by the need for thousands of synapses per neuron (Rast et al., 2008; Furber et al., 2013, 2014). Instead of implementing the neuron model directly in hardware like the other approaches that are featured in **Table 1**, SpiNNaker uses a software neuron model implementation running on ARM cores that enables flexibility in terms of model choice. As a result, SpiNNaker does not directly improve the time it takes to update individual neurons relative to a software only approach. However, the innovations in the SpiNNaker network enable it to achieve impressive scales and performance despite lacking a hardware-based neuron model. Additional details regarding the design of communication networks can be found in Young et al. (2019).

Modern neuromorphic systems can achieve impressive scales and performance while using relatively little power. However, all of the current approaches have drawbacks that can ultimately be traced to technology limitations. It is possible that pursuing a neuromorphic system in a different technology, such as SCE, could open up new possibilities in terms of system capabilities.

2.2. Superconducting Digital Electronics Development

Recently, the field of superconducting digital electronics has been reinvigorated by the IARPA C3 and SuperTools programs. As a result, the capabilities of superconducting digital electronics have been significantly improved and are ready for use in novel architectures. In particular, the C3 program has significantly helped to drive progress in this area

TABLE 1 | A comparison of representative large scale spiking neuromorphic architectures.

	BrainScaleS	Neurogrid	TrueNorth	SpiNNaker	Loihi
Feature size	180 nm	180 nm	28 nm	130 nm	14 nm
Neurons per core	8–512	65 k	256	1 k	1 k
Synapses per core	130 k	100 M	65 k	1 M	16 k
Technology	Analog	Analog	Digital	Digital	Digital
Soma model	AEIF	AQIF	LIF	Variable	Variable
Soma equiv FLOPS	15	12	∼5	Variable (∼13)	Variable (∼5)
Synapse resolution	4 bits	13 bits shared	1 bit	Variable	1–64 bits
Run-time plasticity	STDP	No	No	Variable	Variable
Interconnect	Hierarchical	Tree Multicast	2D Mesh Unicast	2D Toroidal Mesh Multicast	2D Mesh Unicast
Watts/Neuron	2.54 mW	2.31 μW	0.72 nW	62.5 μW	< 5 μW
Joules/Spike	198 pJ	119 pJ	26 pJ	11 nJ	23.6 pJ

The overall comparison expands on analysis from Furber (2016).

Sources for architecture details are Rast et al. (2008), Schemmel et al. (2010), Benjamin et al. (2014), Furber et al. (2013, 2014), Merolla et al. (2014), Meier (2015), Davies et al. (2018), Lin et al. (2018), and Yang and Kim (2020).

Soma Equivalent FLOPS refers to the number of floating point operations required to advance the soma model by a 1 ms time step when the model is implemented in software. This provides a very rough comparison of the complexity of the models that each architecture implements. TrueNorth, SpiNNaker, and Loihi all use soma models that incorporate varying degrees of programmability that allow for a range of model complexities. For the purposes of this comparison we assume an LIF model as the baseline configuration for TrueNorth and Loihi because that is what is reported in the literature (Cassidy et al., 2013b; Furber, 2016; Davies et al., 2018). Both approaches use a modified LIF model that would likely require more computation to simulate than a standard LIF model however it is unclear how much. SpiNNaker is assumed to use an Izhikevich model. FLOPS values are from Izhikevich (2004) and Makhlooghpour et al. (2016).

(Manheimer, 2015). C3 sought to address the unsustainable power demands of future CMOS-based supercomputers by developing energy efficient superconducting processors. To accomplish this goal, the program was divided into two thrusts with one focusing on developing digital logic and the other focusing on memory.

The digital logic portion of the program focused on the development of a processor with one of two competing families of superconducting digital logic: eRSFQ (Kirichenko et al., 2012) and RQL (Herr et al., 2011). The initial work of both the eRSFQ and RQL teams involved developing and demonstrating designs for basic processor components such as adders, shifters, and control logic. The eRSFQ team adopted a bit-sliced architecture for their processor while the RQL team used a more traditional bit-parallel architecture. Over the course of the program, 8-bit and 16-bit adders were successfully designed and demonstrated by the RQL team. In addition, other complex control circuits were also designed and demonstrated in RQL. These results established the feasibility of utilizing superconducting electronics to build complex, large scale processors. Designs for 8-bit and 16-bit Turing complete processors with integrated memory were developed and fabricated as part of the C3 program but are still being evaluated. Importantly, the process of developing the various digital logic designs has led to the development of design methodologies and a better understanding of how to utilize the technology to efficiently perform computations.

The memory portion of the program focused primarily on the development of magnetic memories that could provide the dense arrays needed to support larger scale computation. In general, superconducting electronics is not currently a dense technology from a fabrication standpoint. This is most obvious in the area of memory where the lack of density translates into a lack of capacity. Magnetic memories provide a solution to this because their bit cells can be potentially small and packed tightly together. Several versions of superconducting magnetic memory were investigated as part of the C3 program including JMRAM and CSHE (Ye et al., 2014; Aradhya et al., 2016; Dayton I. M. et al., 2018). In addition, to meet the immediate memory needs of the processor designs, a non-magnetic memory was also demonstrated during the program. This memory, NDRO, only utilizes JJs and so is significantly less dense than the magnetic memory alternatives. However, its characteristics made it a good fit for implementing register files and other small memory arrays that support the operation of a processor (Burnett et al., 2018).

In addition to the C3 program, other researchers have also been working to develop superconducting digital logic. One such effort that should be mentioned is using another family of logic, AQFP, to develop adders with the goal of eventually building a processor. This work has also shown promise and has had some successful demonstrations (Inoue et al., 2013; Inoue et al., 2015; Narama et al., 2015).

These developments have laid the groundwork for the development of larger, more complex digital and mixed-signal systems using superconducting electronics. Many aspects of the system architecture that is explored in this work build upon the successful demonstrations of these programs. For instance, the memory technologies developed on C3 are critical to enabling the local storage of synapse weights that are required by neuromorphic processors. Similarly, the complex control circuits that have been demonstrated are representative of the control circuits that will be needed to organize and coordinate the activities of the neurons across a neuromorphic processor.

2.3. Superconducting Neuromorphic Development

Many features of Josephson junctions and superconducting electronics are advantageous to neuromorphic computing. Josephson junctions have a well-defined threshold for moving into the voltage state, similar to the threshold for neurons to emit action potentials. Low-loss transmission lines can carry pulses without distortion over long distances, effectively acting as axons and dendrites. Mutually-coupled superconducting loops can weight and store circulating currents, helping to perform synaptic and summing operations. These advantages were noticed by groups in Japan in the 1990s, who proposed and measured (Hidaka and Akers, 1991; Mizugaki et al., 1993, 1994; Qian et al., 1995; Rippert and Lomatch, 1997) the first Josephson-based circuits to make simple perceptron neural networks. Work has continued since then, especially over the last 10 years or so. In this section we review recent developments, focusing specifically on the biological realism of the soma, analog synaptic weighting, and extension to larger networks.

The Hodgkin-Huxley model (Hodgkin and Huxley, 1990) is the standard for the dynamics of the action potential generated at the soma. It describes the opening and closing of sodium and potassium ion channels which allow the bi-lipid membrane of the axon to charge up (polarize) and discharge (depolarize), causing the rise and fall of the neural spike. The Josephson junction soma, or JJ soma, is a circuit of two Josephson junctions in a superconducting loop which displays very similar dynamics (Crotty et al., 2010). The two junctions act like the sodium and potassium channels, one allowing magnetic flux to charge up the loop and the other allowing flux to discharge from the loop. The result is a soma with biologically realistic dynamics that are similar to those of the Hodgkin-Huxley model.

The degree of biological realism present in a mathematical neuron model was addressed by Izhikevich (2004), who identified 20 dynamical behaviors possible for neurons. Not all behaviors are present in all neurons, but the more behaviors that a model is capable of generating, the more biologically realistic it is. The Hodgkin-Huxley model, for example, obtains 19 of the 20 behaviors. Recent work in benchmarking the JJ soma has obtained 18 of the 20 behaviors (Crotty et al., in preparation), making it more biologically realistic than alternative neuron models like the Nagumo et al. (1962) or Rose and Hindmarsh (1989). This high level of biological realism is very impressive considering it is obtained with only two Josephson junctions and the circuit is capable of running at very high clock rates ($\sim$20 GHz).

Biological synapses are responsible for coupling neurons together. They receive the action potential as their input and feed forward a small current to the downstream neuron, an action which is mediated through a chemical system of neurotransmitters. The amount of the feed-forward current is dependent on the strength or "weight" of the synapse which can, in principle, take on many values. Excitatory synapses have a positive weight and bring the downstream neuron closer to threshold; inhibitory synapses have a negative weight and push the downstream neuron away from threshold. Over time synapses can change their weight due to the coincident firing of their connecting neurons, a feature called plasticity. Plasticity allows for unsupervised training in artificial neural networks.

In order to mimic this behavior in an electrical circuit, a memory element is necessary to hold the weight of the synapse. It is possible to use a superconducting loop as this memory, holding a value of flux proportional to the synaptic strength; however, memory circuits based on loops tend have a large footprint. Following the variety of magnetic memories in the computer industry, a new kind of native synapse has recently been developed (Schneider et al., 2018a,b) using a magnetic doped Josephson junction. By putting small magnetic nanoparticles inside the insulating barrier between the two superconductors, the critical current of the junction can be changed. Since one can put many of these particles in a single barrier and they can all be in different orientations, the resulting critical current can take on essentially a continuum of values, making it an ideal memory element for the synaptic strength.

In addition to changing the critical current, these magnetic nanoparticles can alter their orientation in response to a current pulse across the junction, provided there is an external magnetic field applied parallel to the junction. This allows for the possibility of inherent plasticity: the pulsing of action potentials applied to the magnetic junction changes the orientation of the nanoparticles and hence the critical current. If this junction is embedded in a synapse circuit in which the critical current encodes the value of synaptic strength, then action potentials which arrive at this circuit will "train" the junction and alter its weight, similar to plasticity in a biological neuron.

In the analog domain, the maximum fan-in and fan-out of single neurons will be about 100 or so, limited by parasitic inductances and fabrication tolerances (Schneider and Segall, 2020). Meanwhile, in the human brain each neuron is connected to about 10,000 synapses, on average. Although it is not clear that this level of connectivity is necessary to do interesting and novel computations, increasing it above the all-electronic level of 100 is certainly worth pursuing. Toward that end, optoelectronic neurons have been recently been proposed (Shainline et al., 2016; Buckley S. M. et al., 2017; Buckley et al., 2018; Shainline et al., 2019). These neurons, operating at low temperatures, receive single photons and produce faint photonic signals. In between, photonic communication can be used to route synaptic signals over long distances through the network. Several aspects of such an integrated system have been demonstrated, including light sources, detectors and full optical links at 4K (Buckley S. et al., 2017) and passive photonic routing networks utilizing multiple planes of waveguides (Chiles et al., 2018). The superconducting neurons which interface with these photonic networks are similar kinds of Josephson neuron circuits (Shainline et al., 2019; Shainline, 2020).

In short, there is a growing "toolbox" of superconducting neuromorphic circuits which can be incorporated into spiking neural networks. One can, in fact, use these circuits on their own to design a fully-dedicated, analog neuromorphic processor. Our approach has been instead to combine some of these analog neuromorphic circuits with superconducting digital logic in a

mixed-signal configuration, which we believe will ultimately result in more flexibility, scalability and programmability.

3. SUPERCONDUCTING MIXED-SIGNAL NEUROMORPHIC ARCHITECTURE

This work builds upon recent developments in SCE and superconducting neuromorphic systems by proposing a novel mixed-signal SCE neuromorphic design: BrainFreeze. The proposed architecture combines previously explored bio-inspired analog neuron circuits with established digital technology to enable scalability and programmability that is not possible in other superconducting approaches. This is because the digital SCE components of the architecture facilitate time-multiplexing, programmable synapse weights, and programmable neuron connections. The time-multiplexing supported by the architecture allows multiple neurons in the simulated network to take turns using some of the same physical components, such as the pipelined digital accumulator. This helps to improve the effective density of the hardware. Communication between neurons in BrainFreeze is performed by a digital network like the one used by other large scale neuromorphic approaches. The digital network allows the architecture to share the wires that are used to connect the neuron cores between multiple simulated neurons. This avoids the need to provide a dedicated physical wire to connect each pair of neurons and thereby greatly improves scalability. The arbitrary connectivity provided by the digital network also allows BrainFreeze to implement a wide variety of neural network organizations by reprogramming the routing tables in the network. In this way, BrainFreeze leverages both recent developments in SCE digital logic and innovations from large scale semiconductor neuromorphic architectures to overcome the primary challenges facing SCE neuromorphic systems.

The BrainFreeze architecture is comprised of 7 major components in its most basic form: control circuitry, a network interface, a spike buffer, a synapse weight memory, an accumulator, a DAC, and at least one analog soma circuit. A block diagram of the overall architecture can be seen in **Figure 2**. We refer to one instance of this architecture as a Neuron Core. The control circuitry ensures that the correct spike buffer entries, synapse weights and other states are used on each time step. The network interface handles address event representation (AER) (Mahowald, 1992; Boahen, 2000; Park et al., 2017) packet formation and interpretation as well as interactions with the network router. The spike buffer temporarily stores incoming spike messages until it is time to apply them to the neuron circuits. These buffers are implemented with Josephson transmission lines and should be very compact compared to other components. Multiple buffers may be needed in the case of multiplexing so that the spikes for each multiplexed neuron can be kept separate. The synapse weight memory stores the synapse weights for all of synapses implemented using the core. The accumulator is used to sum up the total incoming weight that should be applied to the neuron circuit during each time step. In order to provide a single answer as quickly as possible a tree of adders is used in the accumulator. The superconducting DAC is responsible for turning the digital output of the accumulator into the signal needed by the analog soma circuits. Finally, one or more analog soma circuits are included that can implement a biologically suggestive neuron model, such as the JJ soma (Crotty et al., 2010). These components provide the critical functionality that is required to implement a neuron. This architecture allows for the inclusion of bio-inspired analog neuron circuits in a large scale, programmable neuromorphic system.

One of the challenges that must be overcome to enable competitive large scale SCE neuromorphic systems is the limited density of SCE digital logic. SCE fabrication nodes are currently many times larger than those of CMOS. As a result, fewer SCE components can fit on a chip than is the case with CMOS. In the case of large scale neuromorphic systems this density disadvantage can result in a prohibitively small number of implementable neurons. To overcome this situation, multiplexing is used to improve the effective density of BrainFreeze's neuron core. In addition, SCE supports clock rates that are significantly faster than CMOS, so components can be multiplexed to a high degree without significantly slowing down the system relative to current neuromorphic approaches. However, multiplexing introduces latency and memory capacity requirements that must be addressed by the architecture.

Memory capacity has also been a major challenge for SCE systems in the past. This is especially concerning for neuromorphic approaches where large amounts of memory are typically needed to store synapse weights and information about the connections between neurons. To overcome this challenge, BrainFreeze leverages recently developed superconducting memory technologies and multi-temperature memory system organizations in order to ensure that sufficient memory is available to the system. In addition, BrainFreeze implements prefetching to hide some of the longer latencies associated with accessing information at different levels of its memory hierarchy.

The overall goal of the BrainFreeze architecture is to enable a combination of speed, efficiency, scalability, programmability, and biological suggestivity that is not possible in other state-of-the-art approaches. In addition, in order to help address the need for reduced training times, BrainFreeze endeavors to achieve these goals while allowing for the possibility of on-line learning support. To accomplish these goals a mixed-signal approach has been adopted but this introduces additional challenges that must be addressed by the architecture. Detailed trade-off analysis is needed to determine the best way to overcome each challenge and to establish the feasibility of the overall approach.

4. METHODOLOGY

The data presented in this paper is the result of detailed numerical analysis based on results from digital logic experimental demonstrations. In particular, the results are used to inform the area and latency estimates of various components of the proposed neuromorphic system. For instance, the sizes of demonstrated NDRO and JMRAM memories are used when estimating the area required for synapse storage. To estimate the area and latency of a

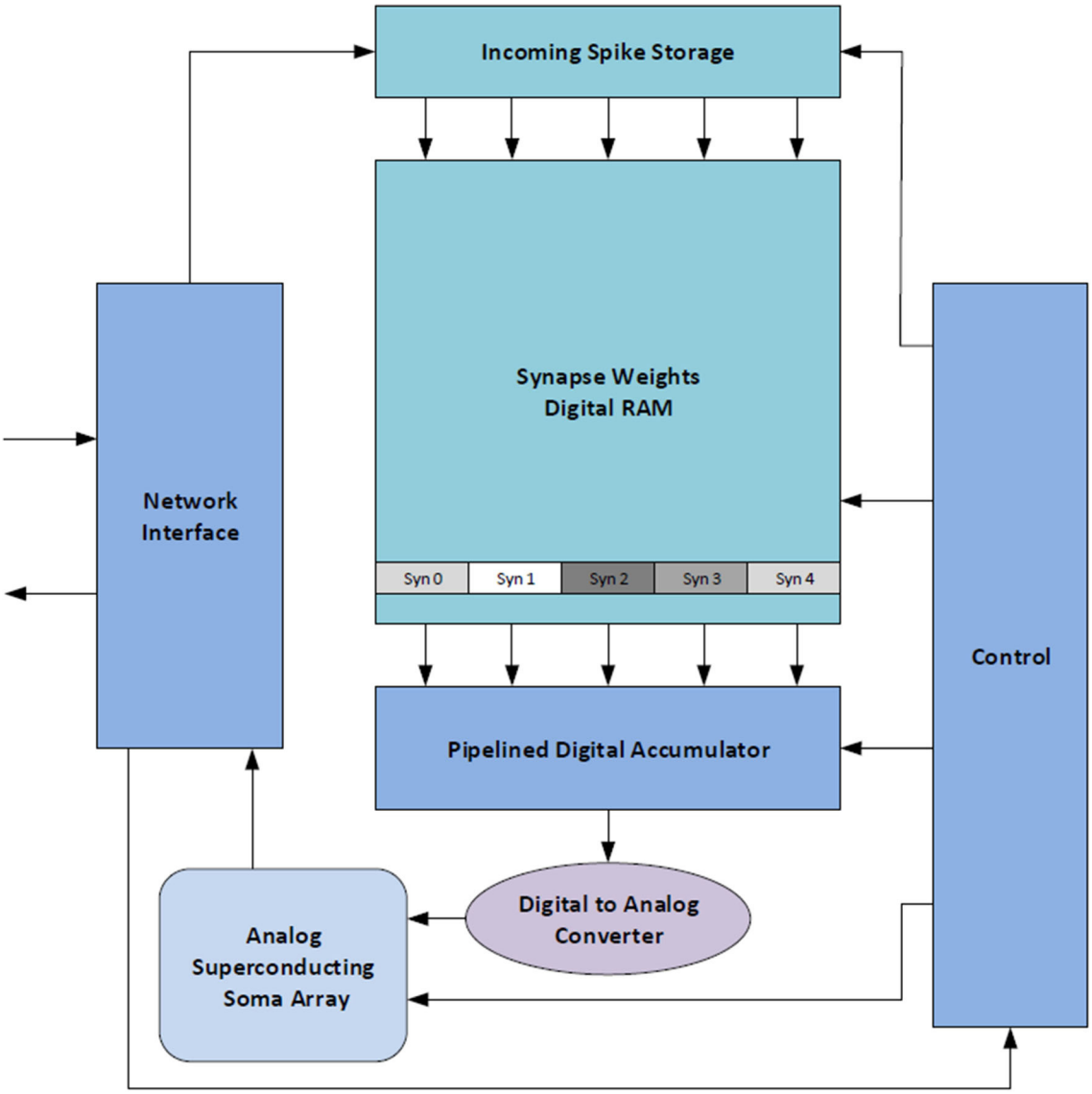


FIGURE 2 | A high-level block diagram of the proposed mixed-signal SCE neuromorphic architecture that includes all of the major components. We refer to this collection of components as a Neuron Core. This architecture combines the scalability and programmability enabled by superconducting digital logic with biological suggestivity enabled by superconducting analog circuits.

neuromorphic core, the appropriate quantities of the component values are added together. Comparing these estimates for different configurations of the BrainFreeze architecture allows us to explore the potential trade-offs of this design. The estimates also help to establish the feasibility of the proposed architecture and provide motivation for future research.

4.1. Analysis Parameters

Table 2 presents the various parameters that are used throughout the analysis in this work. The memory and ALU parameters in this table come from circuits and designs that were developed for experimental demonstrations. For the purposes of this work, the memory parameters are used to estimate the area and delay required to store synapse weights and network routing information. Similarly, the ALU parameters are used to estimate the area and delay of the accumulators and control required in the BrainFreeze core. The area of an ALU is actually more than twice the area of a single adder because the ALUs included logical operations, an adder, a shifter, and control. However, this additional area is included to account for the tree of adders needed by the architecture. In other words a design with 8 parallel adders actually needs 15 adders organized as a tree to quickly produce a single result. The total area required by this tree is estimated as roughly the area of 8 ALUs or 1 ALU for each parallel adder in a design. This rough estimate is used because it is an overestimation and because the ALUs were the most

TABLE 2 | The latency and size parameters that were used to generate the analysis.

	X Size	Y Size	Latency
NDRO (1 bit)	55 μm	55 μm	3.125 ps
JMRAM (1 bit)	5 μm	5 μm	0.25 ps (read) 100 ps (peripheral)
8-bit ALU	844 μm	1,166 μm	550 ps
16-bit ALU	1,771 μm	2,860 μm	650 ps
Spike buffer (1 bit)	12 μm	24 μm	
MCM	200 mm	240 mm	
Cabinet (1,869 mm Tall)	600 mm	912 mm	

The NDRO, JMRAM, and ALU parameters were taken from designs that were used for experimental demonstrations (Dayton I. et al., 2018; Hearne et al., 2018; Vesely et al., 2018).

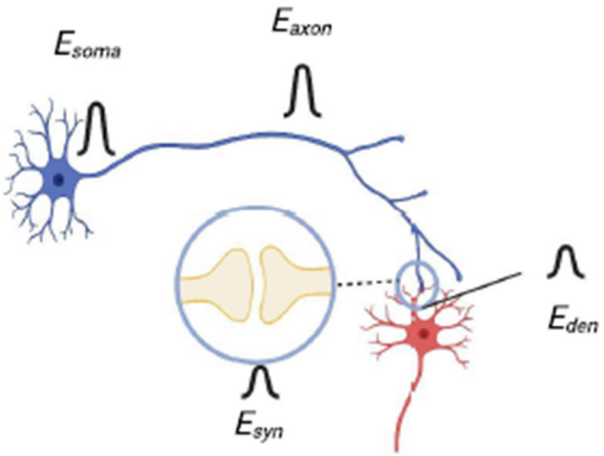


FIGURE 3 | A diagram of neurons connected by a synapse that shows the placement of the components of the neurons and the corresponding energy that is associated with action performed by each component.

suitable demonstrated designs available. The parameters for both 8-bit and 16-bit ALUs are included to show the area and latency impacts of both design choices. For this study only 8-bit adders are considered. For the purposes of this study the latencies and sizes of the DAC and soma are negligible compared to the other components of the architecture. This is possible because there are orders of magnitude fewer DACs and somas in the system than other components such as memory cells and logic gates. As a result, the impact of these components on the overall area and latency of the architecture is minimal. The MCM parameters were selected so that they could theoretically be made using a 300 mm wafer and so that 3 MCMs could fit side by side in a cabinet. The parameters for a cabinet are the dimensions of a standard server rack.

4.2. Energy Efficiency Estimation

One of the main figures of merit (FOM) for energy efficiency in neuromorphic computing is Synaptic Operations per Second per Watt (SOPS/Watt), first introduced by IBM with their True North system. The main argument for this FOM is that it combines power and speed; a SNN which spikes twice as fast will dissipate (at least) twice as much power. For complete, working systems, this number is easily calculated by the following equation:

$$FOM = \frac{fs'}{P} \tag{1}$$

Here f is the average spiking frequency of the SNN, s' is the average number of active synapses and P is the total measured power dissipated by the network. The number of active synapses s' is a fraction of the total number of synapses s, typically in the 30–50% range but theoretically as high as 100%. The most efficient semiconductor SNNs achieve on the order of 10^{10} SOPS/Watt.

For systems which have yet to be fully built, we can estimate this FOM by summing up the power dissipated by the different components of the system. **Figure 3** shows a schematic of a SNN showing the placement of somas, axons, synapses and dendrites. Here we assume there are N total somas and s total synapses; since there is one axon for each soma and one dendrite for each synapse, then there are also N axons and s dendrites. Let us assume that the energy dissipated per spiking event is E_{soma}, E_{axon}, E_{den} and E_{syn} for each soma, axon, dendrite and synapse, respectively. Then we can write the total power dissipation P:

$$P = fNE_{soma} + fNE_{axon} + fs'E_{syn} + fs'E_{den} \tag{2}$$

The upper limit of P occurs when $s' = s$; this is the worst-case scenario. If we assume that the energy dissipation in the axon is small compared to the soma ($E_{soma} >> E_{axon}$) and the dissipation of the dendrite is small compared with the synapse ($E_{syn} >> E_{den}$), then the second and the fourth term in Equation (2) can be ignored. Plugging into Equation (1), we then obtain (for the worst-case scenario where $s = s'$):

$$FOM = \frac{1}{(E_{syn} + (\frac{N}{s})E_{soma})} \tag{3}$$

Finally, in large networks it is often true that $E_{syn} >> (N/s)E_{soma}$, since the number of synapses s are of $O(N^2)$. In that case FOM is simply $1/E_{syn}$. Of course, the above all assumes that any control circuitry or other extraneous energy dissipation in the network is either small or accounted for in one of the four energy terms; otherwise additional contributions must be added to the power in Equation (2). For BrainFreeze, we use $1/E_{syn}$ for our FOM; we are in the limit where $E_{syn} >> (N/s)E_{soma}$ and where our axons and dendrites, being composed of superconducting transmission lines, have negligible power dissipation. To calculate E_{syn}, we count the number of Josephson junctions per synapse and assume that each junction dissipates 10^{-19} Joules per pulse (true for a junction with $I_c = 300\,\mu A$). In addition, we multiply by a factor of 500 to account for the cooling. For example, the 8-bit design of BrainFreeze has about 6,000 junctions per synapse, giving a FOM of 3.3×10^{12} SOPS/Watt, about 70x higher than True North (4.6×10^{10} SOPS/Watt).

5. TIME-MULTIPLEXING

One of the primary motivators behind adopting a mixed-signal approach is to enable the sharing of axon wires and core components between different simulated neurons. This time-multiplexing capability can improve scalability by allowing the system to support a larger number of virtual simulated neurons than the physical hardware would natively support. However, the improved scalability comes at the cost of increased overall simulation run-time, increased area to store the state of the additional virtual neurons, and increased complexity in terms of the organization of events in the simulated neural network. The increased control and organization complexity includes design decisions such as determining when to switch to a different virtual neuron and how to handle out of order or delayed AER packets as well as other considerations. The design and evaluation of this control logic is not relevant to the current study so we leave those tasks for future work.

The impact of time-multiplexing on simulation runtime is determined by the time it takes to update the model of each virtual neuron and the degree of multiplexing. Neuron update latency can be greatly affected by the number of inbound spikes that can be processed in parallel and the memory access delays required to access synapse weights. The number of parallel adders included in the accumulator determines how many spikes can be processed at a time. In some designs, spikes that arrive on one time step could be processed on a later time step if there is not sufficient parallel capacity to accommodate them. It is more likely, however, that such excess spikes will need to be discarded. Therefore, selecting the appropriate amount of parallelism to include affects the functionality of the system in addition to its latency. However, it is unlikely that very many of the presynaptic neurons will fire during the same time step, provided that each time step is sufficiently short. Therefore, a modest number of adders, such as 4 or 8, will likely be sufficient in many implementations of BrainFreeze. It is important to note that the latency of superconducting logic is low enough that considerable multiplexing can be employed before the simulation time step approaches that of most semiconductor architectures.

The degree of multiplexing that can be supported by the architecture is limited by the area that is available to implement the synapse memory component of the neuron core. The physical neuron hardware could potentially perform calculations related to a different virtual neuron on each time step of the system. This means that signals and states need to be preserved until the hardware is again performing calculations related to the appropriate virtual neuron. As a result, the memory system needs to have enough capacity to store all of the required information for each virtual neuron. The capacity of the memory component is directly related to its area so once the available area has been used, no additional capacity can be added to support more virtual neurons. The memory system also needs to be configured to support the number of parallel accesses required by the number of adders used by the architecture. This could be achieved by assigning particular sub-arrays of memory to each adder. In this study we assume one memory bank is provided for each adder and that the bank is sized to accommodate the number of virtual neurons present in the system.

Figure 4 illustrates the trade-off that exists between the degree of multiplexing, area and latency in the design space of the neuron core. The trade-off shown in **Figure 4** captures the worst case scenario for a neuron core where a spike is received on all 1,000 synapses in the same time step. The actual number of spikes that may be seen per time step is a function of design decisions that are beyond the scope of this study. As a result, selecting the optimal number of adders for BrainFreeze is a subject for future work. For the purposes of this study the highly unlikely worst case is used as a stress test to evaluate the impact of neuron update latency. We use 8-bit adders for this trade-off analysis because the synapses in our proposed system have 8 bits of resolution. In this figure we can see that the area impact of increasing the number of adders is minimal until 32 or more adders are used. Prior to that point, the area is primarily affected by the degree of multiplexing due to the memory that is required to store the synapse weights of the additional virtual neurons. The impact of memory technology choice is emphasized by these results as just 2 virtual neurons worth of synapses require more area in NDRO than 32 virtual neurons required in JMRAM. These results suggest that 16 adders may represent a reasonable upper bound design point for a 1,000 synapse BrainFreeze system. A design with 16 adders provides coverage for many simultaneous spike events but does not require too much area or introduce too much delay. A smaller number of adders may be more reasonable if smaller numbers of spikes per time step is typical in a system or if update speed is not the most important design aspect.

6. SYNAPSE DENSITY

A primary metric of feasibility for SCE mixed-signal neuromorphic designs is the number of synapses that can be implemented per neuron using this approach. The quantity and resolution of the synapses that a neuromorphic architecture can implement can greatly affect the types of networks that can be implemented on that architecture and the performance of those networks. Efficiently ensuring the largest number of synapses, each with an acceptable resolution, is a major consideration of all neuromorphic architectures. This is particularly true in superconducting neuromorphic systems where the memory capacity required to store synapse weights is at a premium. The sizing and organization of the synapse memories need to be balanced to provide the maximum number of synapses possible without negatively affecting other aspects of the neuron core design. For this early study of the potential of BrainFreeze, we have settled on an 8-bit synapse resolution. 8-bits is more resolution than is used by many other neuromorphic architectures and provides a wider range of synapse weights.

Synapse latency can be a major contributor to overall core latency. If the memory arrays are too large they can incur very long latencies which will negatively affect performance. In order to minimize synapse weight access latency while providing adequate storage capacity, the memory will likely need to be organized into many banks. Each bank will then hold a subset

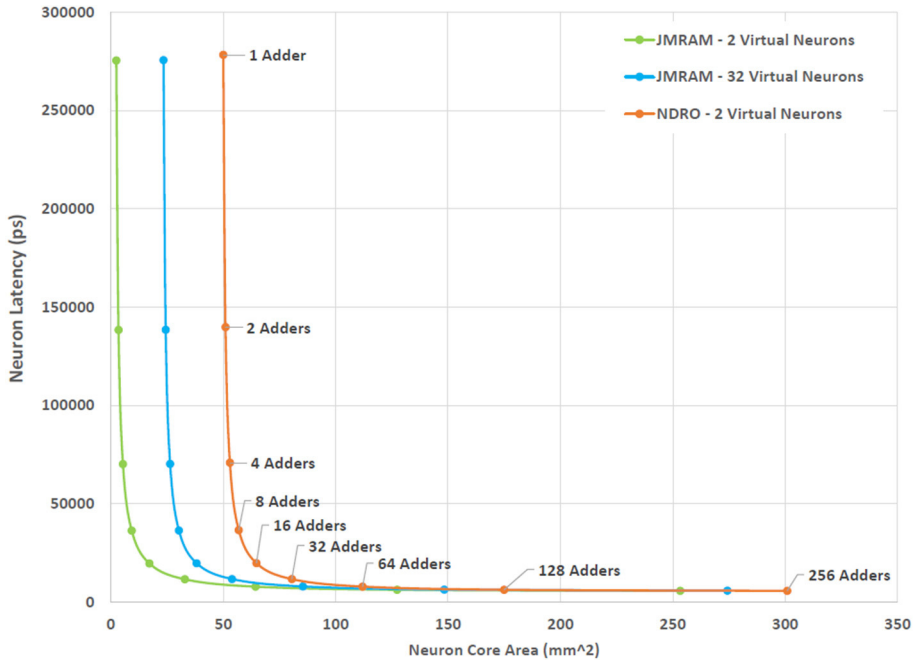


FIGURE 4 | The effect of increasing the degree of multiplexing and parallelism on the area and latency of the BrainFreeze neuron core. Increasing the multiplexing improves the effective neuron density of the architecture but requires additional memory. Adding more adders enables the neuron core to process incoming spikes faster but also adds area. The latency presented here is for handling the worst case situation where all 1,000 inbound synapses receive a spike. The adders used in this analysis are 8-bits wide.

of the total synapse population for each virtual neuron. This organization introduces the issue of bank collisions where several synapse weights that all map to the same bank are needed at the same time. Improperly handling this situation could lead to a degradation in performance or functionality as spikes will either be delayed or discarded as a result of the collision. One possible solution to this issue is aggressively banking the memory array such that each array is so small that very few synapses for a particular virtual neuron will map to the same bank. This would greatly reduce the odds of a collision but would introduce area and delay overheads due to the additional hardware required to interface with and manage so many banks. For the purposes of this study we consider only the area required for the memory to support some number of neurons.

In order to determine how many synapses could be accommodated on typical die sizes, we calculated the area required for a neuron core as the number of virtual neurons per core was increased. The area needed for other components of the neuron core was also included in this calculation. For this study a 16 adder configuration was used based on the result presented in Section 5. The results of these calculations can be seen in **Figure 5**. Here we can see that both NDRO and JMRAM can support more than one virtual neuron using typical die sizes. In fact, hundreds of thousands of synapse weights can fit on a single die when a relatively dense superconducting memory technology, like JMRAM, is used to store them. Nevertheless, even less dense technologies, such as NDRO, still enable synapse counts that are appropriate for near term demonstrations. Importantly, these memories provide sufficient capacity to act as local buffers for systems that use a backing store to enable even larger numbers of virtual neurons. The implications of this capability are discussed later in Sections 8 and 9.

7. NETWORK CONFIGURATION

Programmability is another important aspect of a neuromorphic architecture because it allows the system to implement different neural networks using the same hardware. A key aspect of programmability is that the connections between neurons need to be configurable so that a wide variety of neural network topologies can be implemented. This can accomplished by implementing the connections between neurons as a computer network using routers and shared wires. Each neuron is given a network address and spike messages are routed through the network based on those addresses. AER representation could be used such that each spike message contains all the information needed by the post synaptic neuron core to apply the spike to its current state. This information could be as simple as indicating the presence of a spike on that synapse during this time step or could include finer grained temporal information or other spike characteristics.

The architecture of the digital network between the neurons greatly affects the functionality and speed of the overall network. In general, there are three primary decisions regarding the network architecture of neuromorphic systems: the network

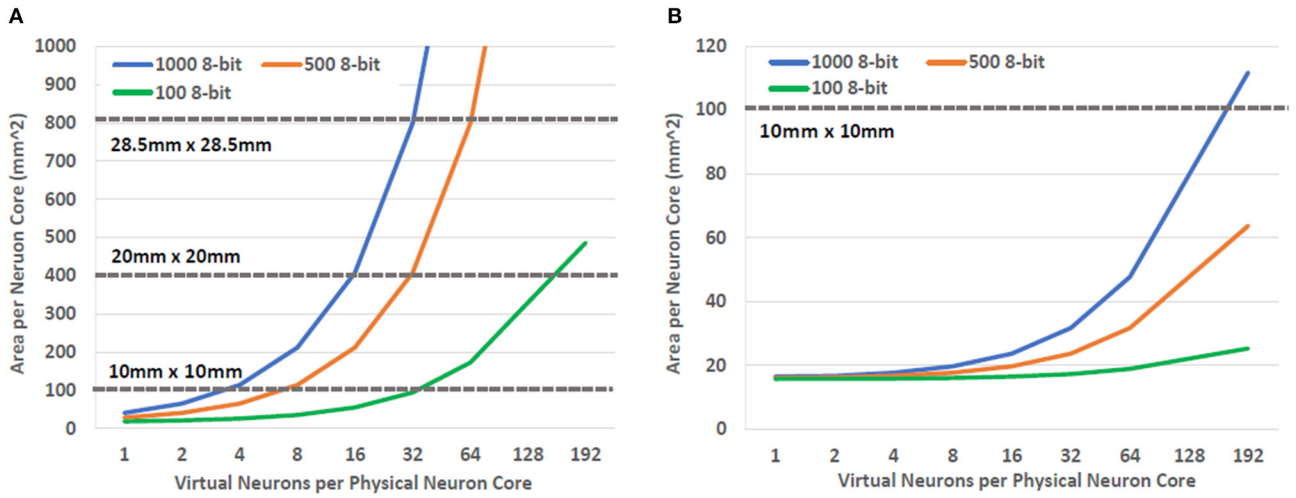


FIGURE 5 | The die area required to support BrainFreeze neuron cores as the number of synapses per neuron and the degree of multiplexing is increased. These results show that **(A)** NDRO and **(B)** JMRAM based designs have sufficient capacity to act as local buffers for neuron information. Both NDRO and JMRAM also provide enough capacity to directly support small scale and larger demonstrations, respectively.

topology, the packet type, and the routing style (Young et al., 2019). The topologies that are often employed by state-of-the-art neuromorphic architectures are the 2-D mesh and the tree. Of these two topologies, the 2-D mesh topology provides better bisection bandwidth due to the larger number of links in the topology compared to a tree topology (Benjamin et al., 2014). However, the tree topology generally has fewer hops between destinations than the 2-D mesh and so it provides better latency. An important aspect of both of these topologies is that they can be implemented using routers with a relatively small number of ports (radix). The radix of the router (number of ports) greatly affects its complexity and increasing the number of ports rapidly grows the size of the router.

A tree topology seems to be a good design choice for SCE neuromorphic systems. The limited density of SCE means that topologies that require higher radix routers are probably not a reasonable design choice at this time. Therefore, the ability to implement a tree topolgy using low radix routers satisfies an important requirement of BrainFreeze. In addition, the superconducting wires used to communicate in BrainFreeze provide a potentially useful advantage over room temperature wires in terms of their efficiency over long distance. A comparison of the energy required for long distance communication using both traditional semiconductor as well as superconductor wires can be seen in **Figure 6**. This difference in communication energy means that topologies that feature longer connections may be better suited to implementation in SCE-based systems. So, the longer connections that can exist in a tree provide additional motivation to use this topology in an SCE-based system like BrainFreeze.

In addition to the topology of the network, designers must also determine what style of packets to use and how to route them. Packets can be either unicast or multicast, meaning that one packet can either have one destination or many. Neural spikes tend to have multiple destinations so a multicast network seems to make the most sense. Networks that support multicast packets can maximize bandwidth efficiency by only generating additional packets at the routers that have destinations on more than one port. Source address routing can help to minimize the logic required in the router to provide this functionality. However, in order to support source address routing a large memory is required in the router to store all of routing information for each source address. Destination address routed packets, on the other hand, include the connectivity information in the packet itself. So they can be implemented without the need for a large memory in the router. These additional requirements in terms of memory and router complexity mean that the multicast and source routed approaches would require routers that would be prohibitively large and slow in an SCE neuromorphic system. As a result, a unicast, destination routed network is likely the best choice when only on-die memory is available to the router.

Even in a unicast, destination routed network, the memory requirements of the destination neuron address storage quickly dominate the overall memory requirements of the architecture as the size of the network grows. In other words, as the network size grows to millions of neurons or more, we need more memory to store the addresses for the post synaptic neurons than we need to store the synapse weights. This can be seen in **Figure 7** where the area required to store neuron addresses increases rapidly as the size of the overall simulated neural network grows. Therefore, JJ-based memories, such as NDRO, are likely only sufficient to provide storage for both synapse weights and post-synaptic addresses in networks that contain a few thousand neurons or less. JMRAM can support larger numbers of neurons but would require a prohibitive number of chips to support the hundreds of millions of neurons that some state-of-the-art approaches can support. The addition of a backing store to the system could alleviate this situation by providing additional capacity for

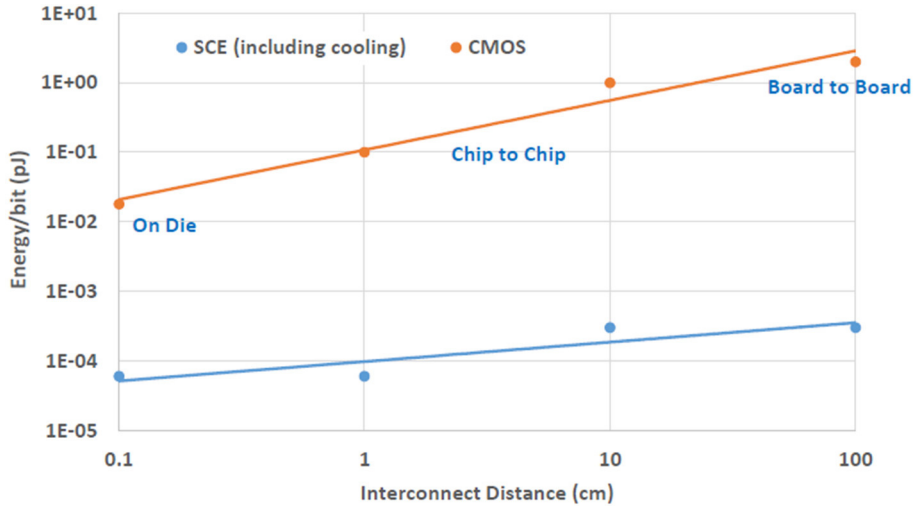


FIGURE 6 | The energy cost of communicating across different distances in both CMOS and SCE. The efficiency of SCE wires provide a distinct advantage that can be leveraged by networks of BrainFreeze neuron cores. The CMOS data in this figure is from Borkar (2011).

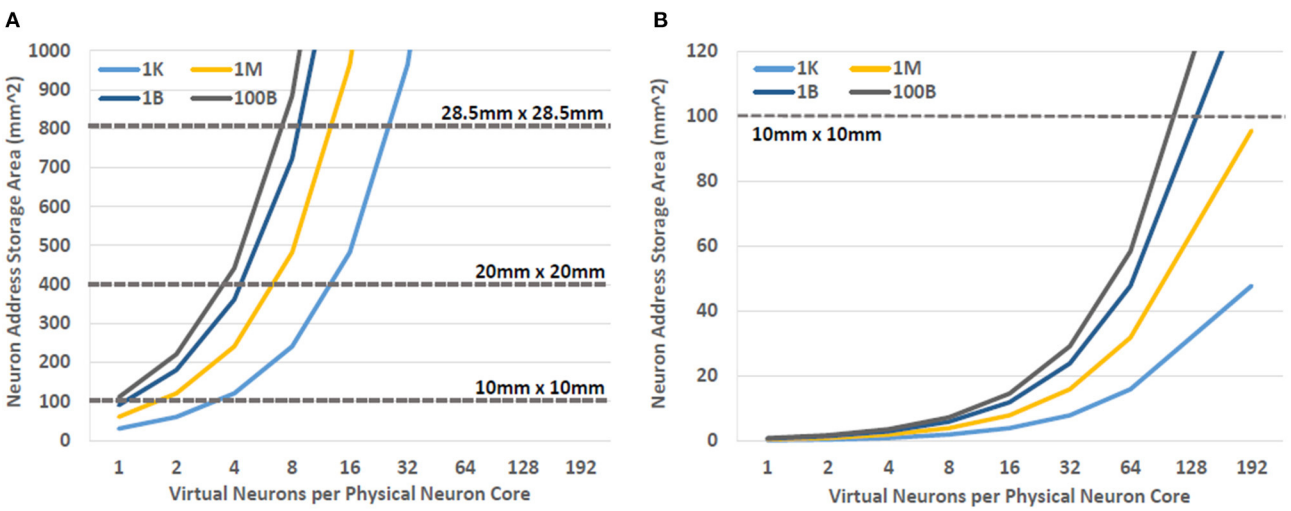


FIGURE 7 | The area required to store the post-synaptic addresses for neural networks with increasing numbers of neurons. As the network grows the area required to store this information can become prohibitive. The incorporation of a backing store may be an effective way of addressing this need for additional memory capacity. **(A)** NDRO and **(B)** JMRAM.

address storage for very large systems. Furthermore, a backing store might also enable support for multicast, source routed networks by providing sufficient capacity for the larger routing tables that that communication scheme requires.

8. MEMORY HIERARCHY

Providing enough memory to meet the requirements of each component in the BrainFreeze architecture is particularly challenging due to the relatively limited memory density available in SCE memory technologies. One solution is to include chips in the architecture that serve as off-die memories for the neuron cores. Theoretically, these off-die memories would free up space on the neuron chips so that more neuron cores could be implemented on those chips. Similarly, the dedicated memory chips should provide additional memory capacity because the memory arrays should fit more tightly together. This approach also has the benefit of easing any integration issues that might arise from incorporating magnetic memories and SCE logic circuits on the same die. However, as **Figure 8** shows, placing neuron cores and synapse memories on separate chips does not actually result in improved neuron core density. This is because removing the memories from the neuron chip does not free up

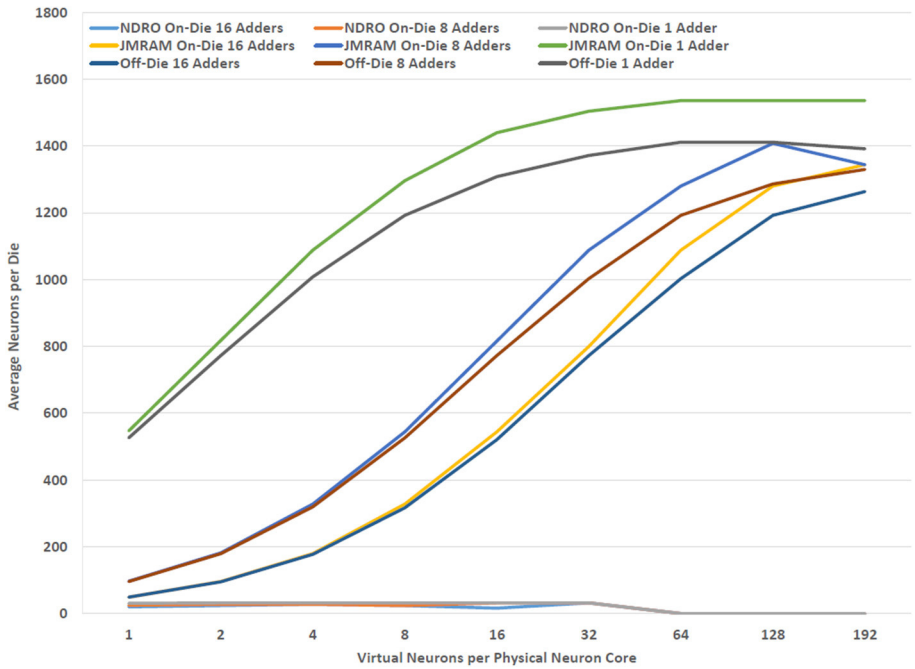


FIGURE 8 | The relationship between effective neuron density and the location and type of memory. Using separate dies for JMRAM does not improve the number of neurons that can be implemented per die in BrainFreeze.

enough space for additional neuron cores to offset the cost of adding the memory chips to the system. Clever floorplanning of neuron cores could potentially address this though that is beyond the scope of this study.

The available on-die memory is sufficient for storing the synapse weights for a reasonable number of virtual neurons. However, the limited capacity of superconducting memories will ultimately limit the number of neurons and synapses that can be implemented using only superconducting electronics. To achieve continued scaling, denser memories are needed. One solution to this problem may be a multi-temperature memory hierarchy that uses semiconductor memory technologies as a backing store for the superconducting memories. An example of such a hierarchy can be found in **Figure 9**. The much denser semiconductor memories enable an overall memory capacity for the superconducting architecture that is comparable to the memory capacities of semiconductor approaches. To alleviate some of the power and heat considerations that would be introduced by adding a large amount of semiconductor memory to the SCE system, the semiconductor memory could be implemented at an intermediate temperature. Cryogenic-DRAM is a technology that could potentially be used to fill this role in the system (Tannu et al., 2017; Ware et al., 2017; Wang et al., 2018; Kelly et al., 2019).

Local superconducting memories will still needed to buffer the data because there is a considerable latency involved with accessing data stored in the semiconductor memories. These local buffers could be much smaller because they would only need to contain a subset of the total system data. This could improve neuron core density, increase the number of synapses that could be implemented per virtual neuron, and result in much better area utilization in off-die memory organizations. In practice, the amount of data that needs to be buffered locally will depend on the type of data. For instance, only the post-synaptic neuron addresses for the current virtual neuron would need to be buffered. However, if the system is supporting spike time dependent plasticity (STDP), then the buffer would need to be large enough to store the synapse weights for multiple virtual neurons. This is because STDP may result in synapse weight changes after a virtual neuron has completed its update. In both cases the buffer needs to have room to store the incoming data that will be used by the next virtual neuron to be handled by the neuron core. So in the address case, the buffer would need to hold two virtual neurons worth of data and in the weight case it would need to hold three or more virtual neurons worth of data. This is still considerably less than the amount of memory that would be needed to store all of addresses and weights for all of the virtual neurons that are assigned to a neuron core. However, the latencies involved with requesting data from the intermediate temperature backing store are prohibitive. A technique is needed to improve or hide these latencies in order for the multi-temperature memory hierarchy to be a viable solution.

9. PREFETCHING

One way to hide the latency involved with retrieving data from a memory is to request it in advance of when it is actually needed.

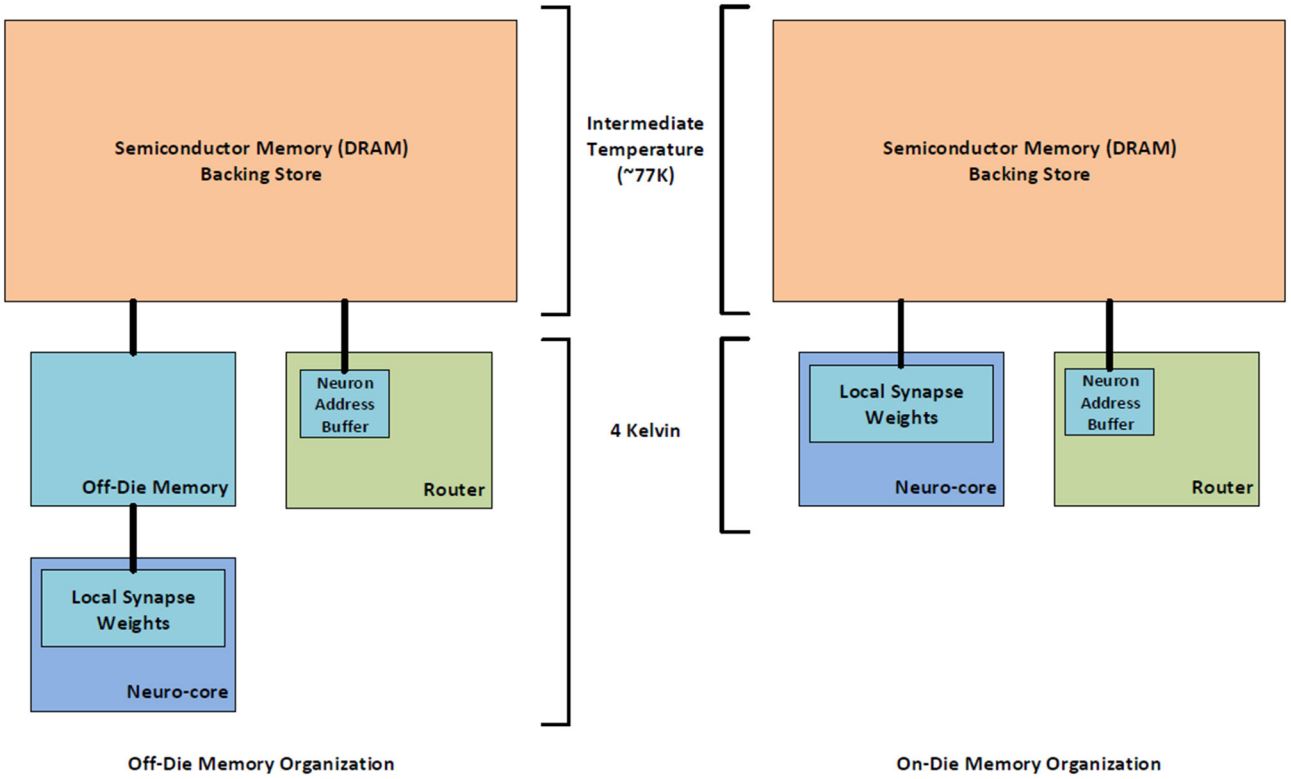


FIGURE 9 | A block diagram of the proposed multi-temperature memory system that would enable the use of cryo-DRAM with the BrainFreeze architecture. This could greatly increase the memory capacity available to BrainFreeze.

That way the data can arrive just when it is needed and the latency involved will not affect the performance of the system. This technique is called prefetching. The problem with prefetching is that it's not always easy to know what should be requested from memory. As a result, prefetching can sometimes result in performance degradation as it can tie up memory resources with requests for data that will not be needed anytime soon. Therefore, the key to effectively utilizing prefetching is correctly anticipating which data will be needed in the near future.

Neuromorphic systems like BrainFreeze are particularly well suited to prefetching because their data accesses follow predictable patterns. This makes determining which data will be needed in the near future relatively straightforward. For instance, the addresses and weights that are needed for the next virtual neuron can be fetched while the current virtual neuron is being updated. This deterministic behavior ensures that the data that is prefetched, is data that is likely to be needed. As a result, prefetching can provide an effective way to hide the latencies incurred by using a multi-temperature memory hierarchy. This allows BrainFreeze to preserve its performance advantage even when using a longer latency backing store.

Of course, not all of the relevant addresses or weights are always used during the update of each virtual neuron in the network. As a result, some data will be fetched that is not used and therefore it is possible that some energy will be wasted with this approach. One way to avoid this problem is to use more information from the system to determine exactly what data should be prefetched. For example, the spike buffer holds a record for each inbound spike that will be applied to a virtual neuron when it is next updated. These records can be utilized to specify a subset of the synapse weights to read from the backing store rather than requesting all of the synapse weights that might be needed by a virtual neuron. In this way, unnecessary data accesses due to prefetching can be greatly reduced and degradation to memory efficiency and performance can be avoided. A block diagram of a potential implementation of this scheme is provided in **Figure 10**.

10. COMPARISON TO STATE-OF-THE-ART

Now that we have established some feasible design choices for the BrainFreeze architecture, it's important to evaluate how that architecture might compare against other state-of-the-art approaches. For the purposes of this evaluation we compare against the neuromorphic architectures that were described in Section 2.1. These architectures represent some of the most successful and studied neuromorphic designs that have been proposed. Details regarding the methodology used to produce these comparisons can be found in Section 4.2 of this paper. This analysis attempts to compare only the neuron implementations of each approach. As a result, only the components of the BrainFreeze neuron core are

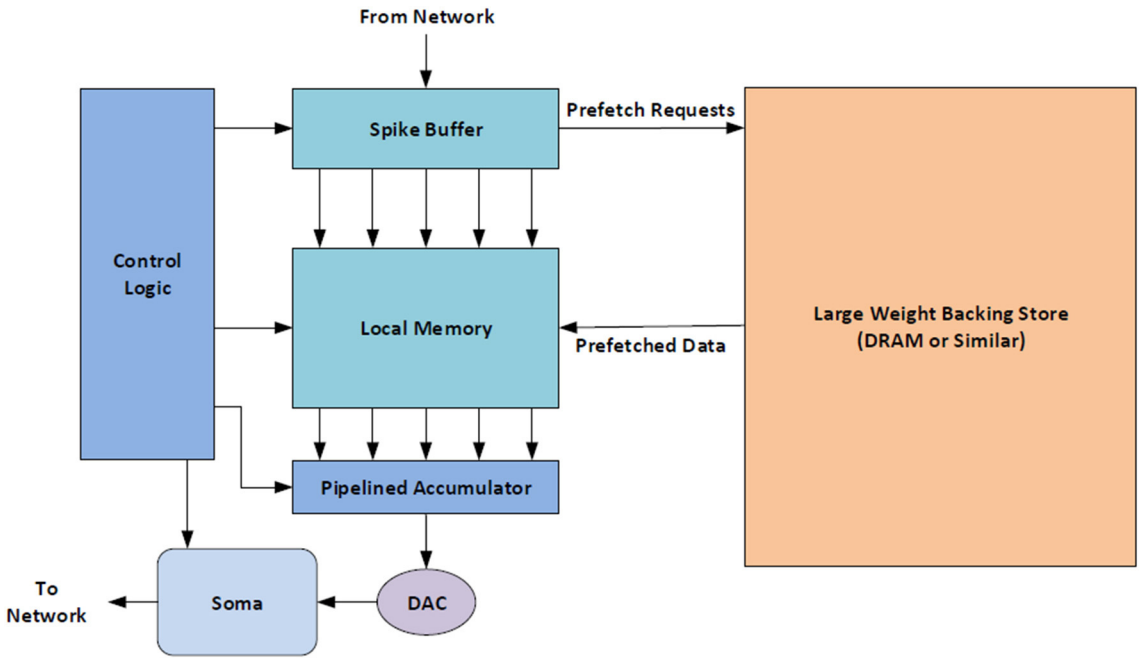


FIGURE 10 | A block diagram of the proposed prefetching system that would use information in the neuron core, such as which synapses received a spike, to better avoid fetching data that will not be needed.

included in these calculations. Other potential supporting hardware, such as a cryo-DRAM backing store, are not considered. To the best of our knowledge, the values used for other neuromorphic approaches also do not include supporting hardware.

One of the most compelling capabilities of SCE is its ability to support very high clock rates. To determine how this might benefit a neuromorphic system we consider the peak rate at which neuron updates might occur in each of the different architectures. For the purposes of this study we consider such a neuron update to be roughly equivalent to the process it takes to produce a spike in response to incoming signals. The peak spike rate is one of several factors that can affect how quickly a simulation can be performed by the hardware. The values in **Figure 11A** for peak spike rate where determined by consulting the literature to determine the rate at which each established architecture can produce a spike (Benjamin et al., 2014; Stromatias et al., 2015; Amir et al., 2017; Davies et al., 2018). For TrueNorth the system clock frequency is used to estimate the maximum spike rate and for Loihi the mesh-wide barrier synchronization time is used. This peak spike rate is calculated on a per-neuron basis, rather than per-chip, to provide a direct comparison between the designs. The peak spike rate for BrainFreeze was calculated for the ideal situation where only a single action potential arrives and only a single adder is needed for its integration. The spike rate for BrainFreeze includes the latency for each of the major components of the SCE neuron core including memory lookup and digital accumulation. The results of this comparison show that, like BrainScaleS, BrainFreeze is capable of running significantly faster than biological real time. This is an important result as it indicates that the high clock rates enabled by SCE will support the accelerated simulation time scales that will likely be necessary for future AI research.

However, speed alone cannot meet all of the anticipated requirements of future AI research. Efficiency is another important aspect of neuromorphic systems as it can ultimately be a limiting factor on the size of the system that can be reasonably implemented. As a result, very large simulations require efficient neuromorphic systems to effectively host them over acceptable time scales. From the comparison in **Figure 11B**. we can see that some approaches, such as TrueNorth, use extremely efficient hardware to achieve impressive overall energy efficiency despite the relatively slow spiking rate of the hardware. Other approaches, such as BrainScales, are capable of achieving extremely fast spike rates and accelerated simulation times but pay a heavy price for this speed in terms of power so their overall energy efficiency is not as good as other designs. Balancing between speed and efficiency often results in designs that deliver impressive overall energy efficiency but that cannot simulate faster than biological real-time. For most of the past neuromorphic approaches this was acceptable since simulating faster than biological real time was not a primary design goal. BrainFreeze is capable of simultaneously delivering both speed and energy efficiency that is nearly two orders of magnitude better than current state-of-the-art approaches. This is possible because of the low power characteristics of SCE.

A discussion of the computational efficiency of the SCE approach should also take into account the complexity of

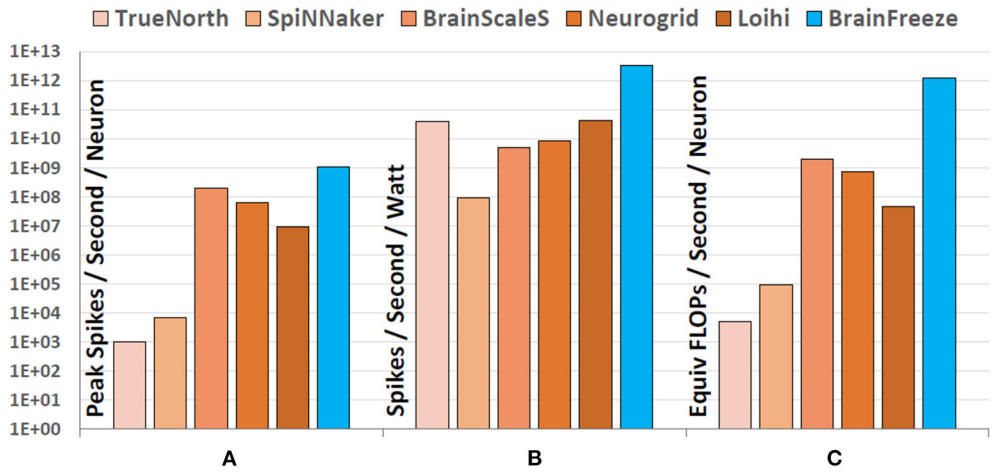


FIGURE 11 | A comparison of state-of-the-art neuromorphic architectures to BrainFreeze in order to demonstrate the potential of the proposed architecture. **(A)** The first comparison looks at how quickly each design can update its neuron model and emit a spike. **(B)** The second comparison looks at how efficient each architecture is at performing its computations in terms of time and power. **(C)** The third comparison looks at the computational complexity of the neuron models implemented by each architecture. Overall, BrainFreeze have the potential to enable significant improvements over current neuromorphic approaches.

neuron model that the approach enables. This is an important point of comparison against existing neuromorphic designs because all of them implement less complex models such as LIF, AdEx or the Izhikevich model. If these designs instead implemented the Hodgkin-Huxley model then it is reasonable to assume that their energy efficiency would greatly decrease either due to the extended neuron update times or due to the power needed to support additional hardware. The ability of BrainFreeze to support a biologically suggestive model similar to Hodgkin-Huxley means that each neuron update is not only happening in less time, it is also more computationally complex. To illustrate this difference in complexity between the neuromorphic approaches we compare the effective FLOPs/Sec. These values are based on the FLOPS per 1 ms simulation time step values that are provided by Izhikevich in Izhikevich (2004). Here we take the 1 ms simulation time step to be roughly equivalent to a neuron update in each of the neuromorphic designs. So, to estimate the equivalent simulation work that is being accomplished by each architecture per second, the FLOPS value is multiplied by the spike rate for each approach. This comparison can be seen in **Figure 11C**. The ability of BrainFreeze to support a model similar to Hodgkin-Huxley means that each neuron update is roughly equivalent to 1,200 FLOPs in a software implementation of the model. This effect combined with the speed of BrainFreeze results in an improvement of nearly three orders of magnitude compared to the other designs.

Overall these results show that a mixed-signal SCE neuromorphic approach could provide significant improvements over the current state of the art in terms of speed, energy efficiency, and model complexity. These improvements are important because they will enable future neural network simulations to run in less time while simultaneously incorporating more biologically inspired functions.

11. ON-LINE LEARNING SUPPORT

Improving the computational efficiency of neuromorphic systems will help to address some of the needs of future neural network simulations but training times also need to be accelerated. **Figure 1** in Section 1 illustrated the immediate need for enhanced training methodologies as training requirements are growing much faster than the capabilities of computer hardware. One way to approach the problem of computationally intensive training is to use on-line learning to perform at least some of the training. On-line learning provides three potentially useful capabilities to a neuromorphic system like BrainFreeze. First, the improved speed and efficiency of BrainFreeze could possibly result in on-line learning reaching an acceptable solution in less time and using less power than other approaches. Second, utilizing on-line learning could allow the implemented neural network to adapt to changes in the input data thereby avoiding the need to completely retrain the network. And third, on-line learning is more biologically relevant and may help to support computational neuroscience experiments. For these reasons incorporating on-line learning into neuromorphic systems could prove to be a valuable capability for future applications and experiments.

On-line learning techniques train the neural network using a stream of input data rather than training the network with an entire set of data prior to run-time. In spiking neuromorphic systems like BrainFreeze, on-line learning techniques can take the form of run-time synaptic plasticity where synapse weights are adjusted as neurons in the system react to inputs. One on-line learning technique that may be particularly well suited to SCE neuromorphic systems is Spike-Time Dependent Plasticity (STDP). STDP is a Hebbian reinforcement learning rule that updates the synaptic weights based on the timing relationship between the input and output spikes of neurons. Hebbian

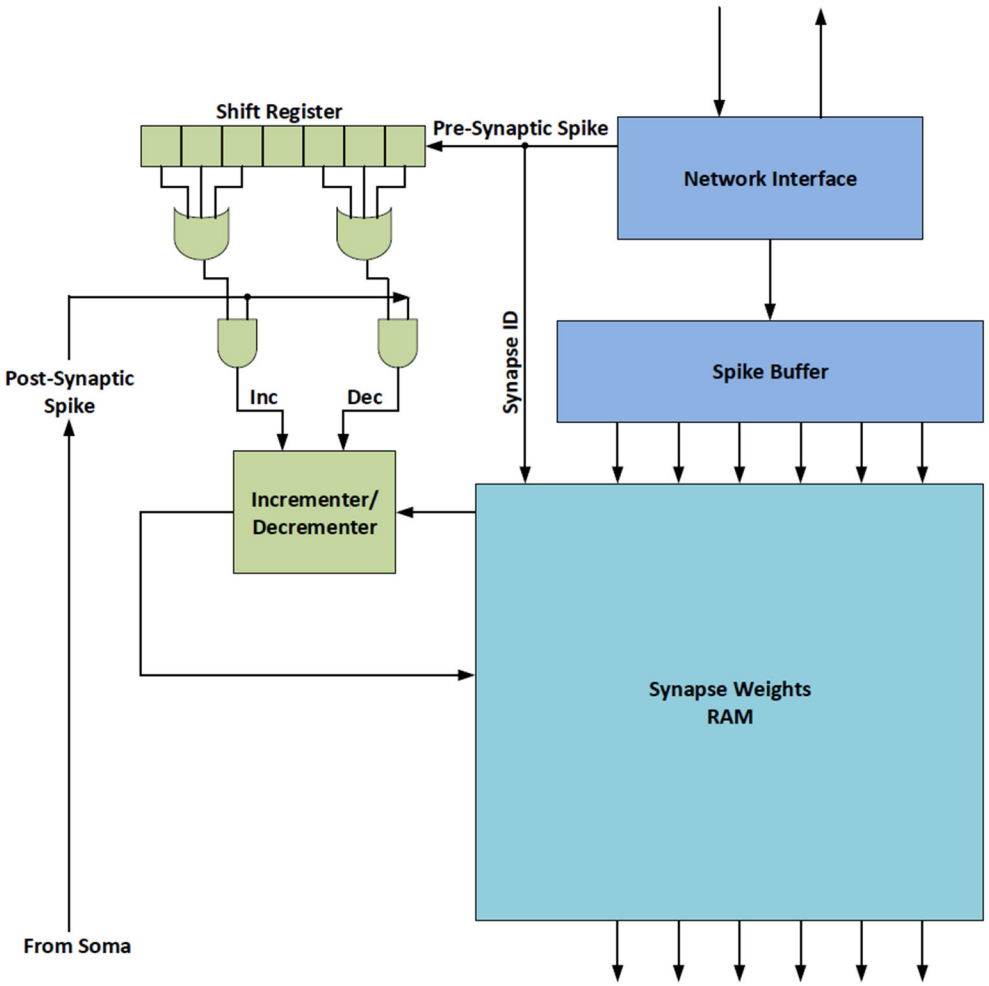


FIGURE 12 | A block diagram of an example hardware implementation of Spike-Time Dependent Plasticity (STDP) that was inspired by Cassidy et al. (2013a). This hardware could allow BrainFreeze to support on-line learning.

learning seeks to increase the synaptic weight of a synapse if the pre-synaptic neuron tends to influence the firing of the post-synaptic neuron. In STDP this influence is determined by considering the timing of the firing of pre and post synaptic neurons. If the pre-synaptic neuron tends to fire before the post-synaptic neuron then the weight of the synapse between them is increased. Conversely, if an action potential from the pre-synaptic neuron does not typically result in an output from the post-synaptic neuron then the weight of the synapse between them should be decreased.

In order to incorporate on-line learning into a large scale neuromorphic architecture it is critical that the hardware required to support the learning functionality is itself scalable. This can be achieved by keeping the hardware simple and by utilizing time multiplexing to share the hardware between neurons in the simulated neural network. It is possible to efficiently integrate STDP into BrainFreeze without disrupting the functionality of the system because BrainFreeze already makes use of time-multiplexing and the required hardware additions are not individually complex. In particular, the functionality of the control component of the neuron core will need to be expanded to make decisions regarding the synapse weight updates. The control will need to determine if an update should occur as well as the degree and type of update. Some small memories will also likely be needed to store information about the learning rules that should be applied. **Figure 12** depicts a potential implementation of on-line learning support hardware that is compatible with the BrainFreeze architecture. In this implementation, a shift register is used to establish the timing relationship between the pre and post synaptic spikes. If the pre-synaptic spike occurred before the post-synaptic spike then the weight of the corresponding synapse is increased. Alternatively, if the pre-synaptic spike occurs after the post-synaptic spike then the weight of the corresponding synapse is decreased. This particular implementation is included to provide a straightforward example of STDP hardware, other implementations are possible and should be explored by future work.

Beyond the changes to the control hardware on the neuron core, supporting STDP also requires additional local memory resources if a multi-temperature memory system is used. This is because the STDP hardware will need to update the weights for previous neurons before those weights can be written back to the backing store. As a result, the local buffers will need to be large enough to store the synapse weights for three or more virtual neurons. In systems that do not include STDP, the local buffers would only need to store the synapse weights for the current neuron and the weights for the next neuron that are being loaded from the backing store. In systems that do not include a multi-temperature memory system, no additional local memory is needed because all of the synapse weights for all of the simulated neurons already need to be present in the memory of the neuron core. Also, in systems that include STDP, the local buffers will need additional ports or additional banks so that a weight update can occur while other synapses are being accessed. However, these buffers will be reused by different simulated neurons due to the time-multiplexed nature of the BrainFreeze architecture so they will not greatly impact the scalability of the system. Synapse weights will require some time to be updated due to communication and access delays. In a time-multiplexed system, this update latency should be hidden since the synapses being updated will not be needed again for many clock cycles. Overall, supporting on-line learning will require additional hardware in the system but the additions should not significantly impact the scalability of the architecture.

12. LARGE SYSTEM SCALING

Building upon the capabilities and configurations of BrainFreeze that have been explored so far, we can roughly estimate how well the architecture would scale for very large simulations. One of the ultimate tests of scalability for neuromorphic architectures is supporting a simulation that involves roughly 80 billion neurons. This is the approximate number of neurons contained in the human brain (Furber, 2016). In order to evaluate the scalability of BrainFreeze, we compare it to other approaches in terms of the number of standard server cabinets of equipment that would be needed to support a simulation of this size. For the purposes of this comparison we ignore the cabinets of peripheral equipment needed by the various approaches and focus only on the cabinets that contain the neuromorphic chips themselves. Many neuromorphic approaches require additional hardware to operate. Additional hardware will also likely be required to support the superconducting approach but the quantity is currently unknown. The neuromorphic chips for the BrainFreeze system studied here include the neuron core, the synapse weight memories, the post-synaptic address memories, and the routers. In addition, we include 25 cabinets that we estimate will be needed to host the roughly 1 petabyte of Cryo-DRAM that will be needed if a backing store is included in the architecture.

The scalability of superconducting neuromorphic systems like BrainFreeze is supported by several important characteristics of SCE. First, the low heat dissipation of SCE means that chips and boards can be packed much closer together than is possible in CMOS. This results in better density per unit volume even though SCE is typically less dense than CMOS in terms of area. Second, the efficiency of long distance communications in SCE means that the long wires needed to build very large systems require much less energy to operate. Third, the overall energy efficiency of SCE means that the neuron cores themselves require much less energy to operate than many CMOS alternatives. As a result of these two characteristics, much larger SCE neuromorphic systems can be built before power requirements become an obstacle. Finally, the speed of SCE means that a much higher degree of multiplexing can be employed before the run-time of the neural network simulation becomes prohibitive. This means that SCE can support more simulated neurons with less hardware. These factors combined mean that approaches like BrainFreeze are uniquely suited to scale to very large systems due to their efficient use of space, energy, and hardware.

For the superconducting system, the anticipated chip size greatly affects the amount of hardware that is required. We assume a chip size roughly equivalent to an NVidia V100 (28.5 x 28.5 mm). Chips of this size can support roughly 3 MB of a JMRAM like memory or 1,000–2,000 neurons per die. We anticipate arranging the chips on a 200 x 240 mm interposer that can accommodate 49 chips. Alternatively smaller interposers could possibly be used and connected together to build the system with a slightly higher volume overhead. Due to the limited heat generated by superconducting electronics, the boards can be placed into the racks with only 7.3 mm of space between each board.

In **Figure 13** we compare BrainFreeze to SpiNNaker. SpiNNaker was chosen for this comparison because it is the state-of-the-art neuromorphic approach that has been scaled to the largest number of neurons (Furber, 2016; Yang and Kim, 2020). It is worth noting that the 866 cabinets that are projected for SpiNNaker are an extrapolation of the available data. From the results in **Figure 13**, we can see that the number of standard server cabinets needed to house 80 billion neurons is roughly 700–2,500 if no backing store is used and 200–1,600 if the backing store is included. This illustrates the difficulty of scaling the architecture to extremely large numbers of neurons using only superconducting memories. However, with the support of a backing store it is possible to implement 80 billion neurons using roughly a quarter of the cabinets that would be required by the SpiNNaker approach. More importantly, 200 cabinets is roughly the size of a modern supercomputer. Therefore, BrainFreeze has the potential to enable extremely large neural network simulations that involve biologically relevant dynamics while requiring a volume that is suitable for a typical datacenter.

13. DISCUSSION

The analysis presented in this paper shows that feasible configurations exist for the BrainFreeze architecture that would enable it to be competitive with other state-of-the-art large scale neuromorphic designs. Local superconducting memory capacity is shown to be sufficient for small scale demonstrations and for acting as a local buffer for large systems. A multi-temperature

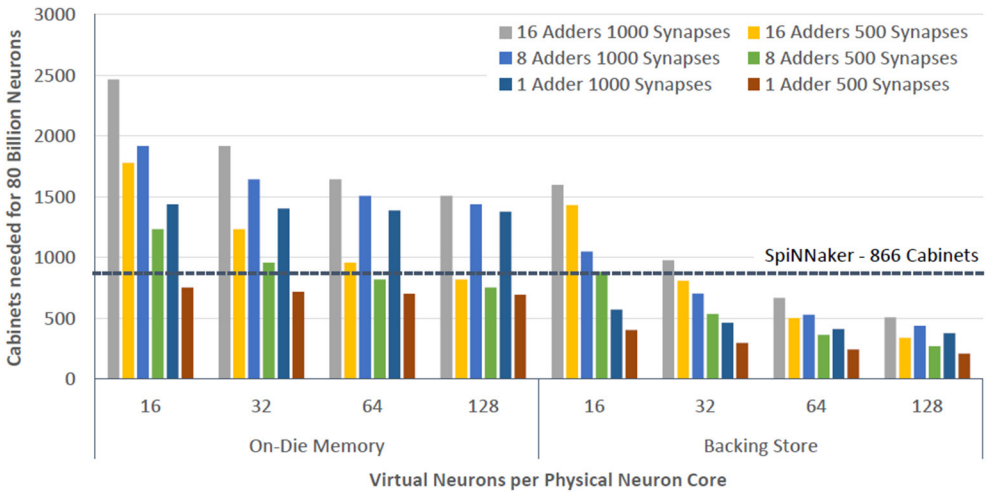


FIGURE 13 | The number of standard rack sized cabinets that would be required to house an 80 billion neuron BrainFreeze system for different degrees of multiplexing. Provided that sufficient multiplexing is employed, BrainFreeze could potentially implement such a system in considerably less volume than other state-of-the-art neuromorphic approaches.

memory hierarchy combined with prefetching and the local buffers is shown to be capable of providing enough memory capacity for even large scale systems. This memory capacity and the high clock rates enabled by SCE can then be utilized to enable multiplexing to improve neuron densities per chip. Finally, the low power dissipation and subsequent low heat generation of SCE allows for more tightly packed systems ultimately resulting in a density per volume that is shown to be potentially superior to existing approaches. Taken together these results describe a system that can simultaneously provide improved performance, energy efficiency, and scalability while supporting biologically suggestive neuron models and on-line learning.

The analysis and results presented in this paper also provide motivation for future work in this area. Research is needed to continue the development of the various digital and analog components that are required to build a mixed-signal SCE neuromorphic system like BrainFreeze. For example, enabling greater configurability in the analog soma circuits could introduce new functionality to the system and expand the research that it could support. In addition, while the parameters used in this work are based on experimental demonstrations, they are still just approximations of the latency and area needed for an actual implementation of BrainFreeze. In order to validate and refine the findings of this analysis, a complete BrainFreeze core should be built in hardware. In particular, the control scheme of the BrainFreeze system needs to be carefully designed in order to preserve the important aspects of the biologically suggestive soma model despite the discretized digital communication of action potentials.

Developing large scale SCE neuromorphic systems could provide a pivotal experimental apparatus to both the machine learning and computational neuroscience communities. Enabling high performance biologically suggestive simulations could support the development of new applications and may help in the development of general AI. A large scale SCE system could be deployed as a cloud appliance thereby allowing researchers from various fields access to it. This would help to distribute the costs of developing and maintaining the system while ensuring the broadest impact of its unique collection of capabilities.

14. CONCLUSIONS

This work has endeavored to explain how a programmable, large scale SCE neuromorphic system could be built using a mixed-signal architecture. The feasibility of the proposed BrainFreeze architecture was supported by numerical analysis and trade studies based on measurements from experimental demonstrations. The results showed that it should be possible to build a BrainFreeze system that simultaneously provides programmability, scalability, speed, energy efficiency, biological suggestivity, and on-line learning support. Such a system could prove to be a critical resource supporting the development of novel machine learning applications, supporting computational neuroscience experiments, and perhaps one day supporting the drive to artificial general intelligence.

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

KS wrote Section 2.3, Section 4.2, and provided revisions to the other sections. PT wrote the sections of the paper

not written by KS and performed the numerical analysis except for the analysis presented in Section 4.2. Both authors contributed to the article and approved the submitted version.

FUNDING

This research was funded as an internal research and development effort of Northrop Grumman.

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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI: IEEE), 7243–7252.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., et al. (2016). "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning* (New York, NY: PMLR), 173–182.

Amodei, D., Hernandez, D., Sastry, G., Clark, J., Brockman, G., and Sutskever, I. (2018). *Ai and Compute*. Available online: https://blog.openai.com/aiand-compute.

Aradhya, S. V., Rowlands, G. E., Oh, J., Ralph, D. C., and Buhrman, R. A. (2016). Nanosecond-timescale low energy switching of in-plane magnetic tunnel junctions through dynamic oersted-field-assisted spin hall effect. *Nano Lett*. 16, 5987–5992. doi: 10.1021/acs.nanolett.6b01443

Arthur, J. V., and Boahen, K. A. (2011). Silicon-neuron design: a dynamical systems approach. *IEEE Trans. Circ. Syst. I.* 58, 1034–1043. doi: 10.1109/TCSI.2010.2089556

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Boahen, K. (2000). Point-to-point connectivity between neuromorphic chips using address events. *IEEE Tran. Circ. Sys. II* 47:416. doi: 10.1109/82.842110

Borkar, S. (2011). "The exascale challenge. keynote presentation at PACT," in *IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)* (Galveston, TX).

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., McCaughan, A. N., Chiles, J., Mirin, R. P., Nam, S. W., Shainline, J. M., et al. (2018). "Design of superconducting optoelectronic networks for neuromorphic computing," in *2018 IEEE International Conference on Rebooting Computing (ICRC)* (McLean, VA: IEEE), 1–7.

Buckley, S. M., Chiles, J., McCaughan, A. N., Mirin, R. P., Nam, S. W., and Shainline, J. M. (2017). "Photonic interconnect with superconducting electronics for large-scale neuromorphic computing (invited paper)," in *2017 IEEE Photonics Society Summer Topical Meeting Series (SUM)* (San Juan, PR). 51–52.

Burnett, R., Clarke, R., Lee, T., Hearne, H., Vogel, J., Herr, Q., et al. (2018). "Demonstration of superconducting memory for an rql cpu," in *Proceedings of the International Symposium on Memory Systems, MEMSYS '18* (New York, NY: ACM), 321–323.

Cassidy, A. S., Georgiou, J., and Andreou, A. G. (2013a). Design of silicon brains in the nano-cmos era: spiking neurons, learning synapses and neural architecture optimization. *Neural Netw*. 45:4–26. doi: 10.1016/j.neunet.2013.05.011

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. (2013b). "Cognitive computing building block: a versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX: IEEE), 1–10.

Chiles, J., Buckley, S. M., Nam, S. W., Mirin, R. P., and Shainline, J. M. (2018). Design, fabrication, and metrology of 10 100 multi-planar integrated photonic routing manifolds for neural networks. *APL Photonics* 3, 106101. doi: 10.1063/1.5039641

Chollet, F. (2017). "Xception: deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI: IEEE), 1251–1258.

Crotty, P., Schult, D., and Segall, K. (2010). Josephson junction simulation of neurons. *Phys. Rev. E* 82:011914. doi: 10.1103/PhysRevE.82.011914

Davies, M., Srinivasa, N., Lin, T., 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

Dayton, I., Baker, H., Loving, M., Ambrose, T., Siwak, N., Keebaugh, S., et al. (2018). "Demonstration of josephson magnetic random access memory (jmram)," in *Applied Superconductivity Conference* (Seattle, WA).

Dayton, I. M., Sage, T., Gingrich, E. C., Loving, M. G., Ambrose, T. F., Siwak, N. P., et al. (2018). Experimental demonstration of a josephson magnetic memory cell with a programmable π-junction. *IEEE Magn. Lett*. 9, 1–5. doi: 10.1109/LMAG.2018.2801820

Ebong, I. E., and Mazumder, P. (2012). Cmos and memristor-based neural network design for position detection. *Proc. IEEE* 100, 2050–2060. doi: 10.1109/JPROC.2011.2173089

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

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

Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2013). Overview of the spinnaker system architecture. *IEEE Trans. Comput*. 62, 2454–2467. doi: 10.1109/TC.2012.142

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: IEEE), 770–778.

Hearne, H., Vogel, J., Clarke, R., Burnett, R., Lee, T., Herr, A., et al. (2018). "Rql encoded 8x16 register file for 16-bit cpu," in *Applied Superconductivity Conference* (Seattle, WA).

Herr, Q. P., Herr, A. Y., Oberg, O. T., and Ioannidis, A. G. (2011). Ultra-low-power superconductor logic. *J. Appl. Phys*. 109, 103903. doi: 10.1063/1.3585849

Hidaka, M., and Akers, L. A. (1991). An artificial neural cell implemented with superconducting circuits. *Supercond. Sci. Technol*. 4, 654–657. doi: 10.1088/0953-2048/4/11/027

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hodgkin, A., and Huxley, A. (1990). A quantitative description of membrane current and its application to conduction and excitation in nerve (reprinted from journal of physiology, vol 117, pg 500-544, 1952). *Bull. Math. Biol.* 52, 25–71. doi: 10.1016/S0092-8240(05)80004-7

Inoue, K., Takeuchi, N., Ehara, K., Yamanashi, Y., and Yoshikawa, N. (2013). Simulation and experimental demonstration of logic circuits using an ultra-low-power adiabatic quantum-flux-parametron. *IEEE Trans. Appl. Supercond*. 23, 1301105–1301105. doi: 10.1109/TASC.2012.2236133

Inoue, K., Takeuchi, N., Narama, T., Yamanashi, Y., and Yoshikawa, N. (2015). Design and demonstration of adiabatic quantum-flux-parametron logic circuits with superconductor magnetic shields. *Supercond. Sci. Technol*. 28, 045020. doi: 10.1088/0953-2048/28/4/045020

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw*. 15, 1063–1070. doi: 10.1109/TNN.2004.832719

Kelly, T., Fernandez, P., Vogelsang, T., McKee, S. A., Gopalakrishnan, L., Magee, S., et al. (2019). Some like it cold: Initial testing results for cryogenic computing components. *J. Phys*. 1182:012004. doi: 10.1088/1742-6596/1182/1/012004

Kirichenko, A. F., Sarwana, S., and Vernik, I. V. (2012). "Ersfq-zero static power dissipation single flux quantum logic," in *Government Microcircuit Applications and Critical Technology Conference (GOMACTech-12)* (Las Vegas, NV), 319–322.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*. 60, 84–90. doi: 10.1145/3065386

Lin, C.-K., Wild, A., Chinya, G. N., Lin, T.-H., Davies, M., and Wang, H. (2018). "Mapping spiking neural networks onto a manycore neuromorphic architecture," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018* (New York, NY: ACM), 78–89.

Mahowald, M. (1992). *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*. Pasadena, CA: California Institute of Technology, (Unpublished). Available online at: https://resolver.caltech.edu/CaltechCSTR:1992.cs-tr-92-15

Makhlooghpour, A., Soleimani, H., Ahmadi, A., Zwolinski, M., and Saif, M. (2016). "High accuracy implementation of adaptive exponential integrated and fire neuron model," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 192–197.

Manheimer, M. A. (2015). Cryogenic computing complexity program: Phase 1 introduction. *IEEE Trans. Appl. Supercond.* 25, 1–4. doi: 10.1109/TASC.2015.2399866

Meier, K. (2015). "A mixed-signal universal neuromorphic computing system," in *2015 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC), 4.6.1–4.6.4.

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

Mizugaki, Y., Nakajima, K., Sawada, Y., and Yamashita, T. (1993). Superconducting neural circuits using fluxon pulses. *Appl. Phys. Lett.* 62, 762–764. doi: 10.1063/1.108571

Mizugaki, Y., Nakajima, K., Sawada, Y., and Yamashita, T. (1994). Implementation of new superconducting neural circuits using coupled squids. *IEEE Trans. Appl. Supercond.* 4, 1–8. doi: 10.1109/77.273058

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nagumo, J., Arimoto, S., and Yoshizawa, S. (1962). Impulses and physiological states in models of nerve membrane. *Proc. Inst. Radio Engrs* 50, 2061–2070. doi: 10.1109/JRPROC.1962.288235

Narama, T., Yamanashi, Y., Takeuchi, N., Ortlepp, T., and Yoshikawa, N. (2015). "Demonstration of 10k gate-scale adiabatic-quantum-flux-parametron circuits," in *2015 15th International Superconductive Electronics Conference (ISEC)* (Nagoya), 1–3.

Park, J., Yu, T., Joshi, S., Maier, C., and Cauwenberghs, G. (2017). Hierarchical address event routing for reconfigurable large-scale neuromorphic systems. *IEEE Trans. Neural Netw. Learn. Syst.* 28:2408. doi: 10.1109/TNNLS.2016.2572164

Qian, G., Cahay, M., and Kothari, R. (1995). A new superconducting neural cell. *Superlattices Microstruct.* 18, 259. doi: 10.1006/spmi.1995.1110

Rast, A. D., Shufan, Y.ang, Khan, M., and Furber, S. B. (2008). "Virtual synaptic interconnect using an asynchronous network-on-chip," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (Hong Kong: IEEE), 2727–2734.

Rippert, E., and Lomatch, S. (1997). A multilayered superconducting neural network implementation. *IEEE Trans. Appil. Supercond.* 7, 3442–3445. doi: 10.1109/77.622126

Rose, R., and Hindmarsh, J. (1989). The assembly of ionic currents in a thalamic neuron i. the three-dimensional model. *Proc. R. Soc. Lond. B Biol. Sci.* 237, 267–288. doi: 10.1098/rspb.1989.0049

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), 1947–1950.

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. (2018b). Ultralow power artificial synapses using nanotextured magnetic josephson junctions. *Sci. Adv.* 4, e1701329. doi: 10.1126/sciadv.1701329

Sebastian, A., Le Gallo, M., Burr, G. W., Kim, S., BrightSky, M., and Eleftheriou, E. (2018). Tutorial: Brain-inspired computing using phase-change memory devices. *J. Appl. Phys.* 124, 111101. doi: 10.1063/1.5042413

Shainline, J. M. (2020). Fluxonic processing of photonic synapse events. *IEEE J. Select. Top. Quantum Electron.* 26, 1–15. doi: 10.1109/JSTQE.2019.2927473

Shainline, J. M., Buckley, S. M., McCaughan, A. N., Chiles, J. T., Salim, A. J., 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. (2016). "Neuromorphic computing with integrated photonics and superconductors," in *2016 IEEE International Conference on Rebooting Computing (ICRC)* (San Diego, CA), 1–8.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017b). Mastering the game of go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sompolinsky, H. (2014). Computational neuroscience: beyond the local circuit. *Curr. Opin. Neurobiol.* 25, xiii–xviii. doi: 10.1016/j.conb.2014.02.002

Soudry, D., Di Castro, D., Gal, A., Kolodny, A., and Kvatinsky, S. (2015). Memristor-based multilayer neural networks with online gradient descent training. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2408–2421. doi: 10.1109/TNNLS.2014.2383395

Stromatias, E., Neil, D., Galluppi, F., Pfeiffer, M., Liu, S.-C., and Furber, S. (2015). "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA: IEEE), 1–9.

Tannu, S. S., Carmean, D. M., and Qureshi, M. K. (2017). "Cryogenic-dram based memory system for scalable quantum computers: a feasibility study," in *Proceedings of the International Symposium on Memory Systems, MEMSYS '17* (New York, NY: ACM), 189–195.

Vesely, M., Tschirhart, P., Clarke, R., Huertas-Morales, Y., Lateef, M., Rahman, S., et al. (2018). "An 8-bit and 16-bit alu for superonducting reciprocal quantum logic (rql) cpus," in *Applied Superconductivity Conference* (Seattle, WA).

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575, 350–354. doi: 10.1038/s41586-019-1724-z

Wang, F., Vogelsang, T., Haukness, B., and Magee, S. C. (2018). "Dram retention at cryogenic temperatures," in *2018 IEEE International Memory Workshop (IMW)* (Kyoto: IEEE), 1–4.

Ware, F., Gopalakrishnan, L., Linstadt, E., McKee, S. A., Vogelsang, T., Wright, K. L., et al. (2017). "Do superconducting processors really need cryogenic memories?: the case for cold dram," in *Proceedings of the International Symposium on Memory Systems, MEMSYS '17* (New York, NY: ACM), 183–188.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yang, Y. S., and Kim, Y. (2020). "Recent trend of neuromorphic computing hardware: Intel's neuromorphic system perspective," in *2020 International SoC Design Conference (ISOCC)* (Yeosu: IEEE), 218–219.

Ye, L., Gopman, D., Rehm, L., Backes, D., Wolf, G., Ohki, T., et al. (2014). Spin-transfer switching of orthogonal spin-valve devices at cryogenic temperatures. *J. Appl. Phys*. 115, 17C725. doi: 10.1063/1.4865464

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

Zeiler, M. D., and Fergus, R. (2014). "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision* (Zurich: Springer), 818–833.

Zoph, B., and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Conflict of Interest: PT was employed by Northrop Grumman while performing this research. PT is also listed as the inventor of patent US11157804B2 - Superconducting neuromorphic core, which is related to some of the ideas discussed in this work.

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.

frontiers | Frontiers in Neuroinformatics

TYPE Technology and Code
PUBLISHED 09 November 2022
DOI 10.3389/fninf.2022.1015624

OPEN ACCESS

EDITED BY
Andrew P. Davison,
UMR9197 Institut des Neurosciences Paris Saclay (Neuro-PSI), France

REVIEWED BY
Yao-Feng Chang,
Intel, United States
Terrence C. Stewart,
National Research Council Canada (NRC), Canada

*CORRESPONDENCE
Carlo Michaelis
carlo.michaelis@phys.uni-goettingen.de

†These authors have contributed equally to this work

RECEIVED 09 August 2022
ACCEPTED 12 October 2022
PUBLISHED 09 November 2022

CITATION
Michaelis C, Lehr AB, Oed W and Tetzlaff C (2022) Brian2Loihi: An emulator for the neuromorphic chip Loihi using the spiking neural network simulator Brian.
Front. Neuroinform. 16:1015624.
doi: 10.3389/fninf.2022.1015624

Brian2Loihi: An emulator for the neuromorphic chip Loihi using the spiking neural network simulator Brian

Carlo Michaelis[1,2*†], Andrew B. Lehr[1,2†], Winfried Oed[1,2†] and Christian Tetzlaff[1,2]

[1]Department of Computational Neuroscience, University of Göttingen, Göttingen, Germany, [2]Bernstein Center for Computational Neuroscience, University of Göttingen, Göttingen, Germany

Developing intelligent neuromorphic solutions remains a challenging endeavor. It requires a solid conceptual understanding of the hardware's fundamental building blocks. Beyond this, accessible and user-friendly prototyping is crucial to speed up the design pipeline. We developed an open source Loihi emulator based on the neural network simulator Brian that can easily be incorporated into existing simulation workflows. We demonstrate errorless Loihi emulation in software for a single neuron and for a recurrently connected spiking neural network. On-chip learning is also reviewed and implemented, with reasonable discrepancy due to stochastic rounding. This work provides a coherent presentation of Loihi's computational unit and introduces a new, easy-to-use Loihi prototyping package with the aim to help streamline conceptualization and deployment of new algorithms.

KEYWORDS

neuromorphic computing, Loihi, Brian2, emulator, spiking neural network, open source

1. Introduction

Neuromorphic computing offers exciting new computational structures. Decentralized units inspired by neurons are implemented in hardware (reviewed by Schuman et al., 2017; Rajendran et al., 2019; Young et al., 2019). These can be connected up to one another, stimulated with inputs, and the resulting activity patterns can be read out from the chip as output. A variety of algorithms and applications have been developed in recent years, including robotic control (DeWolf et al., 2016, 2020; Michaelis et al., 2020; Stagsted et al., 2020), spiking variants of deep learning algorithms, attractor networks, nearest-neighbor or graph search algorithms (reviewed by Davies et al., 2021). Moreover, neuromorphic hardware may provide a suitable substrate for performing large scale simulations of the brain (Furber, 2016; Thakur et al., 2018). Neuromorphic chips specialized for particular computational tasks can either be provided as a neuromorphic computing cluster or be integrated into existing systems, akin to graphics processing units (GPU) in modern computers (Furber et al., 2014; Davies et al., 2021). With the right ideas, networks of spiking units implemented in

neuromorphic hardware can provide the basis for powerful and efficient computation. Nevertheless, the development of new algorithms for spiking neural networks, applicable to neuromorphic hardware, is a challenge (Grüning and Bohte, 2014; Pfeiffer and Pfeil, 2018; Bouvier et al., 2019).

At this point, without much background knowledge of neuromorphic hardware, one can get started programming using the various software development kits available (e.g., Brüderle et al., 2011; Sawada et al., 2016; Lin et al., 2018; Rhodes et al., 2018; Michaelis, 2020; Müller et al., 2020a,b; Spilger et al., 2020; Rueckauer et al., 2021). Emulators for neuromorphic hardware (Furber et al., 2014; Petrovici et al., 2014; Luo et al., 2018; Valancius et al., 2020) running on a standard computer or field programmable gate arrays (FPGA), make it possible to develop neuromorphic network architectures without even needing access to a neuromorphic chip (see e.g., NengoLoihi[1] and Dynap-SE[2]). This can speed up prototyping as the initialization of networks, i.e., distributing neurons and synapses, as well as the readout of the system's state variables on neuromorphic chips takes some time. At the same time emulators transparently contain the main functionalities of the hardware in code and therefore provide insights into how it works. With this understanding, algorithms can be intelligently designed and complex network structures implemented.

In the following, we introduce an emulator for the digital neuromorphic chip `Loihi` (Davies et al., 2018) based on the widely used spiking neural network simulator `Brian` (Stimberg et al., 2019). We first dissect an individual computational unit from `Loihi`. The basic building block is a spiking unit inspired by a current based leaky integrate and fire (LIF) neuron model (see Gerstner et al., 2014). Connections between these units can be plastic, enabling the implementation of diverse on-chip learning rules. Analyzing the computational unit allows us to create an exact emulation of the `Loihi` hardware on the computer. We extend this to a spiking neural network model and demonstrate that both `Loihi` and `Brian` implementations match perfectly. This exact match means one can do prototyping directly on the computer using `Brian` only, which adds another emulator in addition to the existing simulation backend in the Nengo Loihi library. This increases both availability and simplicity of algorithm design for `Loihi`, especially for those who are already used to working with `Brian`. In particular for the computational neuroscience community, this facilitates the translation of neuroscientific models to neuromorphic hardware. Finally, we review and implement synaptic plasticity and show that while individual weights show small deviations due to stochastic rounding, the statistics of a learning rule are preserved. Our aim is to facilitate the development of neuromorphic algorithms by delivering an open source emulator package that can easily be incorporated into existing workflows. In the process we provide a solid understanding of what the hardware computes, laying the appropriate foundation to design precise algorithms from the ground up.

2. Loihi's computational unit and its implementation

Developing a `Loihi` emulator requires precise understanding of how `Loihi` works. And to understand how something works, it is useful to "take it apart and put it back together again". While we will not physically take the `Loihi` chip apart, we can inspect the components of its computational units with "pen and paper". Then, by implementing each component on a computer we will test that, when put back together, the parts act like we expect them to. In the following we highlight how spiking units on `Loihi` approximate a variant of the well-known LIF model using first order Euler numerical integration with integer precision. This understanding enables us to emulate `Loihi`'s spiking units on the computer in a way that is straightforward to use and easy to understand. For a better intuition of how the various parameters on `Loihi` interact, we refer readers to our neuron design tool[3] for `Loihi`. Readers familiar with Davies et al. (2018) and numerical implementations of LIF neurons may prefer to skip to Section 2.3.

2.1. `Loihi`'s neuron model: A recap

The basic computational unit on `Loihi` is inspired by a spiking neuron (Davies et al., 2018). `Loihi` uses a variant of the leaky integrate and fire neuron model (Gerstner et al., 2014) (see Appendix 9.1). Each unit i of `Loihi` implements the dynamics of the voltage v_i

$$\frac{dv_i}{dt} = -\frac{1}{\tau_v} v_i(t) + I_i(t) - v_i^{th}\sigma_i(t), \tag{1}$$

where the first term controls the voltage decay, the second term is the input to the unit, and the third term resets the voltage to zero after a spike by subtracting the threshold. A spike is generated if $v_i > v_i^{th}$ and transmitted to other units to which unit i is connected. In particular, v models the voltage across the membrane of a neuron, τ_v is the time constant for the voltage decay, I is an input variable, v^{th} is the threshold voltage to spike, and $\sigma(t)$ is the so-called spike train which is meant to indicate whether the unit spiked at time t. For each unit i, $\sigma_i(t)$ can be

1 https://www.nengo.ai/nengo-loihi/

2 https://code.ini.uzh.ch/yigit/NICE-workshop-2021

3 https://github.com/andrewlehr/loihi_parameter_tuning_dashboard

written as a sum of Dirac delta distributions

$$\sigma_i(t) = \sum_k \delta(t - t_{i,k}), \tag{2}$$

where $t_{i,k}$ denotes the time of the k-th spike of unit i. Note that σ_i is not a function, but instead defines a *distribution* (i.e., *generalized function*), and is only meaningful under an integral sign. It is to be understood as the linear functional $\langle \sigma_i, f \rangle := \int \sigma_i(t) f(t)\, dt = \sum_k f(t_{i,k})$ for arbitrary, everywhere-defined function f (see Corollary 1 in Appendix 9.1.2).

Input to a unit can come from user defined external stimulation or from other units implemented on chip. Davies et al. (2018) describe the behavior of the input $I(t)$ with

$$I_i(t) = \sum_j J_{ij}(\alpha_I * \sigma_j)(t) + I_i^{\text{bias}}, \tag{3}$$

where J_{ij} is the weight from unit j to i, I_i^{bias} is a constant bias input, and the spike train σ_j of unit j is convolved with the synaptic filter impulse response α_I, given by

$$\alpha_I(t) = \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{4}$$

where τ_I is the time constant of the synaptic response and $H(t)$ the unit step function. Note that $\alpha_I(t)$ is defined differently here than in Davies et al. (2018) (see Appendix 9.1.3 for details). The convolution from Equation (3) is a notational convenience for defining the synaptic input induced by an incoming spike train, simply summing over the time-shifted synaptic response functions, namely $(\sigma_i * f)(t) = \langle \sigma_i, \tau_t \tilde{f} \rangle = \sum_k f(t - t_{i,k})$, where $\tau_t f(x) = f(x - t)$ and $\tilde{f}(x) = f(-x)$ (see Appendix 9.1.2).

2.2. Implementing `Loihi`'s spiking unit in software

From the theoretical model on which `Loihi` is based, we can derive the set of operations each unit implements with a few simple steps. Using a first order approximation for the differential equations gives the update equations for the voltage and synaptic input described in the `Loihi` documentation.[4] Combined with a few other details regarding `Loihi`'s integer precision and the order of operations, we will have all we need to implement a `Loihi` spiking unit in software.

2.2.1. Synaptic input

From Equation (3), we see that the synaptic input can be written as a sum of exponentially decaying functions with amplitude J_{ij} beginning at the time of each spike $t_{j,k}$ (see Appendix 9.1.2). In particular we have

$$I_i(t) = \sum_j J_{ij} \sum_k \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}) + I_i^{\text{bias}}. \tag{5}$$

To understand the behavior of the synaptic input it is helpful to consider the effect of one spike arriving at a single synapse. Simplifying Equation (5) to just one neuron that receives just one input spike at time $t_1 = 0$, for $t \geq 0$ we get

$$I(t) = J \cdot \exp\left(-\frac{t}{\tau_I}\right) \tag{6}$$

and for $t < 0$, $I(t) = 0$. Each spike induces a step increase in the current which decays exponentially with time constant τ_I. Taking the derivative of both sides with respect to t gives

$$\frac{dI}{dt} = -\frac{1}{\tau_I} \cdot I(t), \tag{7}$$

$$I(0) = J. \tag{8}$$

Applying the forward Euler method to the differential equation for $\Delta t = 1$ and $t \geq 0$, $t \in \mathbb{N}$ we get

$$I[t] = I[t-1] - \frac{1}{\tau_I} \cdot I[t-1] + J \cdot s[t], \tag{9}$$

where $s[t]$ is zero unless there is an incoming spike on the synapse, in which case it is one. Here, $s[0] = 1$ and $s[t] = 0$ for $t > 0$. With this we have simply incorporated the initial condition into the update equation. Note that we have switched from a continuous [e.g., $I(t)$] to discrete (e.g., $I[t]$) time formulation, where $\Delta t = 1$ and t is unitless.

`Loihi` has a decay value δ^I, which is inversely proportional to τ_I, namely $\delta^I = 2^{12}/\tau_I$. Swapping τ_I by δ^I reveals

$$I[t] = I[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + J \cdot s[t]. \tag{10}$$

The weight J is defined *via* the mantissa $\tilde{w}_{ij}$ and exponent Θ (see Section 3.1) such that the equation describing the synaptic input becomes (with indices)

$$I_i[t] = I_i[t-1] \cdot (2^{12} - \delta^I) \cdot 2^{-12} + 2^{6+\Theta} \cdot \sum_j \left(\tilde{w}_{ij} \cdot s_j[t]\right), \tag{11}$$

where $s_j[t] \in \{0, 1\}$ is the spike state of the j^{th} input neuron. Please note that Equation (2.2.1) is identical to the `Loihi` documentation.

From this we can conclude that the implementation of synaptic input on Loihi is equivalent to evolving the LIF synaptic input differential equation with the forward Euler numerical integration method (see Figure 1A1).

4 The documentation for the `NxSDK` is available from Intel on request.

2.2.2. Voltage

It is straightforward to perform the same analysis as above for the voltage equation. We consider the subthreshold voltage dynamics for a single neuron and can therefore ignore the reset term $v_i^{th}\sigma_i(t)$ from Equation (1), leaving us with

$$\frac{dv}{dt} = -\frac{1}{\tau_v}v(t) + I(t). \tag{12}$$

Applying forward Euler gives

$$v[t] = v[t-1] - \frac{v[t-1]}{\tau_v} + I[t]. \tag{13}$$

Again, to compare with the `Loihi` documentation we need to swap the time constant τ_v by a voltage decay parameter, δ^v, which is inversely proportional to the time constant, the same as above for synaptic input. Plugging in $\tau_v = 2^{12}/\delta^v$ leads to

$$v[t] = v[t-1] \cdot (2^{12} - \delta^u) \cdot 2^{-12} + I[t]. \tag{14}$$

By introducing a bias term, the voltage update becomes

$$v_i[t] = v_i[t-1] \cdot (2^{12} - \delta^u) \cdot 2^{-12} + I_i[t] + I_i^{\text{bias}}. \tag{15}$$

Equation (15) agrees with the `Loihi` documentation. Like the synaptic input, the voltage implementation on `Loihi` is equivalent to updating the LIF voltage differential equation using forward Euler numerical integration (see Figure 1A2).

2.2.3. Integer precision

`Loihi` uses integer precision. So the mathematical operations in the update equations above are to be understood in terms of integer arithmetic. In particular, for the synaptic input and voltage equations the emulator uses *round away from zero*, which can be defined as

$$x_{\text{round}} \coloneqq \text{sign}(x) \cdot \lceil |x| \rceil. \tag{16}$$

where $\lceil \cdot \rceil$ is the ceiling function and sign$(\cdot)$ the sign function.

2.3. Summary

We now have all of the pieces required to understand and emulate a spiking unit from `Loihi`. Evolving the differential equations for the current-based LIF model with the forward Euler method and using the appropriate rounding (see Section 2.2.3) and update schedule (see Section 4.1 and Appendix 9.2.1) is enough to exactly reproduce `Loihi`'s behavior. This procedure is summarized in Algorithm 1 and an exact match between `Loihi` and an implementation for a single unit in `Brian` is shown in Figure 1A. Please note that during the refractory period `Loihi` uses the voltage trace to count elapsed time (see Figure 1A2, Appendix 9.2.2), while in the emulator the voltage is simply clamped to zero.

Result: Simulate one Loihi unit with one input synapse for t_{max} time steps and read out state variables (I, v) and spikes (σ).

```
# Define round away from zero
```
$\texttt{rnd}(\cdot) \coloneqq \text{sign}(\cdot)\lceil|\cdot|\rceil$
```
# Define input spike train
```
$S_t = \{0, 1\}\ \forall\ t \in \mathbb{N} \mid t \leq t_{max}$
```
# Define synaptic weight
```
$J \coloneqq 2^{6+\Theta} \cdot \tilde{w},\ \Theta \in [-8, 7],\ \tilde{w} \in [-256, 255]$
```
# Define threshold
```
$v_{th} \coloneqq v_{mant} \cdot 2^6,\ v_{mant} \in [0, 131071]$
```
# Define voltage and synaptic input decay
```
$\tau_v^{-1} = \delta^v/2^{12},\ \delta^v \in [0, 4096]$

$\tau_I^{-1} = \delta^I/2^{12},\ \delta^I \in [0, 4096]$
```
# Initialize variables
```
$I_t, v_t, \sigma_t = 0\ \forall\ t \in \mathbb{N} \mid t \leq t_{max}$
```
# Loop over simulation steps
```
for t *from* 1 *to* t_{max} **do**
```
    # Spike input
```
    $s \leftarrow S_t$
```
    # Update and read synaptic input
```
    $I_t \leftarrow I_{t-1} - \texttt{rnd}(\tau_I^{-1} \cdot I_{t-1}) + J \cdot s$
```
    # Update and read voltage
```
    $v_t \leftarrow v_{t-1} - \texttt{rnd}(\tau_v^{-1} \cdot v_{t-1}) + I_t$
```
    # Check threshold
```
    if $v > v_{th}$ **then**
```
        # Read spike
```
        $\sigma_t \leftarrow 1$
```
        # Reset voltage
```
        $v_t \leftarrow 0$
    end
end
```

Algorithm 1. Loihi single neuron emulator.

# 3. Network and plasticity

We now have a working implementation of `Loihi`'s spiking unit. In the next step, we need to connect these units up into networks. And if the network should be able to learn online, connections between units should be plastic. In this section, we review how weights are defined on `Loihi` and how learning rules are applied. This includes the calculation of pre- and post-synaptic traces. Based on this, we outline how these features are implemented in the emulator.

## 3.1. Synaptic weights

The synaptic weight consists of two parts, a weight mantissa $\tilde{w}$ and a weight exponent $\Theta$ and is of the form $\tilde{w} \cdot 2^{6+\Theta}$. However,
```

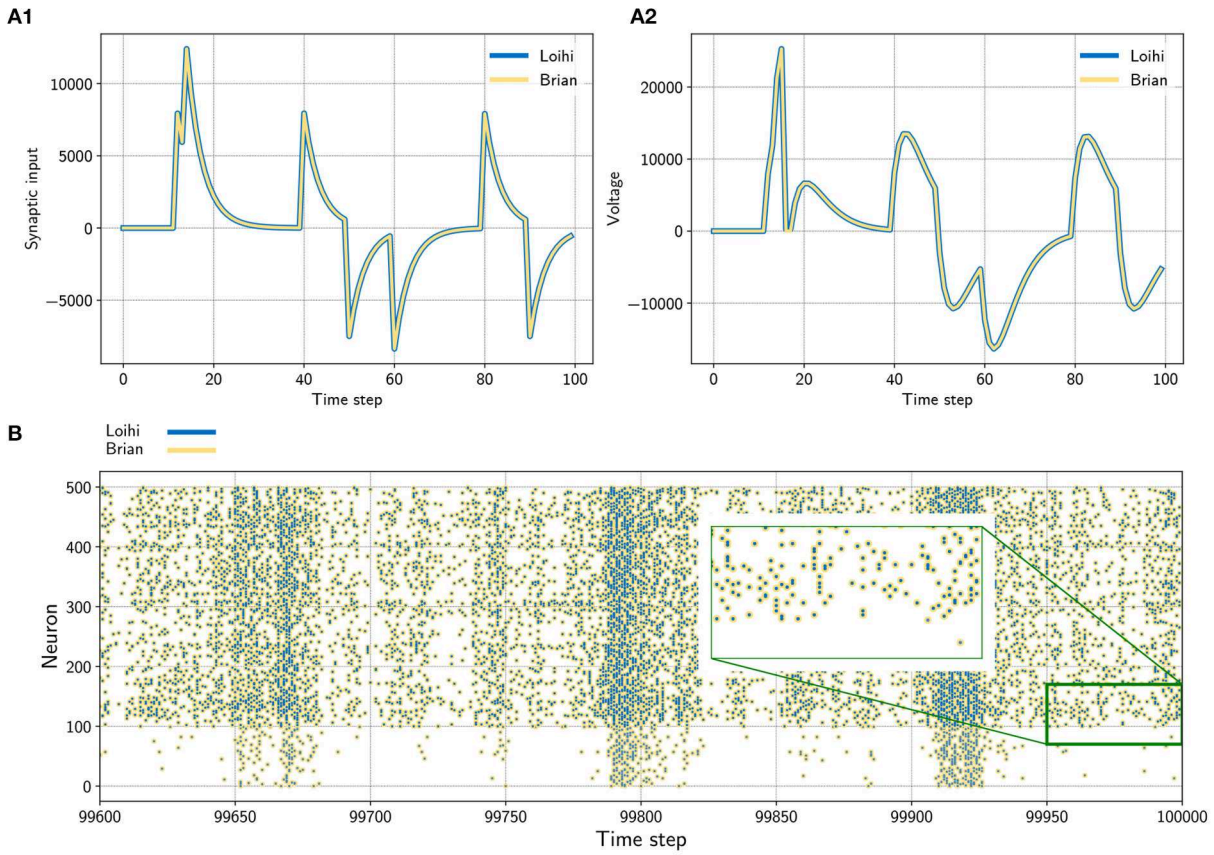


FIGURE 1
(A) Input trace of a single synapse (left) and voltage trace (right) of a neuron. The neuron receives randomly timed excitatory and inhibitory input spikes. The emulator (yellow) matches `Loihi` (blue) in both cases perfectly. Note that `Loihi` uses the voltage register to count refractory time, which results in a functionally irrelevant difference after a spike, e.g time step 17 in A2 (see Appendix 9.2.2). **(B)** Network simulation with 400 excitatory (indices 100 – 500) and 100 inhibitory (indices 0 – 100) neurons. The network is driven by noise from an input population of 40 Poisson spike generators with a connection probability of 0.05. All spikes match exactly between the emulator and `Loihi` for all time steps. The figure shows the last 400 time steps from a simulation with 100, 000 time steps.

in practice the calculation of the synaptic weight depends on bit shifts and its precision depends on a few parameters (see below). The weight exponent is a value between −8 and 7 that scales the weight mantissa exponentially. Depending on the sign mode of the weight (excitatory, inhibitory, or mixed), the mantissa is an integer in the range $\tilde{w} \in [0, 255]$, $\tilde{w} \in [-255, 0]$, or $\tilde{w} \in [-256, 254]$, respectively. The possible values of the mantissa depend on the number of bits available for storing the weight and whether the sign mode is *mixed* or not. In particular, precision is defined as 2^{n_s}, with

$$n_s = 8 - (n_{wb} - \sigma_{\text{mixed}}). \tag{17}$$

This can intuitively be understood with a few examples. If the weight bits for the weight mantissa are set to the default value of $n_{wb} = 8$ bits, it can store 256 values between 0 and 255, i.e., the precision is then $2^{8-(8-0)} = 2^0 = 1$. If $n_{wb} = 6$ bits is chosen, we instead have a precision of $2^{8-(6-0)} = 2^2 = 4$ meaning there are 64 possible values for the weight mantissa, $\tilde{w} \in \{0, 4, 8, 16, ..., 252\}$. If the sign mode is *mixed*, i.e., $\sigma_{mixed} = 1$, one bit is used to store the sign, which reduces the precision. Mixed mode enables both positive and negative weights, with weight mantissa between −256 and 254. Assuming $n_{wb} = 8$ in mixed mode, precision is $2^{8-(8-1)} = 2^1 = 2$ and $\tilde{w} \in \{-256, -254, ..., -4, -2, 0, 2, 4, ..., 254\}$.

3.1.1. Weight initialization

While the user can define an arbitrary weight mantissa within the allowed range, during initialization the value is rounded, given the precision, to the next possible value toward zero. This is achieved *via* bit shifting, that is the weight mantissa is shifted by

$$\tilde{w}^{\text{shifted}} = (\tilde{w} \gg n_s) \ll n_s, \tag{18}$$

where $\gg$ and $\ll$ are a right and left shift respectively. Afterwards the weight exponent is used to scale the weight according to

$$J^{\text{scaled}} = \tilde{w}^{\text{shifted}} \cdot 2^{6+\Theta}. \tag{19}$$

This value cannot be greater than 21 bits and is clipped if it exceeds this limit. Note that this only happens in one case for $\tilde{w} = -256$ and $\Theta = 7$. Finally the scaled value J^{scaled} is shifted again according to

$$J = (J^{\text{scaled}} \gg 6) \ll 6, \tag{20}$$

where J is the final weight.

We provide a table with all 4096 possible weights depending on the mantissa and the exponent in a `Jupyter` notebook[5]. These values are provided for all three sign modes.

3.1.2. Plastic synapses

In the case of a *static* synapse, the initialized weight remains the same as long as the chip/emulator is running. Thus *static* synapses are fully described by the details above. For *plastic* synapses, the weight can change over time. This requires a method to ensure that changes to the weight adhere to its precision.

For *plastic* synapses, *stochastic rounding* is applied to the mantissa during each weight update. Whether the weight mantissa is rounded up or down depends on its proximity to the nearest possible values above and below, i.e.,

$$\text{RS}_{2^{n_s}}(x) = \begin{cases} \text{sign}(x) \cdot \lfloor |x| \rfloor_{2^{n_s}} & \text{with probability } (2^{n_s} - (|x| - \lfloor |x| \rfloor_{2^{n_s}}))/2^{n_s} \\ \text{sign}(x) \cdot (\lfloor |x| \rfloor_{2^{n_s}} + 2^{n_s}) & \text{with probability } (|x| - \lfloor |x| \rfloor_{2^{n_s}}))/2^{n_s} \end{cases} \tag{21}$$

where $\lfloor \cdot \rfloor_{2^{n_s}}$ denotes rounding down to the nearest multiple of 2^{n_s}. After the mantissa is rounded, it is scaled by the weight exponent and the right/left bit shifting is applied to the result to compute the actual weight J. How this is realized in the emulator is shown in Code Listing 3.

To test that our implementation of the weight update for *plastic* synapses matches `Loihi` for each possible number of weight bits, we compared the progression of the weights over time for a simple learning rule. The analysis is described in detail in Appendix 9.4.

3.2. Pre- and post-synaptic traces

Pre- and post-synaptic traces are used for defining learning rules. `Loihi` provides two pre-synaptic traces x_1, x_2 and three post-synaptic traces y_1, y_2, y_3. Pre-synaptic traces are increased by a constant value $\hat{x}_i$, for $i \in \{1, 2\}$, if the pre-synaptic neuron spikes. The post-synaptic traces are increased by $\hat{y}_j$ for $j \in \{1, 2, 3\}$, accordingly. So-called *dependency factors* are available, indicating events like $x_0 = 1$ if the pre-synaptic neuron spikes or $y_0 = 1$ if the post-synaptic neuron spikes. These factors can be combined with the trace variables by addition, subtraction, or multiplication.

A simple spike-time dependent plasticity (STDP) rule with an asymmetric learning window would, for example, look like $dw = x_1 \cdot y_0 - y_1 \cdot x_0$. This rule leads to a positive change in the weight ($dw > 0$) if the pre-synaptic neuron fires shortly before the post-synaptic neuron (i.e., positive trace $x_1 > 0$ when $y_0 = 1$) and to a negative change ($dw < 0$) if the post-synaptic neuron fires shortly before the pre-synaptic neuron (i.e., positive trace $y_1 > 0$ when $x_0 = 1$). Thus, the time window in which changes may occur depends on the shape of the traces (i.e., impulse strength $\hat{x}_i$, $\hat{y}_i$; and decay τ_{x_i}, τ_{y_j}, see below).

For a sequence of spikes $s[t] \in \{0, 1\}$, a trace is defined as

$$x_i[t] = \alpha \cdot x_i[t-1] + \hat{x}_i \cdot s[t], \tag{22}$$

where α is a decay factor (see Davies et al., 2018). This equation holds for presynaptic (x_i) and postsynaptic (y_i) traces. However, in practice, on `Loihi` one does not set α directly but instead decay time constants τ_{x_i} and τ_{y_j}.

In the implementation of the emulator we again assume a first order approximation for synaptic traces, akin to synaptic input and voltage. Under this assumption for the exponential decay, in Equation (22) we replace α by

$$\alpha(\tau_{x_i}) = 1 - \frac{1}{\tau_{x_i}}. \tag{23}$$

Using this approximation gives reasonable results across a number of different τ_{x_i} and τ_{y_i} values (see Figure A2). While this essentially suffices, it could be improved by introducing an additional parameter, e.g., β, and optimizing $\alpha(\tau_{x_i}, \beta)$.

Note that we have integer precision again. But different from the *round away from zero* applied in the neuron model, here *stochastic rounding* is used. Since traces are positive values between 0 and 127 with precision 1, the definition above in Equation (21) simplifies to the following

$$\text{RS}_{1,\geq 0}(x) = \begin{cases} \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \\ \lfloor x \rfloor + 1 & \text{with probability } x - \lfloor x \rfloor \end{cases} \tag{24}$$

Since this rounding procedure is probabilistic and the details of the random number generator are unknown, rounding introduces discrepancies when emulating `Loihi` on the computer. Further improvements are possible if more details of the chip's rounding mechanism were to be considered.

5 https://github.com/sagacitysite/brian2_loihi_utils/blob/main/algorithm/02_weight-calculation.ipynb

3.3. Summary

At this point we are able to connect neurons with synapses and build networks of neurons (see Figure 1B). It was shown how the weights are handled, depending on the user defined number of weight bits or the sign mode. In addition, using the dynamics of the pre- and post synaptic traces, we can now define learning rules. Note that different from the neuron model, the synaptic traces cannot be reproduced exactly since the details of the random number generator, used for stochastic rounding, are unknown. However, Figure 2 shows that the synaptic traces emulated in `Brian` are very close to the original ones in `Loihi` and that the behavior of a standard asymmetric STDP rule can be reproduced with the emulator.

4. Loihi emulator based on Brian

Here we provide an overview over the emulator package and show some examples and results. This enables straightforward emulation of the basic features from `Loihi` as a sandbox for experimenters. Note that we have explicitly not included routing and mapping restrictions, like limitations for the number of neurons or the amount of synapses, as these depend on constraints such as the number of used `Loihi` chips.

4.1. The package

The emulator package is available on *PyPI*[6] and can be installed using the `pip` package manager. The emulator does not provide all functionality of the `Loihi` chip and software, but the main important aspects. An overview over all provided features is given in Table A1 (Appendix). It contains six classes that extend the corresponding `Brian` classes. The classes are briefly introduced in the following. Further details can be taken from the code.[7]

4.1.1. Network

The `LoihiNetwork` class extends the `Brian Network` class. It provides the same attributes as the original `Brian` class. The main difference is that it initializes the default clock, the integration methods and updates the schedule when a `Network` instance is created. Note that it is necessary to make explicitly use of the `LoihiNetwork`. It is not possible to use `Brian`'s *magic network*.

Voltage and synaptic input are evolved with the forward Euler integration method, which was introduced in Section 2.2.

```
lif_equations = '''
    rnd_v = sign(v)*ceil(abs(v*1_tau_v)) : 1
    rnd_I = sign(I)*ceil(abs(I*1_tau_I)) : 1
    dv/dt = -rnd_v/ms + I/ms: 1 (unless refractory)
    dI/dt = -rnd_I/ms : 1
'''
```

Neuron model equations of the voltage and the synaptic input for `Brian`. It contains a *round away from zero* rounding.

Additionally a state updater was defined for the pre- and post-synaptic traces.

The default network update schedule for the computational order of the variables from `Brian` do not match the order of the computation on `Loihi`. The `Brian` update schedule is therefore altered when initializing the `LoihiNetwork`, more details are given in Appendix 9.2.1.

4.1.2. Neuron group

The `LoihiNeuronGroup` extends `Brian`'s `NeuronGroup` class. Parameters of the `LoihiNeuronGroup` class are mostly different from the `Brian` class and are related to `Loihi`. When an instance is created, the given parameters are first checked to match requirements from `Loihi`. Finally, the differential equations to describe the neural system are shown in Code Listing 1. Since `Brian` does not provide a *round away from zero* functionality, we need to define it manually as an equation.

4.1.3. Synapses

The `LoihiSynapses` class extends the `Synapses` class from `Brian`. Again, most of the `Brian` parameters are not supported and instead `Loihi` parameters are available. When instantiating a `LoihiSynapses` object, the needed pre- and post-synaptic traces are included as equations (shown in Code Listing 2) as theoretically introduced in Section 3.2. Moreover, it is verified that the defined learning rule matches the available variables and operations supported by `Loihi`. The equations for the weight update is shown in Code Listing 3.

Since we have no access to the underlying mechanism and we cannot reproduce the pseudo-stochastic mechanisms exactly, we have to find a stochastic rounding that matches `Loihi` in distribution. Note that on `Loihi` the same network configuration leads to reproducible results (i.e., same rounding). Thus to compare the behavior of `Loihi` and the emulator, we simulate over a number of network settings and compare the distribution of the traces. Figure 2B shows the match between the distributions. Note that with this, our implementation is always slightly different from the `Loihi` simulation, due to slight differences in rounding. In Figure 2C, we show that these variations are constant and not diverging. In addition,

6 https://pypi.org/project/brian2-loihi/

7 https://github.com/sagacitysite/brian2_loihi/

```
x1decay_equations = '''
    x1_new = x1 * (1 - (1.0/tau_x1)) : 1
    x1_int = int(x1_new) : 1
    x1_frac = x1_new - x1_int : 1
    x1_add_or_not = int(x1_frac > rand()) : 1 (
        constant over dt)
    x1_rnd = x1_int + x1_add_or_not : 1
    dx1/dt = x1_rnd / ms : 1 (clock-driven)
'''
```

Synaptic decay equation for `Brian`. Only the decay for *x1* is shown, the decay for *x2*, *y1*, *y2*, *y3* is applied analogously. It contains an approximation of the exponential decay and stochastic rounding.

```
weight_equations = '''
    u0 = 1 : 1
    u1 = int(t/ms % 2**1 == 0) : 1
    ...
    u9 = int(t/ms % 2**9 == 0) : 1

    dw_rounded = int(sign(dw)*ceil(abs(dw))) : 1
    quotient = int(dw_rounded / precision) : 1
    remainder = abs(dw_rounded) % precision : 1
    prob = remainder / precision : 1
    add_or_not = sign(dw_rounded) * int(prob > rand())
        : 1 (constant over dt)
    dw_rounded_to_precision = (quotient + add_or_not)
        * precision : 1
    w_updated = w + dw_rounded_to_precision : 1
    w_clipped = clip(w_updated, w_low, w_high) : 1
    dw/dt = w_clipped / ms : 1 (clock-driven)

    w_act_scaled = w_clipped * 2**(6 + w_exp) : 1
    w_act_scaled_shifted = int(floor(w_act_scaled /
        2**6)) * 2**6 : 1
    w_act_clipped = clip(w_act_scaled_shifted, -limit,
        limit) : 1
    dw_act/dt = w_act_clipped / ms : 1 (clock-driven)

    dx0/dt = 0 / ms : 1 (clock-driven)
    dy0/dt = 0 / ms : 1 (clock-driven)
'''
```

Weight equations for `Brian`. The first part creates variables that allow terms of the plasticity rule to be evaluated only at the 2^k time step. *dw* contains the user defined learning rule. The updated weight mantissa is adapted depending on the number of weight bits, which determines the precision. The weight mantissa is rounded with *stochastic rounding*. After clipping, the weight mantissa is updated and the actual weight is calculated.

Figure 2D shows that the principle behavior of a learning rule is preserved.

4.1.4. State monitor and Spike monitor

The `LoihiStateMonitor` class extends the `StateMonitor` class from `Brian`, while the `LoihiSpikeMonitor` class extends the `SpikeMonitor` class. Both classes support the most important parameters from their subclasses and update the schedule for the timing of the probes. This schedule update avoids shifts in the monitored variables compared to `Loihi`.

4.1.5. Spike generator group

The `LoihiSpikeGeneratorGroup` extends the `SpikeGeneratorGroup` class from `Brian`. This class only reduces the available parameters to avoid that users unintentionally change variables which would cause an unwanted emulation behavior.

4.2. Examples

To demonstrate that the `Loihi` emulator works as expected, we provide three examples covering a single neuron, a recurrently connected spiking neural network, and the application of a learning rule. All three examples are available as `Jupyter` notebooks.[8]

4.2.1. Neuron model

In a first test, we simulated a single neuron. The neuron receives randomly timed excitatory and inhibitory input spikes. Figure 1A1 shows the synaptic responses induced by the input spikes for the simulation using the `Loihi` chip and the `Loihi` emulator. The corresponding voltage traces are shown in Figure 1A2. As expected, the synaptic input as well as the voltage match perfectly between the hardware and the emulator.

4.2.2. Network

In a second approach we applied a recurrently connected network of 400 excitatory and 100 inhibitory neurons with log-normal weights. The network gets noisy background input from 40 Poisson generators that are connected to the network with a probability of 0.05. As already shown by others, this setup leads to a highly chaotic behavior (Sompolinsky et al., 1988; Van Vreeswijk and Sompolinsky, 1996; Brunel, 2000; London et al., 2010). Despite the chaotic dynamics, spikes, voltages and synaptic inputs match perfectly for all neurons and over the whole time. The spiking pattern of the network is shown in Figure 2B. All yellow (`Brian`) and blue (`Loihi`) dots match perfectly.

4.2.3. Learning

In the last experiment, we applied a simple STDP learning rule, as introduced in Equation (25), at a single plastic synapse. The experiment is sketched in Figure 2A. One spike generator, denoted *input*, has a plastic connection to a neuron with a very low weight ($\tilde{w} = 128$, $\Theta = -6$), such that it has a negligible effect on the post-synaptic neuron. Another spike generator, denoted *noise*, has a large but static weight ($\tilde{w} = 254$, $\Theta = 0$)

8 https://github.com/sagacitysite/brian2_loihi_utils/tree/main/examples

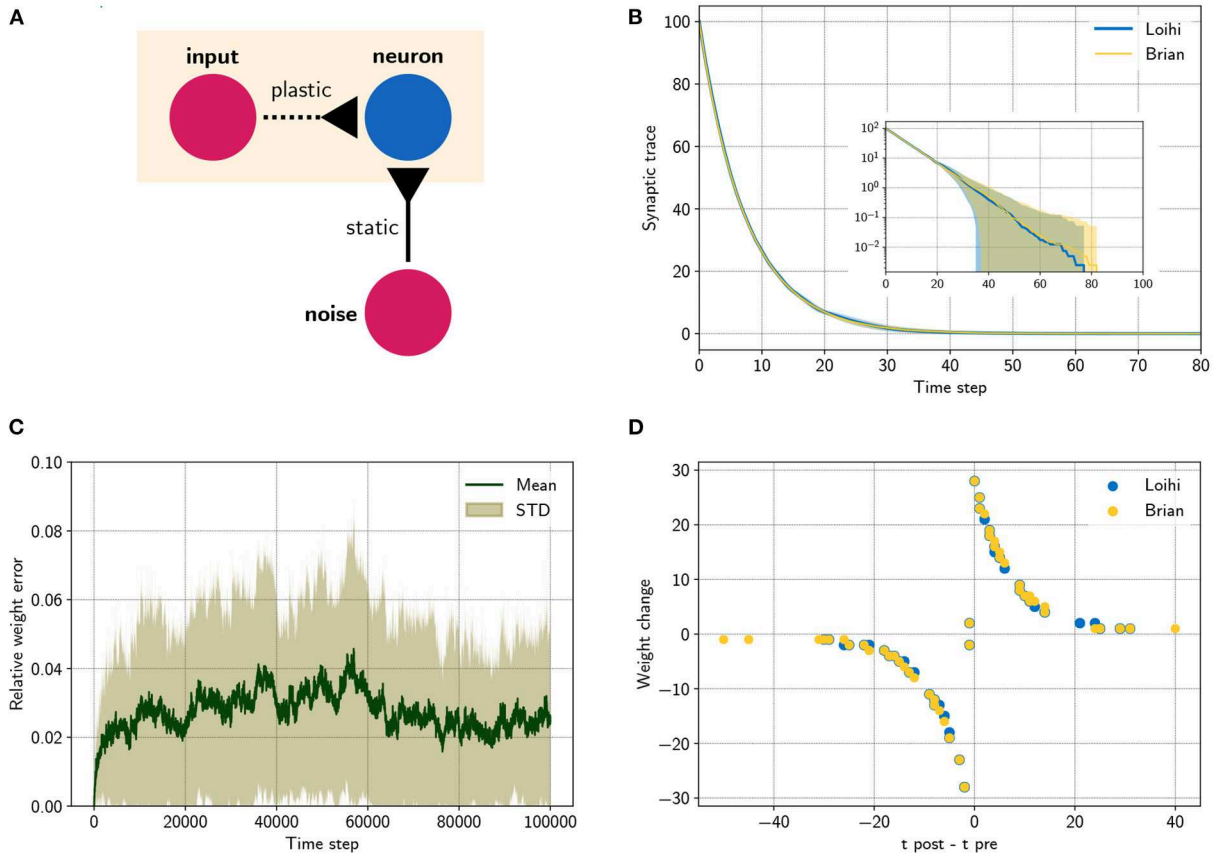


FIGURE 2
Comparing a STDP learning rule performed with the emulator and with `Loihi`. **(A)** Sketch showing the setup. **(B)** Synaptic trace for many trials showing the arithmetic mean and standard deviation. The inset shows the same data in a logarithmic scale. Note that every data point smaller than 10^0 shows the probability of rounding values between 0 and 1 up or down. **(C)** Relative difference $|\tilde{w}_L - \tilde{w}_B|/\tilde{w}_{\max}$ for the plastic weight between the emulator, $\tilde{w}_B$, and the `Loihi` implementation, $\tilde{w}_L$, for 50 simulations, $\tilde{w}_{\max} = 255$. **(D)** STDP weight change in respect to pre- and post-synaptic spike times, data shown for time steps $0 - 2,000$ for visualization purposes.

to reliably induce post-synaptic spikes. Figure 2B compares the distribution of traces between the emulator and `Loihi`. For this 400 trials were simulated.

We chose an asymmetric learning window for the STDP rule. The learning rule uses one pre-synaptic trace x_1 ($\hat{x}_1 = 120$, $\tau_{x_1} = 8$) and one post-synaptic trace y_1 ($\hat{y}_1 = 120$, $\tau_{y_1} = 8$). In addition the dependency factors $x_0 \in 0, 1$ and $y_0 \in 0, 1$ are used, which indicate a pre- and post-synaptic spike respectively. Using these components, the learning rule is defined as

$$dw = 2^{-2} \cdot x_1 \cdot y_0 - 2^{-2} \cdot x_0 \cdot y_1. \tag{25}$$

Due to the stochastic rounding of the traces, differences in the weight changes occur, which are shown in Figure 2C. Fortunately, the relative weight error remains low at a constant level of 0.027 ± 0.027 and does not diverge, even over long simulation times, e.g., 100 000 steps. Despite these variations, the STDP learning window of the emulator reproduces the behavior of the `Loihi` learning window, as shown in Figure 2D.

4.3. Performance tests

An important argument for the development of the `Brian2Loihi` emulator was—besides improving the understanding of `Loihi`'s functionality—its usefulness for prototyping. When developing new models, algorithms, and applications, often large parameter scans are performed in which many networks with different parameter sets are initialized and executed. During this process, it is crucial to be able to read out spiking information to measure performance. For this reason we measured initialization and execution times both with and without spike monitoring on the `Loihi` chip and in the `Loihi` emulator.

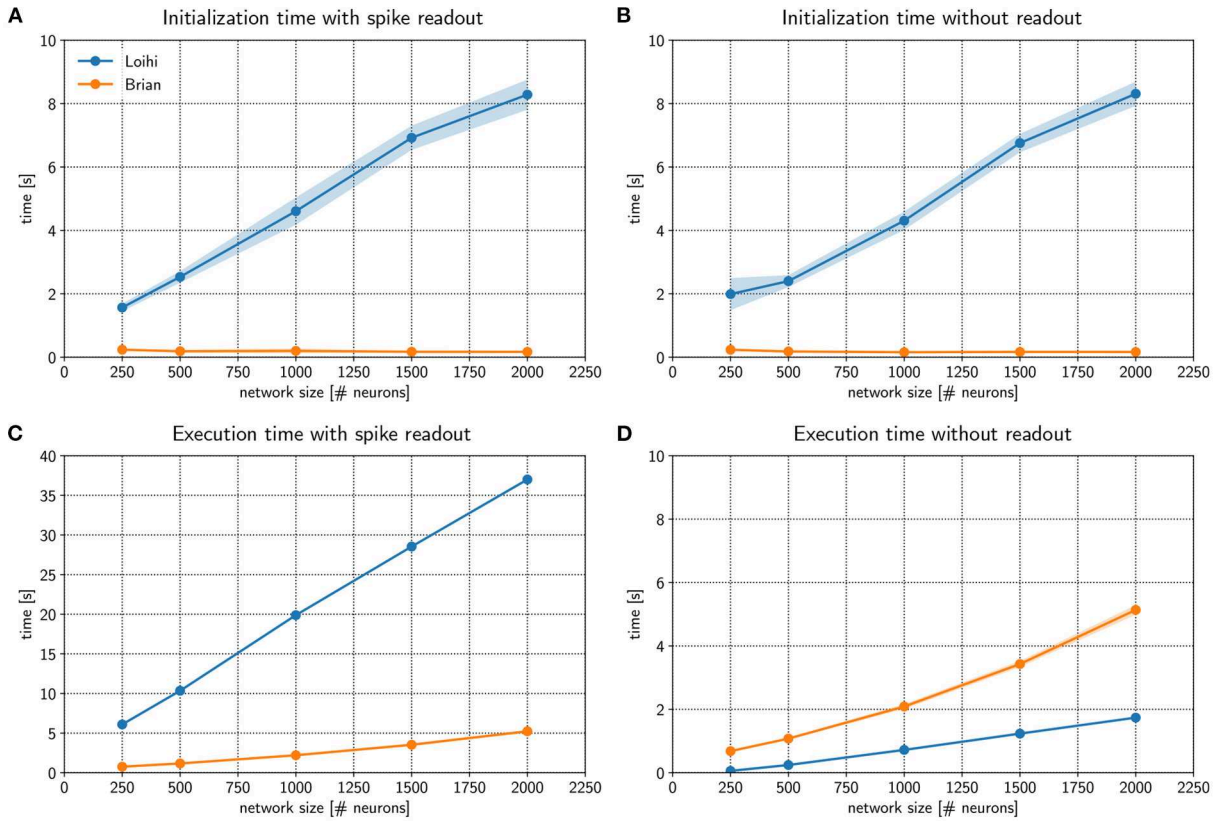


FIGURE 3
Comparing performance of the `Loihi` emulator with the `Loihi` chip. The initialization time **(A,B)** and execution time **(C,D)** for `Loihi`-based (blue) and emulator-based (orange) simulations was compared for different network sizes. In one case spikes were read out for all neurons and time steps **(A,C)** and in a second case no spikes were read out **(B,D)**. The `Brian`-based simulation using `Brian2Loihi` had faster initialization times across all network sizes, both with and without spike monitoring. For the execution time, a `Brian`-based simulation was only faster when a read out was defined. If no spikes were read out, `Loihi`-based execution is faster. Execution time both on the `Loihi` chip and using the emulator increase with network size. Points show the mean and shaded areas show the standard deviation over 5 trials.

Figure 3A compares *initialization* times for a randomly connected network with different sizes. Networks were stimulated with background noise to maintain a consistent firing rate. Note that more details about the network implementation are provided in Appendix 9.2.2. From the figure, it is clear that `Loihi` takes much more time to setup the network compared to the emulator based on `Brian`. If no spiking information is read out from the network during simulation, the result is quite similar, as shown in Figure 3B. `Brian2Loihi` reduces the initialization time drastically, in particular for larger networks. This boost in initialization time is highly valuable for parameter scans across many network configurations.

We were also interested in the comparison for the *execution* times of the simulation. Figure 3C compares `Loihi`- and `Brian`-based execution times if all spikes were read out. Clearly, `Brian2Loihi` is much faster and the difference becomes larger as the network size grows. However, if no read out is performed, Figure 3D shows that in this case `Loihi` is faster in executing the simulation across all network sizes. Therefore, `Brian2Loihi` is more efficient for prototyping networks, when we depend on analyzing comprehensive data from the networks' behavior. For applications where a read out is not important or only few spikes must be read out, execution on `Loihi` is faster.

This underlines the significance of the `Brian2Loihi` emulator for prototyping on one hand and shows the potential of `Loihi` for large and long-term network simulations on the other hand. Note, however, that due to longer initialization times on `Loihi`, faster execution times are likely beneficial only if network initialization must not be performed often, readout is minimal, and the simulation time is long. In many cases, choosing a `Brian`-based simulation for development and a `Loihi`-based simulation for productive use cases could be an efficient combination in our view.

4.4. Applications

As a starting point for working with the emulator beyond the examples above, here we briefly describe two more complex applications

implemented using the emulator. The code is openly available.

4.4.1. Anisotropic network

In a recent study, we showed that a recurrently connected neural network with spatially inhomogeneous locally correlated connectivity (i.e., "the anisotropic network", for original model see Spreizer et al., 2019) could be implemented on `Loihi` to generate noise-robust trajectories for robotic movements (Michaelis et al., 2020). This biologically plausible network model can generate stable sequences of neural activity on the timescale of behavior, making it interesting for both neuroscience and for neuromorphic applications. We implemented this network in the `Loihi` emulator and made it publicly available on GitHub.[9]

4.4.2. SSSP

The goal of the Single Source Shortest Path (SSSP) problem is to find the shortest path from a start node to a target node in a given graph. Spiking neuronal networks can solve the problem through a wave front algorithm (Ponulak and Hopfield, 2013). Within this algorithm a wave of spikes propagates through a network of neurons that acts to represent the graph. The algorithm stops when the target neuron spikes. To enable path back tracing a local learning rule alters the weights during the wave propagation phase accordingly. An implementation using the `Loihi` emulator is available on GitHub.[10]

Furthermore, a new type of the SSSP algorithm for neuromorphic hardware was developed using the `Loihi` emulator, the so-called add-and-minimize (AM) algorithm (Michaelis, 2022, Appendix 9.5). It is capable of solving the SSSP problem for larger graphs, especially when the costs of the edges have a higher resolution. The code is again publicly available.[11]

5. Discussion

This study was motivated by two goals. We hope to simplify the transfer of models to `Loihi` and therefore developed a `Loihi` emulator for `Brian`, featuring many functionalities of the `Loihi` chip. In the process of developing the emulator, we aimed to provide a deeper understanding of the functionality of the neuromorphic research chip `Loihi` by analyzing its neuron and synapse model, as well as synaptic plasticity.

We hope that the analysis of `Loihi`'s spiking units has provided some insight into how `Loihi` computes. With the numerical integration method, numerical precision and related rounding method, as well as the update schedule, we were able to walk from the LIF neuron model down to the computations performed. For neurons and networks without plasticity we are able to emulate `Loihi` without error. Analyzing and implementing synaptic plasticity showed that, due to stochastic rounding, it is not possible to exactly replicate trial by trial behavior when it comes to learning. However, on average the weight changes induced by a learning rule are preserved.

The main benefit of the `Brian2Loihi` emulator lies in lowering the hurdle for the experimenter. Especially in neuroscience, many scientists are accustomed to neuron simulators and in particular `Brian` is widely used. It makes a deep dive into new software frameworks and hardware systems unnecessary. The emulator can be used for simple and fast prototyping, as it improves the initialization time in all cases drastically and the execution time, when a read out is used. In addition, hardware specific complications, like distributing neurons to cores, or constraints like potential limits on the number of available neurons or synapses, or on the speed or size of read-out, do not occur in the emulator. While this will surely improve with new generations of hardware and software in the upcoming years, they can already be ignored by using the emulator.

At this point it is important to note that not all `Loihi` features are included in the emulator, yet. In particular, the homeostasis mechanism, rewards, and tags for the learning rule are not included. In Table A1, we provide a comparison of all functionalities from `Loihi` with those available in the current state of the emulator. Development of this emulator is an open source project and we expect improvements and additions with time. Note that a follow up project, called `Brian2Lava` has already started.[12]

An important vision for the future is to flexibly connect front-end development environments (e.g., `Brian`, NEST, Keras, TensorFlow) with various back-ends, like neuromorphic platforms (e.g., `Loihi`, SpiNNaker, BrainScaleS, Dynap-SE) or emulators for these platforms. PyNN (Davison et al., 2009) is such an approach to unify different front-ends and back-ends in a more general way. Nengo (Bekolay et al., 2014), as another approach, does not provide the use of other simulators, but allows several back-ends and focuses on higher level applications (DeWolf et al., 2020). NxTF (Rueckauer et al., 2021) is an API and compiler aimed at simplifying the efficient deployment of deep convolutional spiking neural networks on `Loihi` using an interface derived from Keras. We think that ideally, one could continue to work in their preferred front-end environment while a package maps their code to existing chips or computer-based emulators of these chips. We expect an interface along these lines will play an important role in the future of neuromorphic computing and want to contribute to this development with our `Brian2Loihi` emulator.

9 https://github.com/andrewlehr/Brian2Loihi_SpreizerNet

10 https://github.com/Winnus/Brian2Loihi_SSSP

11 https://github.com/elena-off/sssp-loihiemulator

12 https://gitlab.com/tetzlab/brian2lava

At least for now, with an emulator at hand, it is easier to prototype network models and assess whether an implementation on `Loihi` is worth considering. When getting started with neuromorphic hardware, to e.g., scale up models or speed up simulations, researchers familiar with `Brian` can directly deploy models prepared with the emulator. We hope that with this, others may find a smooth entry into the quickly emerging field of neuromorphic computing.

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

CM, AL, and WO analyzed Loihi's neuron and synapse model, with a larger contribution from AL and tested and refined the emulator implementation. CM programmed the emulator. WO performed the simulations and created the main figures and edited and reviewed. CM and AL created the supplementary figures and wrote the text. CT acquired funding and supervised the study. All authors reviewed the manuscript. All authors contributed to the article and approved the submitted version.

Funding

This study received funding from Intel Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

Acknowledgments

The work received funds by the Intel Corporation *via* a gift without restrictions. AL currently holds a Natural Sciences and Engineering Research Council of Canada PGSD-3 scholarship. We would like to thank Jonas Neuhöfer, Sebastian Schmitt, Andreas Wild, and Terrence C. Stewart for valuable discussions and input.

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.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2022.1015624/full#supplementary-material

References

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinformatics* 7, 48. doi: 10.3389/fninf.2013.00048

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., et al. (2019). Spiking neural networks hardware implementations and challenges: a survey. *ACM J. Emerg. Technol. Comput. Syst.* 15, 1–35. doi: 10.1145/3304103

Brüderle, D., Petrovici, M. A., Vogginger, B., Ehrlich, M., Pfeil, T., Millner, S., et al. (2011). A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biol. Cybernet.* 104, 263–296. doi: 10.1007/s00422-011-0435-9

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *J. Physiol.* 94, 445–463. doi: 10.1016/S0928-4257(00)01084-6

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., et al. (2021). Advancing neuromorphic computing with Loihi: a survey of results and outlook. *Proc. IEEE.* 109, 911–934. doi: 10.1109/JPROC.2021.3067593

Davies, M., Srinivasa, N., Lin, T., 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

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). PyNN: a common interface for neuronal network simulators. *Front. Neuroinformatics* 2, 11. doi: 10.3389/neuro.11.011.20080

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

DeWolf, T., Stewart, T. C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proc. R. Soc. B Biol. Sci.* 283, 20162134. doi: 10.1098/rspb.2016.2134

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

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

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. New York, NY: Cambridge University Press. doi: 10.1017/CBO9781107447615

Grüning, A., and Bohte, S. M. (2014). "Spiking neural networks: principles and challenges," in *ESANN* (Bruges).

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., et al. (2018). Programming spiking neural networks on Intels Loihi. *Computer* 51, 52–61. doi: 10.1109/MC.2018.157113521

London, M., Roth, A., Beeren, L., Häusser, M., and Latham, P. E. (2010). Sensitivity to perturbations *in vivo* implies high noise and suggests rate coding in cortex. *Nature* 466, 123–127. doi: 10.1038/nature09086

Luo, T., Wang, X., Qu, C., Lee, M. K. F., Tang, W. T., Wong, W.-F., et al. (2018). An FPGA-based hardware emulator for neuromorphic chip with RRAM. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 39, 438–450. doi: 10.1109/TCAD.2018.2889670

Michaelis, C. (2020). PeleNet: a reservoir computing framework for Loihi. *arXiv preprint arXiv:2011.12338*. doi: 10.48550/ARXIV.2011.12338

Michaelis, C., Lehr, A. B., and Tetzlaff, C. (2020). Robust trajectory generation for robotic control on the neuromorphic research chip Loihi. *Front. Neurorobot.* 14, 589532. doi: 10.3389/fnbot.2020.589532

Michaelis, C. (2022). *Think local, act global: robust and real-time movement encoding in spiking neural networks using neuromorphic hardware* (Ph.D. thesis). Göttingen: University Goettingen Repository.

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 preprint arXiv:2003.13749*. doi: 10.48550/ARXIV.2003.13749

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*. doi: 10.48550/ARXIV.2003.13750

Petrovici, M. A., Vogginger, B., Müller, P., Breitwieser, O., Lundqvist, M., Muller, L., et al. (2014). Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PLoS ONE* 9, e108590. doi: 10.1371/journal.pone.0108590

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12, 774. doi: 10.3389/fnins.2018.00774

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

Rajendran, B., Sebastian, A., Schmuker, M., Srinivasa, N., and Eleftheriou, E. (2019). Low-power neuromorphic hardware for signal processing applications: a review of architectural and system-level design approaches. *IEEE Signal Process. Mag.* 36, 97–110. doi: 10.1109/MSP.2019.2933719

Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., et al. (2018). sPyNNaker: a software package for running PyNN simulations on SpiNNaker. *Front. Neurosci.* 12, 816. doi: 10.3389/fnins.2018.00816

Rueckauer, B., Bybee, C., Goettsche, R., Singh, Y., Mishra, J., and Wild, A. (2021). NxTF: an API and compiler for deep spiking neural networks on intel Loihi. *arXiv preprint arXiv:2101.04261*. doi: 10.1145/3501770

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., et al. (2016). "Truenorth ecosystem for brain-inspired computing: scalable systems, software, and applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT: IEEE), 130–141. doi: 10.1109/SC.2016.11

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*. doi: 10.48550/ARXIV.1705.06963

Sompolinsky, H., Crisanti, A., and Sommers, H.-J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61, 259. doi: 10.1103/PhysRevLett.61.259

Spilger, P., Müller, E., Emmel, A., Leibfried, A., Mauch, C., Pehle, C., et al. (2020). "hxtorch: Pytorch for brainscales-2," in *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning* (Ghent: Springer), 189–200. doi: 10.1007/978-3-030-66770-2_14

Spreizer, S., Aertsen, A., and Kumar, A. (2019). From space to time: spatial inhomogeneities lead to the emergence of spatiotemporal sequences in spiking neuronal networks. *PLoS Comput. Biol.* 15, e1007432. doi: 10.1371/journal.pcbi.1007432

Stagsted, R., Vitale, A., Binz, J., Bonde Larsen, L., Sandamirskaya, Y., et al. (2020). "Towards neuromorphic control: a spiking neural network based pid controller for UAV," in *Proceedings of Robotics: Science and Systems* (Corvallis, OR). doi: 10.15607/RSS.2020.XVI.074

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

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. *Front. Neurosci.* 12, 891. doi: 10.3389/fnins.2018.00891

Valancius, S., Richter, E., Purdy, R., Rockowitz, K., Inouye, M., Mack, J., et al. (2020). "FPGA based emulation environment for neuromorphic architectures," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (New Orleans, LA). doi: 10.1109/IPDPSW50202.2020.00022

Van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi: 10.1126/science.274.5293.1724

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

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 13 January 2023
DOI 10.3389/fnins.2022.1050585

OPEN ACCESS

EDITED BY
Gert Cauwenberghs,
University of California, San Diego,
United States

REVIEWED BY
Gina Adam,
George Washington University,
United States
Young-Seok Choi,
Kwangwoon University,
Republic of Korea

*CORRESPONDENCE
Shantanu Chakrabartty
✉ shantanu@wustl.edu

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

RECEIVED 21 September 2022
ACCEPTED 28 December 2022
PUBLISHED 13 January 2023

CITATION
Rahman M, Bose S and Chakrabartty S (2023) On-device synaptic memory consolidation using Fowler-Nordheim quantum-tunneling.
Front. Neurosci. 16:1050585.
doi: 10.3389/fnins.2022.1050585

On-device synaptic memory consolidation using Fowler-Nordheim quantum-tunneling

Mustafizur Rahman, Subhankar Bose and Shantanu Chakrabartty*

Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, United States

Introduction: For artificial synapses whose strengths are assumed to be bounded and can only be updated with finite precision, achieving optimal memory consolidation using primitives from classical physics leads to synaptic models that are too complex to be scaled *in-silico*. Here we report that a relatively simple differential device that operates using the physics of Fowler-Nordheim (FN) quantum-mechanical tunneling can achieve tunable memory consolidation characteristics with different plasticity-stability trade-offs.

Methods: A prototype FN-synapse array was fabricated in a standard silicon process and was used to verify the optimal memory consolidation characteristics and used for estimating the parameters of an FN-synapse analytical model. The analytical model was then used for large-scale memory consolidation and continual learning experiments.

Results: We show that compared to other physical implementations of synapses for memory consolidation, the operation of the FN-synapse is near-optimal in terms of the synaptic lifetime and the consolidation properties. We also demonstrate that a network comprising FN-synapses outperforms a comparable elastic weight consolidation (EWC) network for some benchmark continual learning tasks.

Discussions: With an energy footprint of femtojoules per synaptic update, we believe that the proposed FN-synapse provides an ultra-energy-efficient approach for implementing both synaptic memory consolidation and continual learning on a physical device.

KEYWORDS

hardware synapse, memory consolidation, quantum-tunneling, neuromorphic, continual learning

1. Introduction

There is a growing evidence from the field of neuroscience and neuroscience inspired AI about the importance of implementing synapses as a complex high-dimensional dynamical system (Fusi et al., 2005; Benna and Fusi, 2016), as opposed to a simple and a static storage element, as depicted in standard neural networks (Sohoni et al., 2019). This dynamical systems viewpoint has been motivated by the hypothesis

that complex interactions between plethora of biochemical processes at a synapse (illustrated in Figure 1A) produces *synaptic metaplasticity* (Abraham, 2008) and plays a key role in *synaptic memory consolidation* (Li et al., 2017). Both these phenomena have been observed in biological synapses (Yang et al., 2009, 2014) where the synaptic plasticity (or ease of update) can vary depending on age and task-specific usage that is accumulated during the process of learning. In literature these long-term synaptic memory consolidation dynamics have been captured using different analytical models with varying degrees of complexity (Amit and Fusi, 1994; Fusi, 2002; Fusi et al., 2005; Fusi and Abbott, 2007; Roxin and Fusi, 2013; Benna and Fusi, 2016). One such model is the cascade model (Benna and Fusi, 2016) which has been shown to achieve the theoretically optimal memory consolidation characteristic for benchmark random pattern experiments. However, the physical realization of cascade models as described in Benna and Fusi (2016) uses a complex coupling of dynamical states and diffusion dynamics (an example illustrated in Figure 1B using a reservoir model), which is difficult to mimic and scale *in-silico*. Similar optimal memory consolidation characteristics have been reported in the context of continual learning in artificial neural networks (ANN) where synapses that are found to be important for learning a specific task are consolidated (or become rigid) (Aljundi et al., 2017; Kirkpatrick et al., 2017; Lee et al., 2017; Zenke et al., 2017; Chaudhry et al., 2018; Liu et al., 2018). As a result, when learning a new task the synaptic weight does not significantly deviate from the consolidated weights, hence, the network seeks solutions that work well for as many tasks as possible. However, these synaptic models are algorithmic in nature and it is not clear if the optimal consolidation characteristics can be naturally implemented on the synaptic device *in-silico*. Also, it is not clear if the consolidation properties of the physical synaptic device can be tuned to achieve different *plasticity-stability* trade-offs and hence can overcome the relative disadvantages of the EWC models. In this paper, we report that a simple differential device that operates using the physics of Fowler-Nordheim (FN) quantum-mechanical tunneling can achieve tunable synaptic memory consolidation characteristics similar to the algorithmic consolidation models. The operation of the synaptic device, referred to in this paper as the FN-synapse, can be understood using a reservoir model as shown in Figure 1C). Two reservoirs with fluid levels W^+ and W^- are coupled to each other using a sliding barrier X. The barrier is used to control the fluid flow from the respective reservoirs into an external medium. The respective flows, which are modeled by functions $J(W^+)$ and $J(W^-)$, at time-instant t are modulated by the position of the sliding barrier $X(t)$ and the level of fluid in the external reservoir $m(t)$. In this reservoir model, the synaptic weight is stored as $W_d = \frac{1}{2}(W^+ - W^-)$ whereas $W_c = \frac{1}{2}(W^+ + W^-)$ serves as an indicator of synaptic usage with respect to time.

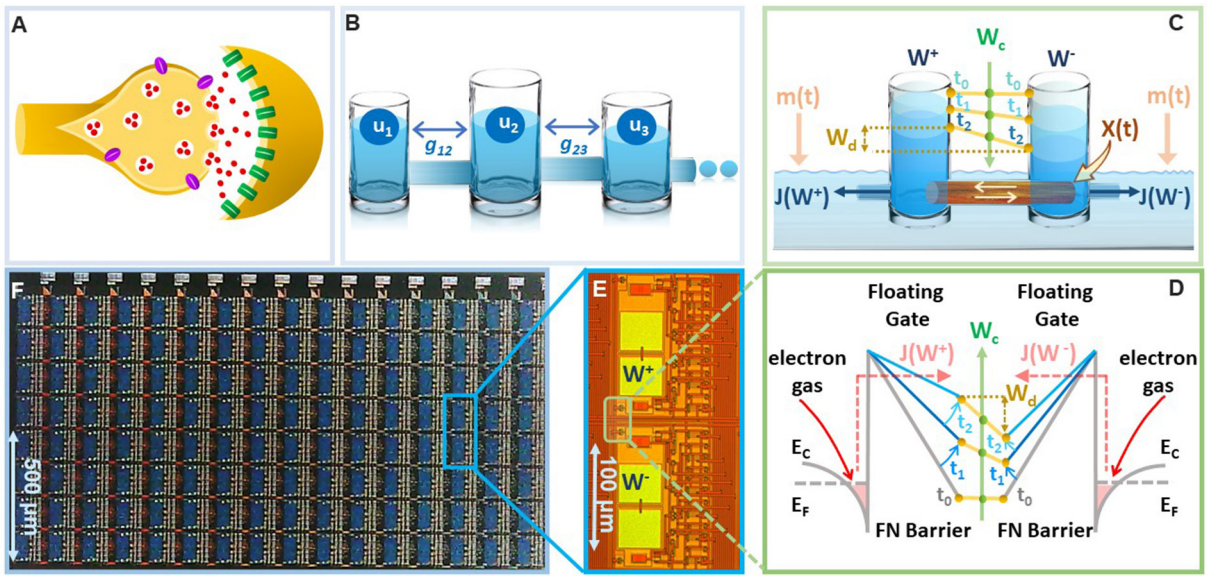


FIGURE 1
On-device memory consolidation using FN-synapses: **(A)** An illustration of a biological synapse with different coupled biochemical processes that determine synaptic dynamics **(B)** physical realization of the cascade model reported in Benna and Fusi (2016) that captures the consolidation dynamics using fluid in reservoirs u_k that are coupled through parameters g_{kj}. **(C)** Illustration of the FN-synapse dynamics using a differential reservoir model and its state at time-instants t_0, t_1, and t_2; **(D)** energy-band diagram to show the implementation of the reservoir model in **(C)** using the physics of Fowler-Nordheim quantum-mechanical tunneling where a single synaptic element (as show in **E**) which stores the weight W_d as the differential charge stored between each tunneling junction, i.e., $W_d = \frac{W^+ - W^-}{2}$ and the common-mode tunneling voltage W_c as the average of the individual charges, i.e., $W_c = \frac{W^+ + W^-}{2}$); **(E)** micrograph of a single FN-synapse; **(F)** micrograph of an array of FN-synaptic devices fabricated in a standard silicon process.

In the Section 3, we show that for a synapse based on a general differential reservoir model [without making assumptions on the nature of the flow function $J(.)$] the synaptic weight W_d evolves in response to the external input $X(t)$ according to the coupled differential equation

$$\frac{dW_d}{dt} = -r(t)W_d + X(t) \tag{1}$$

where

$$r(t) = \frac{d^2W_c}{dt^2}\left(\frac{dW_c}{dt}\right)^{-1} \tag{2}$$

is a time varying decay function that models the dynamics of the synaptic plasticity as a function of the history of synaptic activity (or its usage). The usage parameter W_c evolves according to

$$\frac{dW_c}{dt} = -J(W_c) + m(t) \tag{3}$$

based on the functions $J(.)$ and $m(t)$. Equations (1)–(3) show that the weight W_d update does not directly depend on the non-linear function $J(.)$ but implicitly through the common-mode W_c. Furthermore, Equation (1) conforms to the weight update equation reported in the EWC model (Kirkpatrick et al., 2017) where it has been shown that if $r(t)$ varies according to the network Fisher information metric, then the strength of a stored pattern or memory (typically defined in terms of signal-to-noise ratio) decays at an optimal rate of $1/\sqrt{t}$ when the synaptic network is subjected to random, uncorrelated memory patterns. In Section 3, we show that if the objective is to maximize the operational lifetime of the synapse, then equating the time-evolution profile in Equation (2) to $r(t) \approx \mathcal{O}(1/t)$ (Kirkpatrick et al., 2017) leads to an optimal $J(.)$ of the form $J(V) \propto V^2 \exp(-\beta/V)$ where β is a constant. The expression for $J(V)$ matches the expression for a Fowler-Nordheim (FN) quantum-mechanical tunneling current (Lenzlinger and Snow, 1969) indicating that optimal synaptic memory consolidation could be achieved on a physical device operating on the physics of FN quantum-tunneling.

To verify on-device optimal consolidation dynamics we fabricated an array of FN-synapses and Figures 1D, E show the micrograph of the fabricated prototype. In the Section 3, we show the mapping of the differential reservoir model using the physical variables associated with FN quantum tunneling and Figure 1F shows the mapping using an energy-band diagram. Similar to our previous works (Zhou and Chakrabartty, 2017; Zhou et al., 2019; Rahman et al., 2022), the tunneling junctions have been implemented using polysilicon, silicon-di-oxide, and n-well layers, where the silicon-di-oxide forms the FN-tunneling barrier for electrons to leak out from the n-well onto a polysilicon layer. The polysilicon layer forms a floating-gate where the initial charge can be programmed using a combination of hot-electron injection or quantum-tunneling (Mehta et al., 2020, 2022). The synaptic weight is stored as a differential voltage $W_d = \frac{1}{2}(W^+ - W^-)$ across two floating-gates as shown in Figure 1F. The voltages on the floating-gates W^+ and W^- at any instant of time are modified by the differential signals $\pm\frac{1}{2}X(t)$ that are coupled onto the floating-gates. The dynamics for updating W^+ and W^- are determined by the respective tunneling currents $J(.)$ which discharge the floating-gates. In the Supplementary Figure 1, we describe the complete equivalent circuit for the FN-synapse along with the read-out mechanism used in this work to measure W_d. The presence of additional coupling capacitors in Supplementary Figure 1 provides a mechanism to inject a common-mode modulation signal $m(t)$ into the FN-synapse. We will show in the Section 2 that $m(t)$ can be used to tune the memory consolidation characteristics of the FN-synapse array to achieve memory capacity similar to or better than the cascade consolidation models (with different degrees of complexities) or the task-specific synaptic consolidation corresponding to the EWC model.

2. Results

2.1. FN-synapse characterization

The first set of experiments were designed to understand the *metaplasticity* exhibited by FN-synapses and how the synaptic weight and usage change in response to an external stimulation. The charge stored on the floating-gates in the FN-synapse were first initialized according to the procedure described in the Section 3 and in the Supplementary material. The tunneling barrier thickness in FN-synapse prototype shown in Figures 1D, E was chosen to be greater than 12 nm which makes the probability of direct-tunneling of electrons across the barrier to be negligible. The probability of FN-tunneling of electrons across the barrier (as shown in Figure 1F) is reduced to be negligible by lowering the electric potential of the tunneling nodes W^+ and W^- (see Supplementary Figure 1) with respect to the reference ground to be less than 5 V. In this state the FN-synapse behaves as a standard non-volatile memory storing a weight proportional to $W_d = W^+ - W^-$. To increase the magnitude of the stored weight a differential input pulse $\pm\frac{1}{2}X$ is applied across the capacitors that are coupled to the floating-gates (see Supplementary Figure 1). The electric potential of the floating-gate W^- is increased beyond 7.5 V where the FN-tunneling current $J(W^-)$ is significant. At the same time the electric potential of the floating-gate W^+ is also pushed higher but $W^- > W^+$ such that the FN-tunneling currents $J(W^+) < J(W^-)$. As a result, the W^- node discharges at a rate that is faster than the W^+ node. After the input pulse is removed, the potential of both W^- and W^+ are pulled below 5 V and hence the FN-synapse returns to its non-volatile state. Figures 2A–C

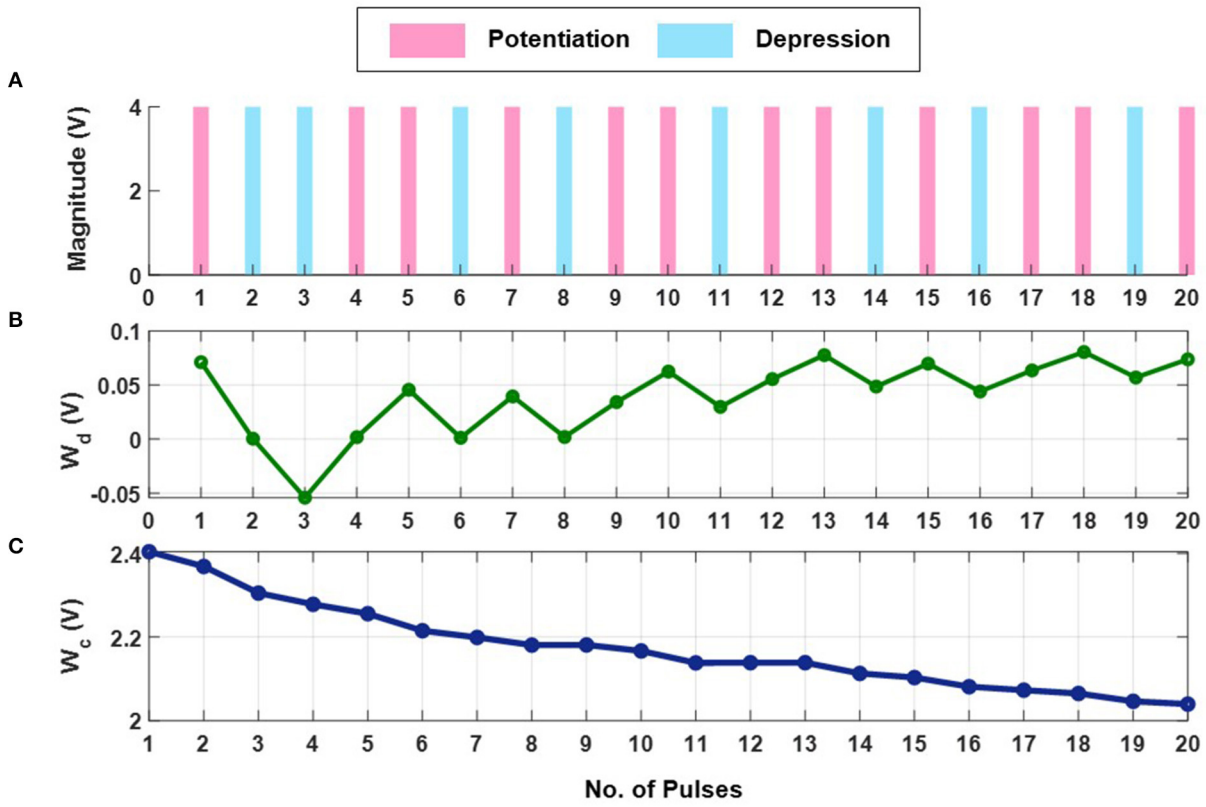


FIGURE 2
Experimental weight evolution of FN-synapse: **(A)** A random set of *potentiation* and *depression* pulses of equal magnitude and duration applied to the FN-synapse leading to **(B)** bidirectional evolution of weight (W_d) and **(C)** the corresponding trajectory followed by the common-mode tunneling node (W_c).

show the measured responses which shows that an FN-synapse can store both the weight and the usage history. When a series of *potentiation* and *depression* pulses of equal magnitude and duration is applied to the FN-synapse, as shown in Figure 2A, the weight stored W_d evolves bidirectionally (like a random walk) due to the input pulses (see Figure 2B). Meanwhile, the common-mode potential W_c decreases monotonically with the number of input pulses irrespective of the polarity of the input, as shown in Figure 2C. Therefore, W_c reliably tracks the usage history of the FN-synapse whereas W_d stores the weight of the synapse. Figures 3A, B show the measured weight update ΔW_d in response to different magnitudes and duration of the input pulses. For this experiment the common-mode $W_c = \frac{1}{2}(W^+ + W^-)$ is held fixed. In Figure 3A, we can observe that for a fixed magnitude of input voltage pulses (= 4V) ΔW_d changes linearly with pulse width. Whereas, Figure 3B shows that the updated ΔW_d changes exponentially with respect to the magnitude of the input pulses (duration = 100 ms). Thus, the results show that pulse width modulation or pulse density modulation provides a more accurate control over the synaptic updates. Furthermore, in regard to energy dissipation per synaptic update pulse width modulation is also more attractive than using pulse magnitude variation. The energy required to write each time on FN-synapse can be estimated by measuring the energy drawn from the differential input source X in Supplementary Figure 1 to charge the coupling capacitor C_c and is given by

$$E_{write} = \frac{1}{2} C_c (X)^2 \tag{4}$$

This means that using smaller pulse magnitude accompanied by longer pulse width is preferable than the other way around in the context of write energy dissipation for the same desired change in weight. However, this would come at a cost of slower writing speed. Therefore, a trade-off exists. For the fabricated FN-synapse prototype, the magnitude of the coupling capacitor C_c is approximately 200f F which leads to 400f J for an input voltage pulse change of 2V across C_c. For the differential input voltage pulse of 4V a total of 800f J of energy was dissipated for each potentiation and depression of the synaptic weights. When the common-mode W_c is not held fixed, irrespective of whether the weight W_d is increased or decreased (depending on the polarity of the input signal) the common-mode always decreases. Thus,

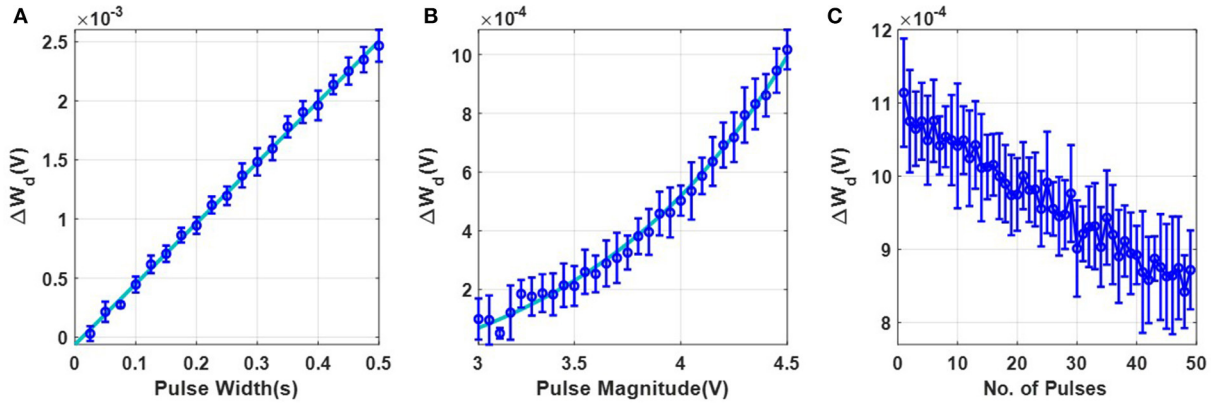


FIGURE 3
Experimental characterization of a single FN-synapse: **(A)** Dependence of change in magnitude of weight with change in pulse-width which follows a linear trajectory defined by $y = mx + c$ (where $m = 0.005136$ and $c = -6.227 \times 10^{-5}$). **(B)** Dependence on pulse magnitude of the input pulse which follows an exponential trajectory defined by $y = c \times exp(ax + b) + d$ (where $a = 1$, $b = -6.611$, $c = 0.009959$ and $d = -0.0002142$). **(C)** Change in the magnitude of successive weight updates (ΔW_d) corresponding to repeated stimulus.

W_c serve as an indicator of the usage of the synapse. Figure 3C shows the *metaplasticity* exhibited by an FN-synapse where we measured ΔW_d as a function of usage by applying successive *potentiation* input pulses of constant magnitude (4 V) and width (100 ms). Figure 3C shows that when the synapse is modulated with same excitation successively, the amount of weight update decreases monotonically with increasing usage, similar to the response illustrated in Figures 1C, F.

2.2. FN-synapse network capacity and memory lifetime without plasticity modulation

The next set of experiments were designed to understand the FN-synaptic memory consolidation characteristics when the array is excited using a random binary input pattern (*potentiation* or *depression* pulses). This type of benchmark experiment is used extensively in memory consolidation studies (Benna and Fusi, 2016; Kirkpatrick et al., 2017) since analytical solutions exist for limiting cases which can be used to validate and compare the experimental results. A network comprising of N FN-synapses is first initialized to store zero weights (or equivalently $W^- = W^+$). New memories were presented as random binary patterns (N dimensional random binary vector) that are applied to the N FN-synapses through either *potentiation* or *depression* pulses. Each synaptic element was provided with balanced input, i.e., equal number of *potentiation* and *depression* pulses. The goal of this experiment is to track the strength of a memory that is imprinted on this array in the presence of repeated new memory patterns. This is illustrated in Figures 4A, B where an initial input pattern (a 2D image of the number "0" comprising of 10×10 pixels) is written on a memory array. The array is then subjected to images of noise patterns that are statistically uncorrelated to the initial input pattern. It can be envisioned that as additional new patterns are written to the same array, the strength of a specific memory (of the image "0") will degrade. Similar to the previous studies (Benna and Fusi, 2016; Kirkpatrick et al., 2017) we quantify this degradation in terms of signal-to-noise ratio (SNR). If n denotes the number of new memory patterns that have been applied to an empty FN-synapse array (initial weight stored on the network is zero), then the Section 3 shows that for the p^{th} update the retrieval memory signal $S(n,p)$ power, the noise $\nu(n,p)$ power and the $SNR(n,p)$ can be expressed analytically as

$$S^2(n,p) = \frac{1}{(n+\gamma)^2}; \quad \nu^2(n,p) = \frac{n}{N(n+\gamma)^2};$$
$$SNR(n,p) = \sqrt{\frac{N}{n}}. \tag{5}$$

where $\gamma > 0$ is a device parameter that depends on the initialization condition, material properties and duration of the input stimuli.

Equation (5) shows that the initial SNR is $\sqrt{N}$ and the SNR falls off according to a power-law decay with a slope of $\frac{1}{\sqrt{n}}$. Like previous consolidation studies (Benna and Fusi, 2016) we will assume that a specific memory pattern is retained as long as its SNR exceeds a predetermined threshold (unity in this experiment). Therefore, according to Equation (5) the network capacity and memory lifetime for FN-synapse scales linearly with the size of the network N when the initial weight across all synapse is zero. We verified the analytical expressions in Equation (5) for a network size of $N = 100$ using results measured from the FN-synapse chipset. Details of the

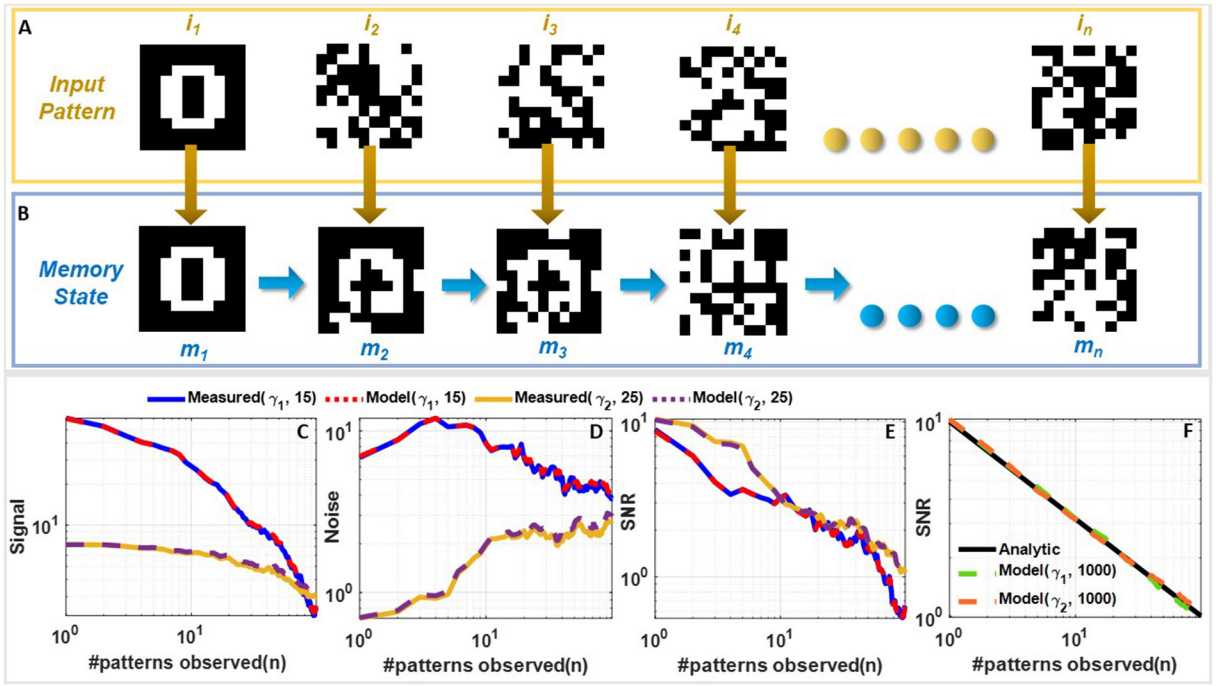


FIGURE 4
Comparison of measured and simulated memory consolidation for an empty FN-synapse network: **(A)** Set of 10×10 randomized noise inputs fed to a network of 100 FN-synapses initialized to store an image of the number 0 and **(B)** the corresponding memory evolution. Comparison of **(C)** signal strength, **(D)** noise strength, and **(E)** SNR for a network size of 100 synapse measured using the fabricated FN-synapse array shown in Figure 1F for 25 (for γ_1) and 15 (for γ_2) Monte-Carlo runs. **(F)** SNR comparison of the γ_1 and γ_2 models with the analytical model for 1,000 Monte Carlo simulations. The legends associated with the plots are specified as (γ, Number of Monte-Carlo runs). All of these results correspond to the behavior of an empty FN-synapse network.

hardware experiment is provided in the Section 3. Figures 4C–E show the retrieval signal, noise, and SNR obtained from the fabricated FN-synapse network for two different values of γ. We observe that the SNR obtained from the hardware results conform to the analytical expressions relatively well. The slight differences can be attributed to the Monte-Carlo simulation artifacts (only 25 and 15 iterations were carried out). In the Supplementary Figure 3, we show verification of these analytic expressions using a behavioral model of the FN-synapse which mimics the hardware prototype with great accuracy (as shown in Supplementary Figure 2). Details on the derivation of FN-synapse model is provided in the Section 3. The simulated results in Figures 4C–E verifies that results from the software model can accurately track the hardware FN-synapse measurements for both values of γ when subjected to the same stimuli. Therefore, FN-synapse and its behavioral model can be used interchangeably. The results in Figure 4F also show that when the number of iterations on the Monte-Carlo simulation is increased (1,000 iterations), the simulated SNR closely approximates the analytic expression. This verifies that hardware FN-synapse is also capable of exactly matching the optimal analytic consolidation characteristics. Figure 3C shows the measured evolution of weights stored in the FN-synapse where initially the weights grow quickly but after a certain number of updates settle to a steady value irrespective of new updates. This implies that the synapses have become rigid with an increase in its usage. This type of memory consolidation is also observed in EWC models which has been used for continual learning. However, note that unlike EWC models that need to store and update some measure of Fisher information, whereas, here the physics of the FN-synapse device itself can achieve similar memory consolidation without any additional computation.

2.3. Plasticity modulation of FN-synapse models

In our next set of experiments, we verified that the plasticity of FN-synapses can be adjusted to mimic the consolidation properties of both EWC and steady-state models (such as cascade models). While the EWC model only allows for the retention of old memories, steady state/cascade models allow for both memory retention and forgetting. As a result, these models avoid *blackout catastrophe* whereas an EWC network is unable to retrieve any previous memories or store new

experiences as the network approaches its capacity. Steady-state models allow the network to gracefully forget old memories and continue to remember new experiences *indefinitely*. For an FN-synapse network, a coupling capacitor in each synapse (shown in Supplementary Figure 1) which is driven by a global voltage signal $V_{mod}(t)$ (which produces $m(t) = \frac{dV_{mod}(t)}{dt}$) can control the plasticity of the FN-synapse to mimic the characteristics of a steady-state model. Details of the modified FN-synapse achieving a steady-state response are provided in the Section 3. To understand and compare the blackout catastrophe in FN-synapse models with a steady-state model, e.g., the cascade model we define the metric *#patterns.retained* as the total number of memory patterns whose SNR exceeds 1 at any given point of time. The *#patterns.retained* for FN-synapse network with modulation profiles $m_0(t)$, $m_1(t)$, $m_2(t)$, $m_3(t)$, and $m_4(t)$ of size $N = 1,000$ is shown in Figure 5A together with those for cascade models of different levels of complexity (Benna and Fusi, 2016) (denoted by $c = 1,..,5$). In order to calculate the *#patterns.retained* the SNR resulting from each stimulus was calculated and tracked at every observation to determine the number of such stimuli that had a corresponding SNR greater than unity. The profiles of $m_1(t)$, $m_2(t)$, and $m_3(t)$ are produced by changing $V_{mod}(t)$ at each update as three quarter, half, and quarter of the average of ΔW_d across all the synapses during the latest update, respectively, while $m_0(t)$ is achieved through a constant voltage signal $V_{mod}(t)$. We can observe in Figure 5A that the FN-synapse network with $m_0(t)$ forgets all observed patterns in addition to not forming any new memories as *#patterns.retained* goes to zero as the network capacity is reached starting from an empty network. Whereas, in the case for FN-synapse under $m_1(t)$ and $m_2(t)$ modulation profile the *#patterns.retained* reaches a finite value similar to that of the cascade models. This indicates that the FN-synapse network when subjected to plasticity modulation profiles continues to form new memory while gracefully forgetting the old ones. For the $m_3(t)$ modulation profile the network is slowly evolving and yet to reach the steady state condition within 2000^{th} update. The FN-synapse network under the $m_4(t)$ modulation profile, which switches between $m_0(t)$ and $m_1(t)$ periodically, is in an oscillatory steady-state with the same periodicity as the modulation profile itself. However, note that the network does not suffer from blackout catastrophe and has a variable capacity. This shows that the capacity of the FN-synapse network can also be tuned to the specificity of different applications. From the figure, we also observe that the steady state network capacity for $m_2(t)$ modulation profile is higher than that of cascade models. Note here that network capacity for cascade models may be increased by increasing the complexities of the synaptic model. Nevertheless, we find that network capacity for FN-synapse is comparable to cascade models of moderate complexities.

In order to understand the plasticity modulation further, we investigated the SNR for patterns introduced to a non-empty network. For this experiment, we tracked the 1000^{th} pattern observed by the network of $N = 1,000$ synapse. Figure 5B shows the SNR of this pattern under $m_1(t) - m_4(t)$ modulation profile along with cascade models of various complexity. Note that the x-axis now represents the age of the stimulus, i.e., number of patterns observed after the tracked pattern. For the modulation profile $m_1(t)$ the initial SNR is large, comparable to that of cascade models, but the SNR falls off quickly indicating high plasticity. Whereas, for modulation profile $m_2(t)$ and $m_3(t)$ the initial SNR is smaller than $m_1(t)$ but it falls off at a much later time similar to cascade models with high complexities. These SNR profiles for FN-synapse model with modulation $m_1(t) - m_3(t)$ are similar to that of a constant weight decay synaptic model used in deep learning neural network as a regularization method. On the other hand, the SNR profile for the 1000^{th} pattern under $m_4(t)$ modulation has both high initial SNR and a large lifetime. However, from Figure 5B, we observe that the network is in an oscillatory state which indicates that this profile is specific to the 1000^{th} pattern, and if we tracked any other pattern the SNR profile would be different (for reference the SNR tracked for the 750^{th} update is also shown). This is not the case for the cascade models which would consistently have similar SNR profiles irrespective of the pattern that is tracked. Nevertheless, this SNR profile for the FN-synapse model would repeat itself corresponding to the periodicity of the modulation profile. This suggests that the amount of plasticity and memory lifetime for the FN-synapse model is readily tunable and depends on the amount of modulation provided to the network. We have also verified that the synaptic strength of FN-synapse is bounded similarly to that of the cascade models. This can be observed in Supplementary Figure 10 which shows that the variance in retrieval signal (Noise) of an FN-synapse network with both constant modulation and time-varying modulations remains bounded. Furthermore, Supplementary Figure 11 shows that plasticity modulation indeed introduces a forgetting mechanism as the SNR for different modulation profiles (when tracked from an empty network) starts to fall off earlier than the one without modulation. In addition to different modulation profile, the plasticity-lifetime tradeoff of the FN-synapse model can also be achieved by varying the parameter γ as shown in Supplementary Figure 12. Therefore, our synaptic models can exhibit memory consolidation properties similar to both EWC and steady-state models while being physically realizable and scalable for large networks.

2.4. Continual learning using FN-synapse

The next set of experiments was designed to evaluate the performance of FN-synapse neural network for a benchmark continual learning task. A fully-connected neural network with two hidden layers was trained sequentially on multiple supervised learning tasks. Details of the neural network architecture and training are given in Section 3 and in the

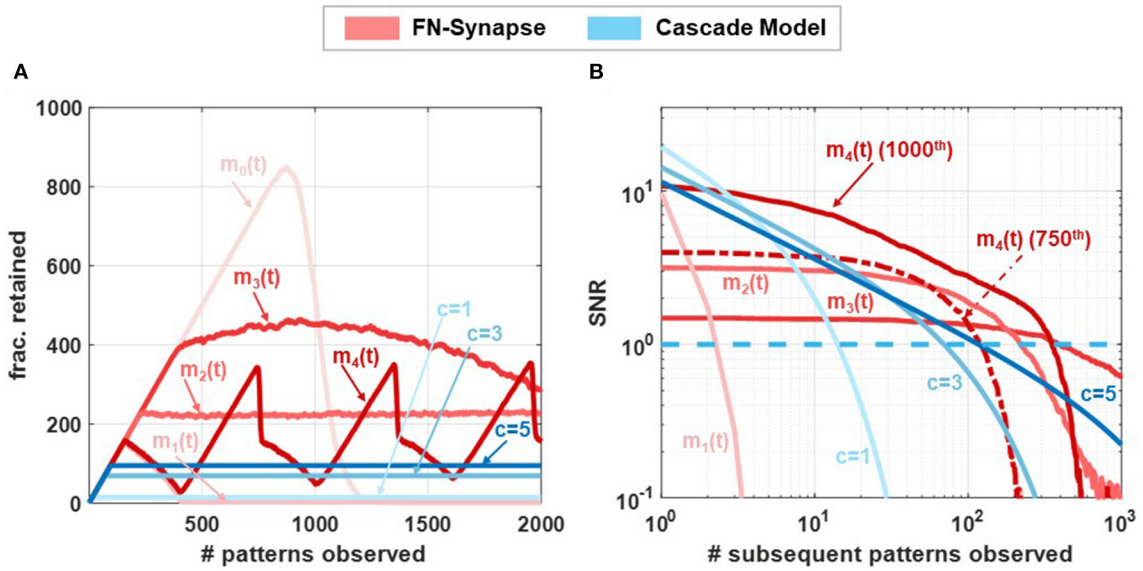


FIGURE 5
Network capacity and saturation experiments: Comparison of **(A)** no. of patterns retained by networks composed of 1,000 synapses following different synaptic models when exposed to 2,000 patterns and **(B)** steady-state SNR of the 1000th update ($p = 1,000$) of networks consisting of 1,000 synapses with various synaptic models when exposed to subsequent updates. For $m_{4(t)}$ modulation SNR profiles for both 450th and 1000th ($p = 450,1000$) updates are shown.

Supplementary material. The network was trained on each task for a fixed number of epochs and after the completion of its training on a particular task t_n, the dataset from t_n was not used for the successive task t_{n+1}.

The aforementioned tasks were constructed from the Modified National Institute of Standards and Technology (MNIST) dataset, to address the problem of classifying handwritten digits in accordance with schemes popularly used in several continual-learning literature (Hsu et al., 2018). Also known as incremental domain learning using split-MNIST dataset, each task of this continual learning benchmark dictates the neural network to be trained as binary classifier which distinguishes between a set of two hand-written digits, i.e., the network is first trained to distinguish between the set $[0, 1]$ as t_1 and is then trained to distinguish between $[2, 3]$ in t_2, $[4, 5]$ in t_3, $[6, 7]$ in t_4, and $[8, 9]$ in t_5. Thus, the network acts as an even-odd number classifier during every task.

Supplementary Figures 7A–E compare the task-wise accuracy of networks trained with different learning and consolidation approaches. Note here that the absence of a data-point corresponding to a particular approach indicates that the accuracy obtained is below 50%. All the approaches taken into consideration perform equally well at learning t_1 as illustrated in Supplementary Figure 7A. However, as the networks learn t_2 (see Supplementary Figure 7B), the performance of both EWC (Kirkpatrick et al., 2017) and online EWC (Liu et al., 2018) degrade for task t_1 as do the networks with conventional memory using SGD and ADAM. The FN-synapse based networks on the other hand retain the accuracy of task t_1 far better in comparison. This advantage in retention comes at the cost of learning t_2 marginally poorer than others. This trend of retaining the older memories or tasks far better than other approaches continues in successive tasks. Particularly, if we consider the retention of t_1 when the networks are trained on t_3 (see Supplementary Figure 7C), it can be observed that it is only the FN-synapse based networks that retain t_1 while others fall below the 50% threshold. Similar trends can be observed in Supplementary Figures 7D, E. There are a few instances during the five tasks where the EWC variants and SGD with conventional memory marginally outperform or match the FN-synapse in terms of retention. However, if the overall average accuracy of all these approaches are compared (see Figure 6A), it is clearly evident that both the FN-synapse networks significantly outperform the others. It is also worth noting here that even when a network equipped with FN-synapse is trained using a computationally-inexpensive optimizer such as SGD, it shows remarkably superior performance than highly computationally-expensive approaches such as ADAM with conventional memory and ADAM with EWC variants.

The only drawback of the FN-synapse based approach is that its ability to learn the present task slightly degrades with every new task. This phenomenon results from the FN-synapses becoming more rigid and can be seen in Figure 6B which shows the evolution of plasticity of weights in the output and input layer of the network with successive tasks with respect to W_c. As mentioned earlier, W_c keeps track of the importance of

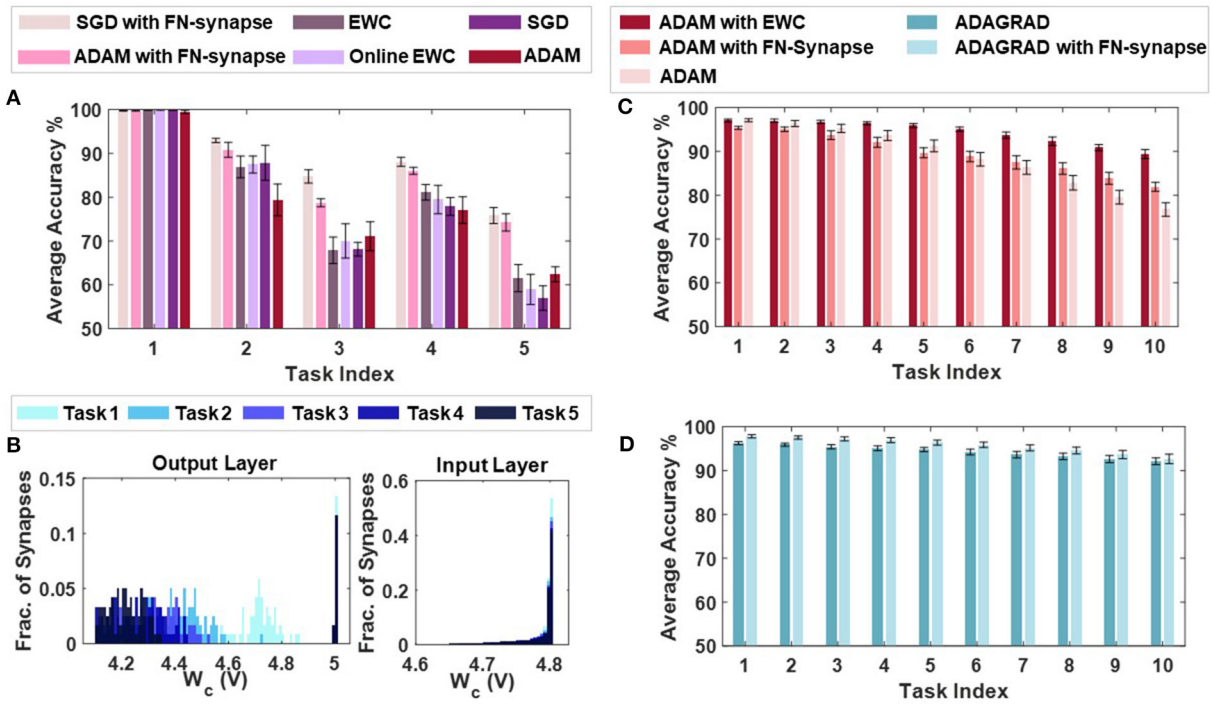


FIGURE 6
Continual learning benchmarks results and insights: **(A)** Overall average accuracy comparison of SGD and ADAM with FN-synapse, ADAM with EWC and Online EWC, SGD, and ADAM with conventional memory. **(B)** Distribution of the usage profile of weights in the output layer and the input layer of the FN-synapse neural network. Overall Average Accuracy comparison of incremental-domain learning scenarios on the Permuted MNIST dataset using **(C)** ADAM with EWC, ADAM with FN-Synapse and ADAM with conventional memory and **(D)** ADAGRAD with conventional memory and ADAGRAD with FN-synapse.

each weight as a function of the number of times it is used. The higher the W_c of a particular weight, the less it has been used and therefore, the more plastic it is and sensitive to change. On the other hand, a more rigid and frequently used weight has a lower value of W_c. Suppose the output layer is considered from Figure 6B. In that case, it can be observed that with each successive task the W_c of the weights of the network collectively reduces, leading to more consolidation and consequently leaving the network with fewer plastic synapses to learn a new task. In comparison, the majority of the weights in the input layer remain relatively more plastic (or less spread out) owing to the redundancies in the network arising from the vanishing gradient problem (see Section 4 for more details). In Supplementary Figure 5, we show that the ability of the network to learn or forget new tasks is a function of the initial plasticity of the FN-synapses and can be readily adjusted.

In addition to the split-MNIST benchmark, the performance of FN-synapse based network was compared with EWC for the permuted MNIST benchmark. These incremental-domain learning experiments were carried out by randomly permuting the order of pixels of the images in the MNIST dataset in accordance with Hsu et al. (2018) to create new tasks. The overall average accuracy for 10 Monte Carlo simulations when using ADAM as the optimizer with EWC, FN-Synapse and conventional memory are depicted in Figure 6C. We can observe from Figure 6C that despite not being as retentive as EWC in this particular scenario, the network equipped with FN-synapse as the memory element performs better than the network without any memory consolidation mechanism, thereby exhibiting continual learning ability. Furthermore, when compared to a network with traditional memory employing an optimizer like ADAGRAD, which has been shown to be suitable for this learning scenario (Hsu et al., 2018), the FN-synapse network with ADAGRAD exhibits marginal improvements without any drop in performance with respect to the former as shown in Figure 6D.

3. Materials and methods

The main methods are described in this section of the paper while Supplementary material includes additional details, supporting information, and figures.

3.1. Weight update for differential synaptic model

Consider the differential synaptic model described by Figure 1C where the evolution of two dynamical systems with

state variables W^+ and W^- is governed by

$$\frac{dW^+}{dt} = -J(W^+) + \frac{1}{2}X(t) + \frac{1}{2}m(t) \tag{6}$$

$$\frac{dW^-}{dt} = -J(W^-) - \frac{1}{2}X(t) + \frac{1}{2}m(t) \tag{7}$$

where $J(.)$ is an arbitrary function of the state variables, $+\frac{1}{2}X(t)$ or $-\frac{1}{2}X(t)$ are differential time varying inputs and $m(t)$ is a common mode modulation input. In this differential architecture, we define the weight parameter W_d as $W_d = \frac{1}{2}(W^+ - W^-)$ which represents the memory and the common-mode parameter W_c as $W_c = \frac{1}{2}(W^+ + W^-)$ which represents the usage of the synapse. Applying this definition to (6) and (7), we obtain:

$$\frac{d(W_c + W_d)}{dt} = -J(W_c + W_d) + \frac{1}{2}X(t) + \frac{1}{2}m(t) \tag{8}$$

$$\frac{d(W_c - W_d)}{dt} = -J(W_c - W_d) - \frac{1}{2}X(t) + \frac{1}{2}m(t) \tag{9}$$

Now, adding and subtracting (8) and (9), we get:

$$\frac{dW_c}{dt} = -\left(\frac{J(W_c + W_d) + J(W_c - W_d)}{2}\right) + m(t) \tag{10}$$

$$\frac{dW_d}{dt} = -\left(\frac{J(W_c + W_d) - J(W_c - W_d)}{2}\right) + X(t) \tag{11}$$

Assuming that $W_c >> W_d$, applying Taylor series expansion on (10) and (11) leads to

$$\frac{dW_c}{dt} = -J\left(W_c\right) + m(t) \tag{12}$$

$$\frac{dW_d}{dt} = -J'\left(W_c\right) W_d + X(t). \tag{13}$$

This means that the modulation input impacts the usage of the synapse. Therefore, the plasticity of the synapse can be *tuned* using $m(t)$ when needed. Now we first look into the trivial case when a constant modulation input is provided, i.e., $m(t) = c$ where c is any arbitrary constant. In this scenario the plasticity of the synapse is solely dependent on the usage of the synapse as $m(t)$ does not change with time. Substituting the derivative of W_c from (12), when $m(t)$ is constant, into (13), the rate of change in W_d can be formulated as:

$$\frac{dW_d}{dt} = -\left[\frac{d^2W_c}{dt^2}\left(\frac{dW_c}{dt}\right)^{-1}\right] W_d + X(t) \tag{14}$$

Please refer to the Supplementary material for detailed derivation. Equation (14) shows that the change in weight ΔW_d is directly proportional to the *curvature* of usage while being inversely proportional to the rate of usage.

3.2. Optimal usage profile

We define the decaying term in (14) as

$$r(t) = -\left[\frac{d^2W_c}{dt^2}\left(\frac{dW_c}{dt}\right)^{-1}\right] \tag{15}$$

Now, comparing the weight update equation in (14) to the weight update equation for EWC in the balanced input scenario, the decay term has the following dependency with time for avoiding catastrophic forgetting.

$$r(t) = O\left(\frac{1}{t}\right) \tag{16}$$

Now, the usage of a synapse is always monotonically increasing and since W_c represents the usage, it too needs to monotonic. At the same time W_c also needs to be bounded, therefore W_c has to monotonically decrease with increasing usage while satisfying the relationship in Equation (16). It can be shown that Equations (16) and (15) can be satisfied by any dynamical system of the form

$$W_c = \frac{1}{f(\log t)} \tag{17}$$

where $f(.) \geq 0$ is any monotonic function. Substituting Equation (17) in Equation (15) we obtain the corresponding usage profile as follows

$$r(t) = \frac{1}{t}\left(1 + \frac{2f'(\log t)}{\log t} - \frac{f''(\log t)}{f'(\log t)}\right) \tag{18}$$

where $f'(\log t)$ and $f''(\log t)$ are derivatives of $f(\log t)$ with respect to $\log t$. While several choices of $f(.)$ are possible, the simplest usage profile can be expressed as

$$W_c = \frac{\beta}{\log(t)} \tag{19}$$

where β is any arbitrary constant. The corresponding non-linear function in this model is determined by substituting Equation (19) in Equation (12) to obtain

$$J\left(W_c\right) = \frac{1}{\beta}W_c^2 \exp\left(-\frac{\beta}{W_c}\right). \tag{20}$$

The expression for $J(.)$ in Equation (20) bears similarity with the form of FN quantum-tunneling current (Lenzlinger and Snow, 1969) and Figures 1D–F show the realization of Equations (6) and (7) using FN tunneling junctions.

3.3. Achieving optimal usage profile on FN-synapse

For the differential FN tunneling junctions shown in Figure 1F and its equivalent circuit shown in the Supplementary Figure 1, the dynamical systems model is given by

$$C_T \frac{dW^+}{dt} = -J(W^+) + \frac{C_c}{2}\frac{dv_{in}}{dt} \tag{21}$$

$$C_T \frac{dW^-}{dt} = -J(W^-) - \frac{C_c}{2}\frac{dv_{in}}{dt} \tag{22}$$

where W^+, W^- are the tunneling junction potentials, C_c is the input coupling capacitance, $v_{in}(t)$ is the input voltage to the coupling capacitance and $C_T = C_c + C_{fg}$ is the total capacitance comprising of the coupling capacitance and the floating-gate capacitance C_{fg}. $J(.)$ are the FN tunneling currents given by

$$J\left(W^+\right) = \left(\frac{k_1}{k_2}\right)\left(W^+\right)^2 \exp\left(-\frac{k_2}{W^+}\right) \tag{23}$$

$$J\left(W^-\right) = \left(\frac{k_1}{k_2}\right)\left(W^-\right)^2 \exp\left(-\frac{k_2}{W^-}\right) \tag{24}$$

where k_1 and k_2 are device specific and fabrication specific parameters that remain relatively constant under isothermal conditions. Following the derivations in the previous sections and the expression in Equation (19) leads to a common-mode voltage W_c profile as

$$W_c(t) = \frac{k_2}{\log(k_1 t + k_0)} \tag{25}$$

where $k_0 = \exp\left(\frac{k_2}{W_{c0}}\right)$ and W_{c0} refers to the initial voltage at the floating-gate.

3.4. FN-synpase network SNR estimation for random pattern experiment

Upon following the same procedure used in previous sections, the weight update equation for an FN-synapse using Equation (21) and Equation (22) can be expressed as

$$C_T \frac{dW_d}{dt} = -\left[\frac{d^2 W_c}{dt^2}\left(\frac{dW_c}{dt}\right)^{-1}\right] W_d + C_c \frac{dv_{in}}{dt} \tag{26}$$

We designed the floating-gate potential and the input voltage pulses such that the FN-dynamics is only active when there is an memory update. Therefore, the dynamics in Equation (26) evolve in a discrete manner with respect to the number of modulations. Assuming $C_T = C_c$ we formulate a discretized version of the weight update dynamics from Equation (26) in accordance with the floating-gate potential profile of the device expressed in Equation (25) as follows

$$\frac{\triangle W_d(n)}{\triangle t} = -k_1\left(1 + \frac{2}{\log\left(k_1 \triangle tn + k_0\right)}\right)\left(\frac{1}{k_1 \triangle tn + k_0}\right) W_d(n-1) + \frac{\triangle v_{in}(n)}{\triangle t} \tag{27}$$

$$W_d(n) = \left[1 - \left(1 + \frac{2}{\log\left(k_1 \triangle tn + k_0\right)}\right)\left(\frac{1}{n + \frac{k_0}{k_1 \triangle t}}\right)\right] W_d(n-1) + \left(v_{in}(n) - v_{in}(n-1)\right) \tag{28}$$

where n represents the number of patterns observed and Δt is the duration of the input pulse. Let us denote the weight decay term as

$$\alpha(n) = \left[1 - \left(1 + \frac{2}{\log\left(k_1 \triangle tn + k_0\right)}\right)\left(\frac{1}{n + \frac{k_0}{k_1 \triangle t}}\right)\right] \tag{29}$$

Thus, we obtain the weight update equation with respect to number of patterns observed as

$$W_d(n) = \alpha(n) W_d(n-1) + \left(v_{in}(n) - v_{in}(n-1)\right) \tag{30}$$

When we start from an empty network, i.e., $W_d(0) = 0$, the memory update can be expressed as a weighted sum over the past input as

$$W_d(n) = \sum_{i=1}^{n-2}\left\{(\alpha(i+1) - 1)\left(\prod_{j=i+2}^{n}\alpha(j)\right) v_{in}(i)\right\} + (\alpha(n) - 1) v_{in}(n-1) + v_{in}(n) \tag{31}$$

We define the retrieval signal and the noise associated with it as per the definition in Benna and Fusi (2016). For a network comprising of N synapses, each weight in the network is indexed as $W_d(a, n)$ where $a = 1, ..., N$. Similarly, the input applied to the a^{th} synapse after n patterns is $v_{in}(a, n)$. Then, the signal strength for the p^{th} update (where $p < n$) introduced to the initially empty network tracked after n patterns can be formulated as:

$$S(n, p) = \frac{1}{N}\left\langle \sum_{a=1}^{N} W_d(a, n) v_{in}(a, p)\right\rangle \tag{32}$$

where angle brackets denote averaging over the ensemble of all of the input patterns seen by the network. If we assume that

the input patterns are random binary events of ±1 and are uncorrelated between different synapses and memory patterns then substituting Equation (31) in Equation (32), we obtain

$$S(n,p) = (\alpha(p+1) - 1) \prod_{j=p+2}^{n} \alpha(j) \tag{33}$$

Given that in Equation (29), $k_0 = \mathcal{O}(10^{19})$ and $k_1 = \mathcal{O}(10^{16})$, the term $\left(1 + \frac{2}{\ln(k_1 \triangle t n + k_0)}\right) \approx 1$, the signal power simplifies to:

$$S^2(n,p) = \frac{1}{(n+\gamma)^2} \tag{34}$$

where $\gamma = \frac{k_0}{k_1 \triangle t}$ and depends on the pulse-width $\triangle t$ and the initial condition k_0. The above equation shows that the signal's strength is a function of the system parameter γ and decays with the number of memory pattern observed. If we assume that the weight $W_d(n)$ is uncorrelated from the input $v_{in}(n)$ and that the inputs $v_{in}(1), v_{in}(2), ...v_{in}(n)$ are uncorrelated from each other, then the corresponding noise power is given by the variance of the retrieval signal expressed in Equation (32). This can be estimated as the sum of the power of all signals tracked at n except for the retrieval signal corresponding to the p^{th} update we are tracking and is given by:

$$\nu^2(n,p) = \frac{1}{N} \sum_{i=1, i\neq p}^{n} S^2(n,i) \tag{35}$$

However, in order to derive a more tractable analytical expression for further analysis we added the retrieval signal as well into the summation which introduces a small error in the estimation (overestimating the noise by the retrieval signal term). This leads us to the following estimation of the noise power:

$$\nu^2(n,p) = \frac{n}{N(n+\gamma)^2} \tag{36}$$

Based on the value of n in comparison to γ, we obtain two trends for the noise profile. When $\gamma >> n$,

$$\nu(n,p) = \frac{1}{\sqrt{N}} \left(\frac{\sqrt{n}}{\gamma}\right) \tag{37}$$

which implies that noise increases with increase in updates initially. On the other hand, when $\gamma << n$,

$$\nu(n,p) = \frac{\sqrt{n}}{\sqrt{N}n} = \frac{1}{\sqrt{N}} \left(\frac{1}{\sqrt{n}}\right) \tag{38}$$

which implies that noise falls with increase in updates in the later stages. The signal-to-noise ratio (SNR) of a network of size N can then be obtained as:

$$SNR(n,p) = \sqrt{\frac{S^2(n,p)}{\nu^2(n,p)}} = \sqrt{\frac{N}{n}} \tag{39}$$

3.5. FN-synapse with tunable consolidation characteristics

In the previous sections, we derived the analytical expressions for the memory retrieval signal, the noise associated with it, and the corresponding SNR for the case when the modulation input $m(t)$ was kept constant. This led to a synaptic memory consolidation which is similar to that of EWC. However, blackout catastrophic forgetting occurs in networks with such memory consolidation due to the absence of a balanced pattern retention and forgetting mechanism. The *forgetting* mechanism is naturally present in a steady state model such as the cascade model which do not suffer from memory "blackouts". Since the increase in *retention* is equivalent to an increase in rigidity and *forgetting* is tantamount to a decrease in rigidity, it is necessary to adjust the plasticity/rigidity of the synapse accordingly. From Figures 2A, B, we notice that without external modulation W_c decreases monotonically with each new updates which correspondingly makes the synapse only rigid. Therefore, to balance the same, the idea is to keep W_c as steady as possible to keep the synapse plastic as long as possible by applying a modulation profile $m(t)$ that *recovers/restores* W_c after every synaptic update. This results in $m(t)$ of the form

$$m(t) = m(i)\delta(t - iT) \tag{40}$$

where $\delta(t)$ is the Dirac-delta, $m(i)$ is the magnitude of the modulation increment, and T is the time between each modulation increment. This increment is determined by the rate of the differential update to the FN-synapse. Integrating this form of $m(t)$ into Equation (12) leads to

$$\frac{dW_c}{dt} = -J(W_c) + m(i)\delta(t - iT) \tag{41}$$

which implies a tunable plasticity profile for the FN-synapse. An analytical solution to the differential equation (41) is difficult and hence we resort to a recursive solution. Due to the nature of the $m(t)$, it can be seen that the initial condition of the variable W_c changes at increments of T, whereas between two modulation increments W_c evolves naturally according to Equation (25). Thus, the dynamics of W_c in the presence of the modulation increments can be described as

$$W_c(t) = \begin{cases} W_{c0} & ; \quad t = 0 \\ W_c(t) + V_{mod}(t) & ; \quad t = iT \\ \frac{k_2}{\log(k_1(t-iT)+\exp(\frac{k_2}{W_c(iT)}))} & ; \; iT < t < (i+1)T \end{cases} \tag{42}$$

where $V_{mod}(t)$ is an external voltage signal applied to the FN-synapse as shown in Supplementary Figure 1 and is given by:

$$V_{mod}(t) = \sum_{i=1}^{\infty} m(i)\delta(t - iT) \tag{43}$$

In this case the change in plasticity of the synapse is determined by the step-size of the staircase voltage function $V_{mod}(t)$. Note that the weight update equation in (13) is still valid since $m(t)$ is kept constant during differential input.

Although an analytic expression for the SNR is no longer tractable in this iterative form, the ability of the modulation term to regulate the plasticity and induce a more graceful form of forgetting is shown in the corresponding no. of patterns retained plot in Figure 5A and the SNR plot Figure 5B for various modulation input profiles.

3.6. Programming and initialization of FN-synapses

The potential corresponding to the tunneling nodes W^+ and W^- can be accessed through a capacitively coupled node, as shown in Supplementary Figure 1. This configuration minimizes readout disturbances and the capacitive coupling also acts as a voltage divider so that the readout voltage is within the input dynamic range of the buffer. The configuration also prevents hot-electron injection of charge into the floating gate during readout operation. Details of initialization and programming are discussed in Mehta et al. (2020), so here we describe the methods specific for this work. The tunneling node potential was initialized at a specific region where FN-tunneling only occurs while there is a voltage pulse at the input node and the rest of the time it behaves as a non-volatile memory. This was achieved by first measuring the readout voltage every 1 s for a period of 5 min to ensure that the floating gate was not discharging naturally. During this period the noise floor of the readout voltage was measured to be $\approx 100\mu V$. At this stage, an voltage pulse of magnitude 1 V and duration 1 ms was applied at the input node and the change in readout voltage was measured. If the change was within the noise floor of the readout voltage, the potential of the tunneling nodes were increased by pumping electrons out of the floating gate using the program tunneling pin. This process involves gradually increasing the voltage at the program tunneling pin to 20.5 V (either from external source or from on-chip charge pump). The voltage at the program tunneling pin was held for a period of 30 s, after which it was set to 0 V. The process was repeated until substantial change in the readout voltage was observed ($\approx 300\mu V$) after providing an input pulse. The readout voltage in this region was around 1.8 V.

3.7. Hardware and software experiments for random pattern updates

The fabricated prototype contained 128 differential FN tunneling junctions, which corresponds to 64 FN-synapses. However, due to the peripheral circuitry only one tunneling node could be accessed at a time for readout and modification. Now, since the memory pattern is completely random, each synapse can be modified independently without affecting the outcome of the experiment. Therefore, two tunneling nodes were initialized following the method described in the aforementioned section. Input pulses of magnitude 4 V and duration 100 ms was applied to both the tunneling nodes. The change in the readout voltages were measured, and the region where the update sizes of both the tunneling node would be equal was chosen as the initial zero memory point for the rest of the experiment. The nodes were then modified with a series of 100 *potentiation* and *depression* pulses of magnitude 4.5 V and duration 250 ms and the corresponding weights were recorded. This procedure represented the 100 updates of a single synapse. The tunneling nodes were then reinitialized to the zero memory point and the procedure was repeated with different random series of input pulses representing the modification of other 99 synapse in the network. The first input pulses of each series of modification forms the tracked memory pattern. To modify the value of γ the FN-synapses were initialized at a higher tunneling node potential.

The behavioral model of the FN-synapse was generated by extracting the device parameters k_1 and k_2 from the hardware prototype. The extracted parameters have been shown to capture the hardware response with an accuracy greater than 99.5% in our previous works (Zhou and Chakrabartty, 2017; Zhou et al., 2019). These extracted parameters were fed into a dynamical system which follows the usage profile described in the hardware implementation subsection and follow the weight update rule elaborated in the SNR estimation subsection to reliably imitate the behavior of the FN-synapse. The behavioral model network was started with exactly the same initial condition as hardware synapses and subjected to the exact memory patterns used for the hardware experiment for the same number of iterations. The simulation was also extended to 1,000 iterations and the corresponding responses are included in Figure 4F.

3.8. Probabilistic FN-synapse model

Adaption of FN-synapse occurs by tunneling of electrons through a triangular FN quantum-tunneling barrier. The tunneling current density is dependent on the barrier profile which in turn is a function of the floating-gate potential. When W^+, W^- is around 7 V the synaptic update ΔW_d due to an external pulse can be determined by the continuous and deterministic form of the FN-synapse model (as described in the previous sections). Since the number of electrons tunneling across the barrier is relatively large ($\gg$1), the method is adequate for determining ΔW_d. However, once W^+, W^- is around 6 V, each updates occurs due to the transport of a few electrons tunneling across the barrier and in the limit by a single electron tunneling across the barrier at a time. In this regime, the continuous behavioral model is no longer valid. Therefore, the behavioral model of the FN-synapse has to switch to a probabilistic model. In this mode of operation, we can assume that each electron tunneling event follows a Poisson process where the number of electrons $e^+(n), e^-(n)$ tunneling across the two junctions during the n^{th} input pulse is estimated by sampling from a Poisson distribution with rate parameters λ^+, λ^- given by

$$\lambda^+(n) = \frac{AJ(W^+(n))}{q} \tag{44}$$

$$\lambda^-(n) = \frac{AJ(W^-(n))}{q}. \tag{45}$$

q is the charge of an electron, A is the cross-sectional area of the tunneling junction. Using the sampled values of $e^+(n), e^-(n)$, the corresponding discrete-time stochastic equation governing the dynamics of the tunneling node potentials $W^+(n)$, $W^-(n)$ is given by

$$W^+(n) = W^+(n-1) - \frac{qe^+(n)}{C_T} \tag{46}$$

$$W^-(n) = W^-(n-1) - \frac{qe^-(n)}{C_T} \tag{47}$$

where C_T is the equivalent capacitance of the tunneling node.

We have verified the validity/accuracy of the probabilistic model against the continuous-time deterministic model in high tunneling rate regimes. Supplementary Figure 4A shows that the output of the probabilistic model matches closely to the deterministic model and the deviation which arises due to the random nature of the probabilistic updates (shown in Supplementary Figure 4B) is within $200\mu V$. Using the probabilistic model we performed the memory retention and network capacity experiments (as discussed in the main manuscript) by initializing the tunneling nodes at a low potential. In this regime, each updates to the FN synapse results from tunneling of a few electrons. Supplementary Figures 4C, D show that even when each update sizes are on the order of tens of electrons, the network capacity and memory retention time remains unaffected. However, as the synaptic voltage is modified by less than ten electrons per update (shown in Supplementary Figure 4E), the SNR curve starts to shift downwards and the network capacity along with memory retention time decreases. The tunneling node potential can be pushed further down to a region where the synapses might not even register modifications at times and other times update sizes drop down to single electron per modification (see Supplementary Figure 4F). In this regime, the SNR curve shifts down further, the SNR decay still obeys the power-law curve.

3.9. Neural network implementation using FN-synapses

The MNIST dataset was split into 60,000 training images and 10,000 test images which yielded about 6,000 training images and 1,000 test images per digit. Each image, originally of 28×28 pixels, was converted to 32 × 32 pixels through zero-padding. This was followed by standard normalization to zero mean with unit variance. The code for implementing the non-FN-synapse approaches such as EWC and online EWC were obtained from the repository mentioned in Hsu et al. (2018). To enforce an equitable comparison, the same neural network architecture (as shown in Supplementary Figure 6), in the form a multi-layered perceptron (MLP) with an input layer of 1024 nodes, two hidden layers of 400 nodes each (paired with the ReLU activation function) and a softmax output layer of 2 nodes, has been utilized by every method mentioned in this work. Based on the optimizer in use, a learning rate of 0.001 was chosen for both SGD and ADAM (with additional parameters β_1, β_2, and ϵ set to 0.9, 0.999, and 10^{-8}, respectively, for the latter). Each model was trained with a mini-batch size of 128 for a period of 4 epochs.

Similar to the continual learning experiments conducted on split-MNIST, benchmark incremental-domain learning experiments were also carried out by randomly permuting the order of pixels of the images in the MNIST dataset in accordance with Hsu et al. (2018) which is referred as the Permuted-MNIST. The architecture of the neural network employed is similar to the one for the split-MNIST with the exception of being equipped with 1,000 neurons in each of the two hidden layers instead of 400 and with 10 neurons in the output layer instead of 2. This essentially means that at each task, the network learns a new set of permutations of the 10 digits. The network was trained on 10 such tasks for 3 epochs using a learning rate of 0.0001 for ADAM and 0.001 for ADAGRAD.

Corresponding to every weight/bias in the MLP, an instance of the FN-synapse model was created and initialized to a tunneling region according to the initial W_c value. As demonstrated by the measured results, ΔW_d can be modulated linearly and precisely by changing the pulse-width of the

potentiation/depression pulses. Therefore, each weight update (calculated according to the optimizer in use) is mapped as an input pulse of proportional duration for the FN synapse instance. Then, every instance of the FN-synapse model is updated according to Equation (27) and the W_d thus obtained in voltage is scaled back to a unit-less value and within the required range of the network.

4. Discussion

In this paper, we reported a differential FN quantum-tunneling based synaptic device that can exhibit near-optimal memory consolidation that has been previously demonstrated using only algorithmic models. The device called FN-synpase, like its algorithmic counterparts, stores the value of the weight and a relative usage of the weight that determines the plasticity of the synapse. Similar to algorithmic consolidation models, an FN-synapse, "protects" important memory by reducing the plasticity of the synapse according to its usage for a specific task. However, unlike its algorithmic counterparts like the cascade or EWC models, the FN-Synapse doesn't require any additional computational or storage resources. In EWC models memory consolidation in continual learning is achieved by augmenting the loss function using penalty terms that are associated with either Fisher information (Kirkpatrick et al., 2017) or the historical trajectory of the parameter over the course of learning (Chaudhry et al., 2018; Liu et al., 2018). Thus, the synaptic updates require additional pre-processing of the gradients, which in some cases could be computationally and resource intensive. FN-synapse on the other hand, does not require any pre-processing of gradients and instead can exploit the physics of the device itself for synaptic intelligence and for continual learning. For some benchmark tasks, we have shown an FN-synapse network shows better multi-task accuracy compared to other continual learning approaches. This leads to the possibility that the intrinsic dynamics of the FN-synapse could provide important clues on how to improve the accuracy of other continual learning models as well.

Figures 6A, B also show the importance of the learning algorithm in fully exploiting the available network capacity. While the entropy of the FN-synapse weights for the output layer is relatively high, the entropy of the weights of the input layer is still relatively low, implying most of the input layer weights remain unused. This is an artifact of *vanishing gradients* in a standard backpropagation based neural network learning. Thus, it is possible that improved backpropagation algorithms (Deng et al., 2016; Tan and Lim, 2019) might be able to mitigate this artifact and in the process enhance the capacity and the performance of the FN-synapse network. In Supplementary Figure 8, we show that FN-synapse based neural network is able to maintain its performance even when the network size is increased. Thus, it is possible that the network becomes capable of learning more complex tasks due to increase in overall plasticity of the network while ensuring considerably better retention than neural networks with traditional synapses.

In addition to being physically realizable, the FN-synapse implementation also allows interpolation between a steady state consolidation model and the EWC consolidation models. This is important because it is widely accepted that the EWC model can potentially suffer from blackout catastrophe (Kirkpatrick et al., 2017) as the learning network approaches its capacity. During this phase, the network becomes incapable of retrieving any previous memory as well as is unable to learn new ones (Kirkpatrick et al., 2017). Steady-state models such as the cascade consolidation models and SGD-based continuous learning models avoid this catastrophe by gracefully forgetting old memories. As shown in Figure 5A, an FN-synapse network, through the use of a global modulation factor $m(t)$, is able to interpolate between the two models. In fact, the results in Figures 5A, B, show that the number of patterns/memories retained in an FN-synapse network under modulation profile $m_2(t)$ at steady state is higher compared to that of a high-complexity cascade model for a network size of $N = 1,000$ synapses. Even though we have not used the interpolation feature for benchmark experiments, we believe that this attribute is going to provide significant improvements for continuous learning of a large number of tasks.

The interpolation property of FN-synapse could mimic some attributes of *metaplasticity* observed in biological synapses and dendritic spines (Mahajan and Nadkarni, 2019). The role of metaplasticity, the second-order plasticity of a synapse which assigns a task-specific importance to every successive task being learned (Laborieux et al., 2021), is widely accepted as the fundamental component of neural processes key to memory and learning in the hippocampus (Abraham and Bear, 1996; Abraham, 2008). Since unregulated plasticity leads to runaway effects resulting in previously stored memories to be impaired at saturation of synaptic strength (Brun et al., 2001), metaplasticity serves as a regulatory mechanism which dynamically links the history of neuronal activity with the current response (Hulme et al., 2014). The FN-synapse mimics the same regulatory mechanism through the decaying term $r(t)$ that takes into account the history of usage or neuronal activity to determine the plasticity of the synapse for future use as well as prevents runaway effects by making the synapses rigid at saturation.

The on-device memory consolidation in FN-synapse can not only minimize the energy requirements in continual learning tasks, additionally, the energy required for a single synaptic weight update is also lower than memristor-based synaptic updates for a fixed precision of update. This attribute has been validated in our previous works (Mehta et al., 2022) where the update energy was estimated to be as low as 5f J increasing up to 2.5p J depending on the status of the FN-synapse and the desired change in synaptic weights. Note that the energy required to change the synaptic weight is derived from the FN-tunneling

current and not from the electrostatic energy used for charging the coupling capacitor. Thus, by designing more efficient charge-sharing techniques across the coupling capacitors the energy-efficiency of FN-synaptic updates can be significantly improved. Furthermore, when implemented on more advanced silicon process nodes, the capacitances could be scaled which can improve the energy-efficiency of FN-synapse by an order of magnitude. Compared to memristor-based synapses, the FN-synapse can also exhibit high endurance $10^6 - 10^7$ cycles without any deterioration. However, the key distinction lies in terms of the dynamic range of the stored weights. Generally, a single memristor has two distinct conductive states (corresponding to "0" or "1") which give each device a 1-bit resolution. When used in a crossbar array, highly-dense designs can reach densities up to 76.5 nm^2 per bit as reported by Poddar et al. (2021) where a 3-D memristor array was constructed using Perovskite quantum wires. The dynamic range or resolution of such designs is determined by the number of memristive devices that can be packed into the smallest feasible physical form factor. If we consider multi-level memristors instead, the resolution per memristor can reach up to 3-5 bits depending on the number of stable distinguishable conductive states (He et al., 2017; Wu et al., 2019; Lee et al., 2021). In comparison, the dynamic range of the FN-synapse (a single device) is considerably higher as it is determined by the number of electrons stored on the floating-gates which in-turn is determined by the FN-synapse form-factor and the dielectric property of the tunneling barrier. Thus, theoretically, the dynamic range and the operational-life of the FN-synapse seems to be constrained by the single-electron quantization. However, at low-tunneling regimes, the transport of single electrons across the tunneling barrier becomes probabilistic where the probability of tunneling is now modulated by the external signals $X(t)$ and $m(t)$. In the Section 3 and in Supplementary Figure 4, we show that a stochastic dynamical system model emulating the single-electron dynamics in the FN-synapse can produce $\mathcal{O}(1/\sqrt{t})$ consolidation characteristics for the benchmark random input patterns experiment for an empty network. The SNR still follows the power-law curve and the FN-synapse network continues to learn new experiences even if the synaptic updates are based on discrete single-electron transport. A more pragmatic challenge in using the FN-synapse will be the ability of the read-out circuitry to discriminate between the changes in floating-gate voltage due to single-electron tunneling events. For the magnitude of the floating-gate capacitance, the change in voltage would be in the order of 100 nV per tunneling event. A more realistic scenario would be to measure the change in voltage after 1,000 electron tunneling events which would imply measuring 100 μV changes. Although this will reduce the resolution of the stored weights/updates to 14 bits, recent studies have shown that neural networks with training precisions as low as 8 bits (Sun et al., 2019) and networks with inference precisions as low as 2–4 bits (Choi et al., 2018, 2019) are often capable of exhibiting remarkably good learning abilities. In Supplementary Figure 9, we show that for the split-MNIST task, the performance of the FN-synapse based neural network remains robust even in the presence of 5% device mismatch.

Another point of discussion is whether the optimal decay profile $r(t) \approx \mathcal{O}(1/t)$ can be implemented by other synaptic devices, in particular, the energy-efficient memristor-based synapses that have been proposed for neuromorphic computing (Tuma et al., 2016; Fuller et al., 2019; Pal et al., 2019a,b; Karunaratne et al., 2020; Mehonic et al., 2020). Recent works using memristive devices have demonstrated on-device *metaplasticity* (Giotis et al., 2022), however, achieving an optimal decay profile would require additional control circuitry, storage and read-out circuits. In this regard, we believe that the FN-synapse represents one of the few, if not the only class of synaptic devices that can achieve optimal memory consolidation on a single device.

Data availability statement

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

Author contributions

SC and MR came up with the concept of FN-synapse. MR, SB, and SC designed the hardware and simulation experiments. MR designed the 64 element FN-synapse chipset. MR and SB conducted the simulation and hardware experiments. SC provided supervision on all tasks. All authors contributed toward writing and proof-reading the manuscript. All authors contributed to the article and approved the submitted version.

Funding

This work was supported in part by the National Science Foundation Grants FET: 2208770 and ECCS: 1935073.

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.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2022.1050585/full#supplementary-material

References

Abraham, W. C. (2008). Metaplasticity: tuning synapses and networks for plasticity. *Nat. Rev. Neurosci.* 9, 387–387. doi: 10.1038/nrn2356

Abraham, W. C., and Bear, M. F. (1996). Metaplasticity: the plasticity of synaptic plasticity. *Trends Neurosci.* 19, 126–130. doi: 10.1016/S0166-2236(96)80018-X

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2017). Memory aware synapses: learning what (not) to forget. *arXiv preprint arXiv: 1711.09601*. doi: 10.48550/arXiv.1711.09601

Amit, D. J., and Fusi, S. (1994). Learning in neural networks with material synapses. *Neural Comput.* 6, 957–982. doi: 10.1162/neco.1994.6.5.957

Benna, M., and Fusi, S. (2016). Computational principles of synaptic memory consolidation. *Nat. Neurosci.* 19, 1697–1706. doi: 10.1038/nn.4401

Brun, V. H., Ytterbø, K., Morris, R. G., Moser, M.-B., and Moser, E. I. (2001). Retrograde amnesia for spatial memory induced by NMDA receptor-mediated long-term potentiation. *J. Neurosci.* 21, 356–362. doi: 10.1523/JNEUROSCI.21-01-00356.2001

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. S. (2018). Riemannian walk for incremental learning: understanding forgetting and intransigence. *arXiv preprint arXiv: 1801.10112*. doi: 10.1007/978-3-030-01252-6_33

Choi, J., Venkataramani, S., Srinivasan, V. V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. *Proc. Mach. Learn. Syst.* 1, 348–359.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I., Srinivasan, V., and Gopalakrishnan, K. (2018). PACT: parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv: 1805.06085*. doi: 10.48550/arXiv.1805.06085

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 653–664. doi: 10.1109/TNNLS.2016.2522401

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

Fusi, S. (2002). Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. *Biol. Cybernet.* 87, 459–470. doi: 10.1007/s00422-002-0356-8

Fusi, S., and Abbott, L. (2007). Limits on the memory storage capacity of bounded synapses. *Nat. Neurosci.* 10, 485–493. doi: 10.1038/nn1859

Fusi, S., Drew, P. J., and Abbott, L. (2005). Cascade models of synaptically stored memories. *Neuron* 45, 599–611. doi: 10.1016/j.neuron.2005.02.001

Giotis, C., Serb, A., Manouras, V., Stathopoulos, S., and Prodromakis, T. (2022). Palimpsest memories stored in memristive synapses. *Sci. Adv.* 8, eabn7920. doi: 10.1126/sciadv.abn7920

He, W., Sun, H., Zhou, Y., Lu, K., Xue, K., and Miao, X. (2017). Customized binary and multi-level HfO2- x-based memristors tuned by oxidation conditions. *Sci. Rep.* 7, 1–9. doi: 10.1038/s41598-017-09413-9

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. (2018). Re-evaluating continual learning scenarios: a categorization and case for strong baselines. *arXiv preprint arXiv: 1810.12488*. doi: 10.48550/arXiv.1810.12488

Hulme, S. R., Jones, O. D., Raymond, C. R., Sah, P., and Abraham, W. C. (2014). Mechanisms of heterosynaptic metaplasticity. *Philos. Trans. R. Soc. B Biol. Sci.* 369, 20130148. doi: 10.1098/rstb.2013.0148

Karunaratne, G., Le Gallo, M., Cherubini, G., Benini, L., Rahimi, A., and Sebastian, A. (2020). In-memory hyperdimensional computing. *Nat. Electron.* 3, 327–337. doi: 10.1038/s41928-020-0410-3

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. U.S.A.* 114, 3521–3526. doi: 10.1073/pnas.1611835114

Laborieux, A., Ernoult, M., Hirtzlin, T., and Querlioz, D. (2021). Synaptic metaplasticity in binarized neural networks. *Nat. Commun.* 12, 1–12. doi: 10.1038/s41467-021-22768-y

Lee, S., Jeon, J., Eom, K., Jeong, C., Yang, Y., Park, J.-Y., et al. (2021). Multi-level memristors based on two-dimensional electron gases in oxide heterostructures for high precision neuromorphic computing. *Res. Square*. doi: 10.21203/rs.3.rs-1019162/v1

Lee, S., Kim, J., Ha, J., and Zhang, B. (2017). Overcoming catastrophic forgetting by incremental moment matching. *arXiv preprint arXiv: 1703.08475*. doi: 10.48550/arXiv.1703.08475

Lenzlinger, M., and Snow, E. H. (1969). Fowler–Nordheim tunneling into thermally grown SiO2. *J. Appl. Phys.* 40, 278–283. doi: 10.1063/1.1657043

Li, Q., Navakkode, S., Rothkegel, M., Soong, T. W., Sajikumar, S., and Korte, M. (2017). Metaplasticity mechanisms restore plasticity and associativity in an animal model of Alzheimer's disease. *Proc. Natl. Acad. Sci. U.S.A.* 114, 5527–5532. doi: 10.1073/pnas.1613700114

Liu, X., Masana, M., Herranz, L., van de Weijer, J., López, A. M., and Bagdanov, A. D. (2018). Rotate your networks: better weight consolidation and less catastrophic forgetting. *arXiv preprint arXiv: 1802.02950*. doi: 10.1109/ICPR.2018.8545895

Mahajan, G., and Nadkarni, S. (2019). Intracellular calcium stores mediate metaplasticity at hippocampal dendritic spines. *J. Physiol.* 597, 3473–3502. doi: 10.1113/JP277726

Mehonic, A., Sebastian, A., Rajendran, B., Simeone, O., Vasilaki, E., and Kenyon, A. J. (2020). Memristors–from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing. *Adv. Intell. Syst.* 2, 2000085. doi: 10.1002/aisy.202000085

Mehta, D., Aono, K., and Chakrabartty, S. (2020). A self-powered analog sensor-data-logging device based on Fowler-Nordheim dynamical systems. *Nat. Commun.* 11. doi: 10.1038/s41467-020-19292-w

Mehta, D., Rahman, M., Aono, K., and Chakrabartty, S. (2022). An adaptive synaptic array using Fowler–Nordheim dynamic analog memory. *Nat. Commun.* 13, 1–11. doi: 10.1038/s41467-022-29320-6

Pal, S., Bose, S., and Islam, A. (2019a). Design of memristor based low power and highly reliable reram cell. *Microsyst. Technol.* 28, 1–15. doi: 10.1007/s00542-019-04582-1

Pal, S., Bose, S., Ki, W.-H., and Islam, A. (2019b). Design of power-and variability-aware nonvolatile rRAM cell using memristor as a memory element. *IEEE J. Electron Devices Soc.* 7, 701–709. doi: 10.1109/JEDS.2019.2928830

Poddar, S., Zhang, Y., Gu, L., Zhang, D., Zhang, Q., Yan, S., et al. (2021). Down-scalable and ultra-fast memristors with ultra-high density three-dimensional arrays of perovskite quantum wires. *Nano Lett.* 21, 5036–5044. doi: 10.1021/acs.nanolett.1c00834

Rahman, M., Zhou, L., and Chakrabartty, S. (2022). SpotKD: a protocol for symmetric key distribution over public channels using self-powered timekeeping devices. *IEEE Trans. Inform. Forensics Sec.* 17, 1159–1171. doi: 10.1109/TIFS.2022.3158089

Roxin, A., and Fusi, S. (2013). Efficient partitioning of memory systems and its importance for memory consolidation. *PLoS Comput. Biol.* 9, e1003146. doi: 10.1371/journal.pcbi.1003146

Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J., and Re, C. (2019). Low-memory neural network training: a technical report. *arXiv preprint arXiv:1904.10631*. doi: 10.48550/arXiv.1904.10631

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V. V., et al. (2019). "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks, in *Advances in Neural Information Processing Systems*, Vol. 32, eds H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (Vancouver, CA: Curran Associates, Inc.).

Tan, H. H., and Lim, K. H. (2019). "Vanishing gradient mitigation with deep learning neural network optimization, in *2019 7th International Conference on Smart Computing & Communications (ICSCC)* (IEEE), 1–4. doi: 10.1109/ICSCC.2019.8843652

Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A., and Eleftheriou, E. (2016). Stochastic phase-change neurons. *Nat. Nanotechnol.* 11, 693–699. doi: 10.1038/nnano.2016.70

Wu, L., Liu, H., Li, J., Wang, S., and Wang, X. (2019). A multi-level memristor based on AL-doped HfO2 thin film. *Nanoscale Res. Lett.* 14, 1–7. doi: 10.1186/s11671-019-3015-x

Yang, G., Lai, C. S. W., Cichon, J., Ma, L., Li, W., and Gan, W.-B. (2014). Sleep promotes branch-specific formation of dendritic spines after learning. *Science* 344, 1173–1178. doi: 10.1126/science.1249098

Yang, G., Pan, F., and Gan, W.-B. (2009). Stably maintained dendritic spines are associated with lifelong memories. *Nature* 462, 920–924. doi: 10.1038/nature08577

Zenke, F., Poole, B., and Ganguli, S. (2017). Improved multitask learning through synaptic intelligence. *arXiv preprint arXiv: 1703.04200*. doi: 10.48550/arXiv.1703.04200

Zhou, L., and Chakrabartty, S. (2017). Self-powered timekeeping and synchronization using Fowler-Nordheim tunneling-based floating-gate integrators. *IEEE Trans. Electron Devices*. 64, 1–7. doi: 10.1109/TED.2016.2645379

Zhou, L., Kondapalli, S. H., Aono, K., and Chakrabartty, S. (2019). Desynchronization of self-powered fn tunneling timers for trust verification of IoT supply chain. *IEEE Internet Things J.* 6, 6537–6547. doi: 10.1109/JIOT.2019.2907930

frontiers | Frontiers in Neuroinformatics

TYPE Original Research
PUBLISHED 21 March 2023
DOI 10.3389/fninf.2023.1125844

OPEN ACCESS

EDITED BY
Toshiyuki Yamane,
IBM Research, Tokyo, Japan

REVIEWED BY
Shuangming Yang,
Tianjin University, China
Chenglong Zou,
Peking University, China

*CORRESPONDENCE
Michael Hopkins
✉ michael.hopkins@manchester.ac.uk

RECEIVED 16 December 2022
ACCEPTED 03 March 2023
PUBLISHED 21 March 2023

CITATION
Hopkins M, Fil J, Jones EG and Furber S (2023) BitBrain and Sparse Binary Coincidence (SBC) memories: Fast, robust learning and inference for neuromorphic architectures.
Front. Neuroinform. 17:1125844.
doi: 10.3389/fninf.2023.1125844

BitBrain and Sparse Binary Coincidence (SBC) memories: Fast, robust learning and inference for neuromorphic architectures

Michael Hopkins*, Jakub Fil, Edward George Jones and Steve Furber

Advanced Processor Technologies Group, Department of Computer Science, The University of Manchester, Manchester, United Kingdom

We present an innovative working mechanism (the *SBC memory*) and surrounding infrastructure (*BitBrain*) based upon a novel synthesis of ideas from sparse coding, computational neuroscience and information theory that enables fast and adaptive learning and accurate, robust inference. The mechanism is designed to be implemented efficiently on current and future neuromorphic devices as well as on more conventional CPU and memory architectures. An example implementation on the SpiNNaker neuromorphic platform has been developed and initial results are presented. The SBC memory stores coincidences between features detected in class examples in a training set, and infers the class of a previously unseen test example by identifying the class with which it shares the highest number of feature coincidences. A number of SBC memories may be combined in a *BitBrain* to increase the diversity of the contributing feature coincidences. The resulting inference mechanism is shown to have excellent classification performance on benchmarks such as MNIST and EMNIST, achieving classification accuracy with single-pass learning approaching that of state-of-the-art deep networks with much larger tuneable parameter spaces and much higher training costs. It can also be made very robust to noise. *BitBrain* is designed to be very efficient in training and inference on both conventional and neuromorphic architectures. It provides a unique combination of single-pass, single-shot and continuous supervised learning; following a very simple unsupervised phase. Accurate classification inference that is very robust against imperfect inputs has been demonstrated. These contributions make it uniquely well-suited for edge and IoT applications.

KEYWORDS

single-pass learning, neuromorphic, efficient inference, classification, machine learning, robust, event-based, IoT

1. Introduction

With the inevitable ubiquity of AI (Artificial Intelligence) decision-making that the IoT (Internet of Things) will facilitate, there is a clear need for mechanisms and architectures that allow accurate inferences to be made quickly, robustly (in the presence of imperfect input data) and with low energy use. Current state of the art information architectures such as deep learning can provide excellent inference but at the cost of vast numbers of parameters that need to be learned at training time and computed at inference time. This makes their

learning phase a huge resource commitment in both energy and time, and discourages a local implementation of inference at the end of realistic network branches where storage, energy and communication resources are likely to be severely constrained. Arguably, their requirement for huge parameter sets also makes them liable to over-fitting and the lack of robustness that this leads to, leading some researchers to use *ad hoc* methods such as dropout which can sometimes improve performance but at the cost of a further significant learning burden. It is also currently not clear how well they can be applied to the problem of continuous learning which is likely to become more important in practical applications.

In the near future, intelligent decisions and classifications are going to become required in an increasing number of devices and architectures with constrained resources. This will focus attention on the following issues, and technologies that help to address them will become desirable:

- Faster learning and inference.
- Energy-efficient learning and inference.
- More robust inference in the presence of imperfections and noise at the network inputs and partial system errors so that performance degradation is graceful rather than brittle.
- An easy mapping onto the growing number of neuromorphic architectures that facilitate large gains in speed and energy efficiency.
- A natural receiving mechanism for event-based visual and audio sensors which leverage further gains in energy use and communication bandwidth.

These technologies should be of interest to anyone who wants to address the issues described above, in particular those looking to make fast and reliable inferences at the edge. Another possibility is those who would like to create a large number of small and efficient self-contained inference modules which perform potentially complex pattern recognition with a very low energy-latency product and communicate *via* relatively simple and sparse messages. The latter is a good conceptual match to the dendritic computation paradigm in computational neuroscience which is gaining traction, and which changes quite significantly the balance between computation and communication in large neural networks where each action potential now becomes a carrier of more information content.

We introduce an innovative working mechanism (the *SBC memory*) and surrounding infrastructure (*BitBrain*) based upon a novel synthesis of ideas from sparse coding, computational neuroscience and information theory.[1] The key contributions of this technology presented here are:

- Single-pass and single-shot supervised learning; following a simple unsupervised phase where parameters are learned quickly in a simple and "local" way that does not require global optimisation over high-dimensional spaces or the calculation of derivatives.
- Accurate inference (currently classification) that is very robust against imperfect inputs.
- Simple support for continuous adaptive learning.
- Algorithms that are designed to be implemented with excellent energy efficiency on conventional CPUs and memory architectures, and on current and future neuromorphic devices.
- A natural target for the increasing number of event-based sensors such as silicon retinas, enabling further energy and bandwidth gains to be exploited—in particular for edge computing and IoT devices.

2. Background

The ideas that contributed to the *BitBrain* mechanism are drawn from a variety of areas:

2.1. Sparsity of activity and homeostasis

An aspect of neural activity which is clear in the neocortex and also globally to some extent is that activity—in terms of action potentials at least—is relatively sparse. Perhaps 5–10% of neurons are firing in a specific time window. If one considers the massive complexity of the brain and all its interconnections then this surely must imply some kind of self-adaption or self-regulation, and for this to work consistently one can further infer that it should be present at the local level. Hence we believe that some form of homeostasis in the basic functional mechanisms is an important part of any neural or neurally-inspired mechanism. In this context, homeostasis means that the underlying physiological mechanisms tend toward a natural, equilibrium rate of activity despite all the complex non-linearities and interactions that they share.

Sparse memory mechanisms have been discussed before. One important example is the concept of Sparse Distributed Memory (SDM) introduced by Pentti Kanerva in his PhD thesis (Kanerva, 1988). The key concept in an SDM is to use random address decoders to map a binary input space into a very high-dimensional intermediate space where associated information can be stored very sparsely and redundantly, leading to robust recovery of that information whenever a similar input is presented to the memory. Kanerva speculated that a similar mechanism might be at work in the cerebellum, where there are similarities between the neuronal organisation and the structure of an SDM. Furber et al. developed this idea further showing that the same approach was effective using sparse *N-of-M* codes for both the input and the stored information (Furber et al., 2004) and an SDM could even be used to store and recover the temporal patterns (Furber et al., 2007) when rank-order codes (Thorpe and Gautrais, 1998) were used.

There are similarities between the address decoders used in our SBCs and those used in SDMs, though we have added homeostatic tuning mechanisms such as threshold adjustment and structural plasticity to the original address decoder concept.

1 A patent GB 2113341.8 covering this technology was filed by MH and SF at the UKIPO on 17th September 2021.

2.2. Robustness in the presence of noise, errors, or partial failure

Biological neural systems are extremely good at inferring correct decisions and actions from imperfect sources of information, whether this is poor data from sensory systems in noisy or otherwise perturbed environments, imperfect conditions within the neural mechanism itself (such as the presence of alcohol or other disorders of the ideal equilibrium) or a total failure of some parts of the system.

Neurons and synapses are far from perfect processing elements, with probabilistic and somewhat unreliable transfer functions even when working at their full potential, and prone to change and failure as are all biological mechanisms. Yet in the presence of all these perturbations the system as a whole performs very well. Understanding how this apparent paradox can be explained will be an important step forwards in engineering mechanisms of inference that degrade gracefully in the presence of realistic amounts of error and uncertainty in their working environment. In the 1980s, inspired partly by the thinking of the Japanese engineer Genichi Taguchi, much research and practical work focused on the importance of this issue in engineering robust systems (Phadke, 1989; Edwards et al., 2000) and the outcome was a change of focus in the design process which remains current today.

It has also been suggested that noise and uncertainty within the processing mechanism is not a problem but is in fact a valuable resource in allowing this robustness to occur (Maass, 2014) and this is a view that we agree with.

2.3. Avoiding optimisation over high-dimensional parameter spaces

Optimisation over high-dimensional parameter spaces with multiple non-linear objective functions where there are vast numbers of local optima is very hard to do consistently well, and intuitions about dimensions above about 10 do not serve well in realistic problems. When one considers that in some contemporary deep neural networks (DNNs) the learning mechanism is optimising an error function over perhaps billions or trillions of parameters, one can see that both the energy and time cost, and the almost guaranteed sub-optimality that will result, are major issues. That is to say nothing of the inevitability of over-fitting, which is clear from probability and information theory when the numbers of independent degrees of freedom in the system may be several orders of magnitude higher than required for the appropriate model (see for example, chapter 4 in both Sivia and Skilling, 2006 and Jaynes, 2003 books). As well as being too numerous these degrees of freedom are rarely if ever, apportioned where they are most justified within the resulting very complex models though work has been done on how to approach this problem on a rational basis (Tishby et al., 1999).

2.4. Spike-time coding, dendritic computation, and local unsupervised learning

There has been a long history of debate about the coding mechanisms used in brains to represent and transmit information. The main two contenders at the base computational level are spike-time and rate coding, though these apparently distinct categories can overlap somewhat at the extremes of their ranges (Reike et al., 1996). For output neurons that drive muscles there is general agreement that rate coding is used, however within more time-critical and energy-sensitive parts of the brain many believe that spike-time coding must be involved. This opens up many avenues of exploration for the kinds of representations and learning mechanisms that generate the sparse activity that we observe, whilst at the same time allowing fast and energy-efficient computation.

It is clear that neural systems do at least some of their learning (i) locally and (ii) without reference to a global error or utility function. This is presumably to help the mechanism as a whole orient itself in relation to the representations required in order to solve the higher-level problem using the minimum amount of time and energy. In machine learning terms we can say this is *unsupervised learning*. A proven neurally-inspired mechanism for facilitating this is spike-time dependent plasticity (STDP), but the decision about where to access the required post-synaptic signal can be debated. Many implementations choose to use the action potential at the soma after having back-propagated through the dendritic tree, but there are some issues with both timing and reliability in this model. With the recent increased interest in dendritic computation (London and Häusser, 2005; Stuart et al., 2016) it has become apparent that NMDA (N-methyl-D-aspartate) potentials local to the synapses involved (i.e., within the local synaptic cluster or at least in a close part of the dendritic branch) can provide the signal required without these issues (Larkum and Nevian, 2008; Branco and Häusser, 2010; Govindarajan et al., 2011) and some work has already been done to apply these understandings to neuromorphic architectures (Yang et al., 2021a). There are many other aspects of dendritic computation which may elucidate mechanisms that allow for sparse and robust representations which balance local and global behaviours (Mel, 1992; Papoutsi et al., 2014; Kastellakis et al., 2015; Ahmad and Hawkins, 2016; Richards and Lillicrap, 2019).

Although adjusting the size of local synapses and hence their drive capability is often the chosen mechanism for STDP, we instead choose structural plasticity as an effective mechanism so that synapses are added or removed using a Hebbian approach (Hopkins et al., 2018) where synapse size only relates to its longevity. This allows us to stay with binary computation and connectivity in order to stay consonant with some of the other aims outlined in this section.

2.5. Low resolution computation and mapping onto neuromorphic substrates

In neural systems memory and computation are colocated, setting it apart from the von Neumann model. The ideas inherent

in neuromorphic computation should help us to understand how biological neural systems achieve their remarkable performance and energy-efficiency. Such mechanisms should take advantage of engineering opportunities for: energy efficiency, sparsity of activity, low resolution (ideally binary) computation and communication, massive parallelism, asynchrony and event-driven computation. Choices made at the algorithmic design stage can facilitate this mapping onto current and future substrates.

Contemporary large-scale machine learning displays a strong trend toward lower-resolution parameters and computation to leverage the large gains in energy and storage efficiency that result. The lowest useful resolution is the single bit. Earlier work has taken similar directions, though at the time probably for different reasons. Random Access Memory (RAM)-based methods for machine learning have been around since the late 1950s. Their direct use of RAM for storing the inference mechanism based upon complex patterns of binary logic learned directly from the data provides a simple and fast mechanism that should be amenable to hardware optimisation. After a period of relative obscurity, these ideas had a renaissance in the 1970s as *RAM-nets* or *N-Tuple methods*, and particularly in the 1980s where they benefited from some of the mindshare developed by the renewed interest in neural networks under the name of *weightless neural networks* where Austin (1998) collects together some of the more advanced work in this area. There are resonances from those ideas in the work presented here.

Another more contemporary neuromorphic approach using custom system and routing hardware and multiple FPGAs is inspired directly by brain connectivity patterns and provides an alternative set of trade-offs for energy, scaling and speed in realistic neural simulation and learning scenarios (Yang et al., 2021b).

2.6. Kernel methods and their mapping into high-dimensional feature spaces

Kernel methods have proven to be a powerful and versatile tool in many areas of machine learning (Shawe-Taylor and Cristianini, 2004; Rasmussen and Williams, 2006). By exploiting only the similarity/difference between cases and projecting (usually non-linearly) into high-dimensional feature spaces that match the data distribution in some sense, both continuous approximation and discrete classification problems can be solved accurately and with few assumptions. The practical limitations are primarily due to the quantity of data and the expensive $\mathbf{O}(n^3)$ linear computations that usually result, and the necessity of finding closed-form kernel functions for practical efficiencies, in particular for inference.

As discussed more thoroughly in Section 6, we see a number of analogies between Kernel methods and the ideas that we are exploring here. Our method can be seen as constructing a non-linear projection into a high-dimensional feature space and dot products in this space can be used to assess similarity or difference and generate impressive inference accuracies considering that it is a simple and automatic algorithm. Both methods are basically non-parametric, i.e., using the data themselves for inference rather than parameters learned from them. It is also the *coincidences between* our analogue of feature detectors abstracted from synaptic clusters (Mel, 1992) that are at the heart of our method and these too could be seen as dot products in some space. We hope that a further understanding of these parallels will provide a better foundation for the theory of our method.

Neal (1996) elucidates and explores interesting parallels between Kernel methods (specifically *Gaussian Processes*) and neural networks.

3. An overview of the basic mechanisms

Taking inspiration from the conception of synaptic clusters and their ability to both create and learn from local NMDA plateau potentials, a key concept within the SBC memory is that of an Address Decoder Element (*ADE*). This subsamples from the input stream—initially in a random fashion—and then during the unsupervised learning phase each ADE "homes in" on a feature and at the same time learns a homeostatic threshold θ to facilitate a target firing probability that creates sparse activity patterns. By adding delays of differing values to the synapses in an ADE one can also detect temporal coincidence patterns and so carry out a combined spatio-temporal classification for input data where this is relevant. Figure 1 illustrates the basic mechanism.

In the form of an equation where j indexes every ADE.

$$\forall_j \; activation_j = \sum_{i=1}^{synapses(ADE_j)} input_i \times weight_i \tag{1}$$

where $synapses(ADE_j) = 6$ in Figure 1. The homeostatic threshold θ_j has been learned for each ADE during the unsupervised phase. Then

$$\forall_j \text{ if } activation_j \geq \theta_j \rightarrow ADE_j \text{ fires} \tag{2}$$

The ADEs can be organised in flexible ways. One method that is convenient for software exploration and a simple description is in vectors which we will call Address Decoders (*AD*s) as they now look similar to more conventional memory mechanisms. During the supervised learning phase that follows, the coincidental firing between pairs (or higher-dimensional n-tuples) of the ADEs are used to access a memory structure for writing according to certain rules that can be adapted to the particular problem in various useful ways (e.g., choice of class encoding, delays to induce robustness and/or control memory occupancy, biases between classes to improve quality of inference). An equation defines this coincidence mechanism where j & k are indices over the lengths of each AD

$$\forall_{j,k} \text{ if } AD1_j \cap AD2_k \rightarrow \text{set } SBC_{jkl} \text{ where } l = class \tag{3}$$

During inference, the ADEs are driven by test cases and the now populated memory is read using these same coincidence patterns and a simple function of the count of active memory location by class is used to make a class prediction.

For example, imagine a 2D memory with different ADs along the row and column edges. Each ADE in these ADs connects to a different subset of the input data and has learned a different feature. Typically, the width (i.e., the size of subsample ≡ number

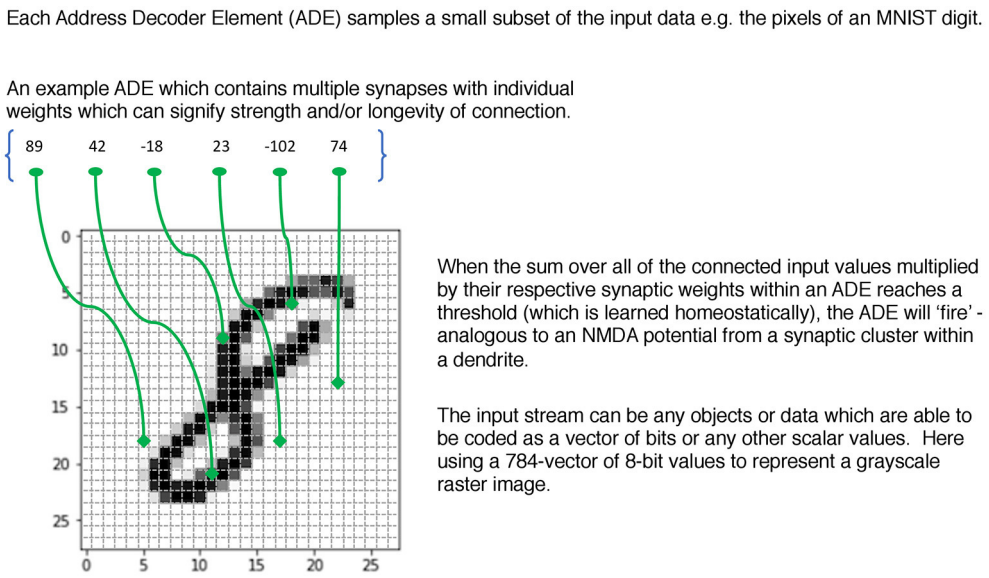


FIGURE 1
An example ADE subsampling a greyscale raster input from MNIST.

of synapses in the cluster) of the ADEs would be different between ADs but the same within an AD. This allows the ADEs within an AD to learn features of similar sizes, whereas those in different ADs learn features of different sizes, perhaps analogous to pooling or convolutional nets of different scales. The upper panel of Figure 2 illustrates this mechanism for a single 2D SBC memory.

Each point formed by the coincidence of 2 ADEs is an accessible region of memory. This region stores a set of the number of classes required in the input data using an encoding appropriate to the problem. The simplest mechanism is "one-hot" encoding. This is illustrated in the lower panel of Figure 2 for 10 classes, each represented by a unique bit in the memory "depth".

4. A sample *BitBrain* implementation

This section gives an overview of the processing steps required for a basic *BitBrain* implementation. This should just be seen as a bare-bones description to clarify what is required.[2] In Section 7.1, we outline a number of interesting variations that we already have some experience with and there will certainly be other novel developments as the technology matures. To fix ideas, we will assume that the data input is a 784D vector as used in the MNIST and EMNIST examples given in the next section.

4.1. Using the global data distribution

Firstly, the global distribution of data over the input vector is calculated over the training set. This is as simple as summing every pixel value into one of 784 bins whilst ensuring no overflow. This vector is then passed to a Metropolis-Hastings (M-H) sampling algorithm [see, for example, Section 5.5 of Bernardo and Smith's book (Bernardo and Smith, 2000) or Section 8.71 of O'Hagan's book (O'Hagan and Kendall, 1994)] which is a reliable method for drawing pseudo-random samples from an arbitrary distribution—in this case generating synapse connections into the input space.

Although simple to achieve, there is no need to normalise this vector to a genuine probability distribution because the M-H algorithm that works with it only requires relative probabilities. In fact, it is not the actual global distribution that is used here but the *sqrt*() of the distribution. There are two possible reasons why this appears to work optimally on problems that have been explored so far:

1. These can be seen as counts and therefore each bin has a Poisson distribution. This means that the uncertainty rises with the mean. The *sqrt*() of a Poisson distribution is approximately homoscedastic above very small counts i.e., the uncertainty becomes independent of the mean.
2. By taking the *sqrt*() of the distribution we are "flattening" it and therefore allowing synapses to be sampled slightly outside of the training data distribution. There are good reasons to believe that this is a good idea for working with data not yet seen in the training set.

It is possible that for data other than greyscale pixels different transformations may be useful, or perhaps a different approach altogether may be preferable at this stage.

4.2. Initial AD and ADE setup

We choose w for the length of the ADs, let's say $w = 2{,}048$. This will also define the size of the SBC memories. Larger is typically better but slower and there are diminishing returns beyond a

2 An implementation of this basic mechanism in self-contained C source code with the necessary data files for the example in Section 5.1 is available here: https://doi.org/10.48420/c.6331565.v1

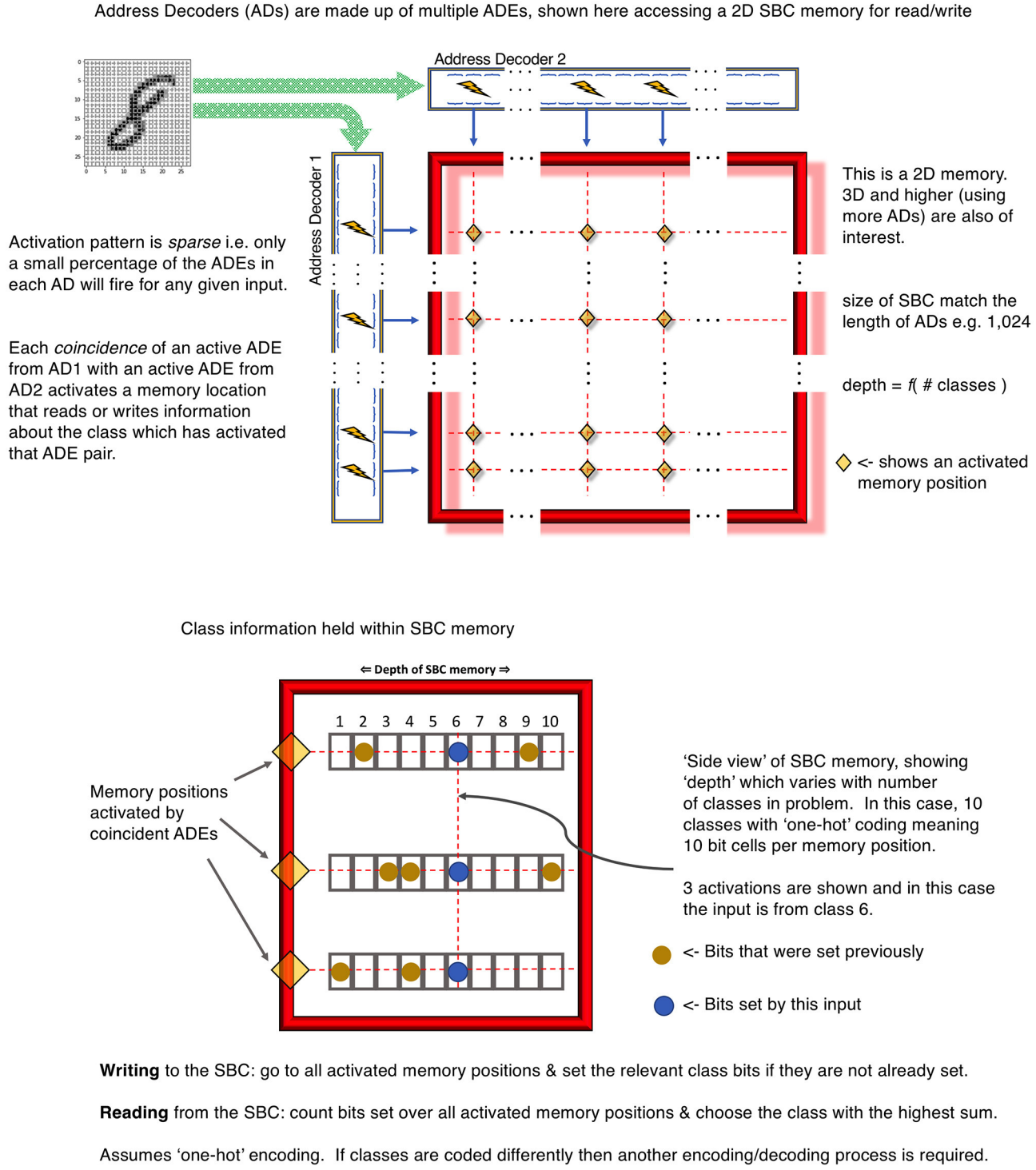


FIGURE 2
Upper panel is an example 2D SBC memory using 2 ADs. Lower panel is an example class encoding using the "depth" dimension of the 2D SBC memory.

certain point. This is likely to differ between problems. An initial default value for the threshold is set for each ADE. These will be adapted as necessary during the unsupervised learning phase.

Each AD is likely to contain ADEs of the same width n (i.e., the number of synapses in the ADE) but typically n will differ for each AD. This allows each AD to work with features of different sizes which is useful in image processing and may also be useful with other data types. Each synapse is assigned to one pixel of the image by drawing from the global target distribution—calculated above—using M-H sampling. Hence, pixels which appear more in

the data are more likely to be sampled and pixels in the corner with almost no "ink" and no variation between images and classes (and therefore no information) will almost certainly not be sampled. Currently multapses (where more than one synapse per ADE connects to the same pixel) are disallowed.

4.3. Choice of synapse types and data

There are several choices available here; we will describe two current ones. One can use binary synapses (i.e., either positive or negative) and their weights then relate to the "longevity" of the synapse which is used during the unsupervised learning phase described below. Initial values of longevity are also set here. Alternatively, one can use 8-bit signed weights drawn pseudo-randomly from some distribution such as uniform or Gaussian which are used to multiply the pixel values.

4.4. Preprocessing of the pixel data

In all cases, the pixel greyscale values are centred either side of zero by subtracting 127 from the original 8-bit unsigned values. This provides a bipolar input which can then be multiplied by the weights described above. Early versions of the algorithm discarded all greyscale information by using binary synapses and thresholding the greyscale inputs. The loss in performance was very small and this version is a more natural match to event-based inputs that are becoming increasingly common in neuromorphic sensors.

4.5. Clustering of synapses for image and related data

In the case of image or volumetric data it is likely that enforcing a locality constraint will be useful (Dahmen et al., 2022) so that pixels chosen in each ADE must be from the same part of the image. To enforce this, if one draw of M-H sampling does not conform to this constraint then another draw is made until one is found. This can be justified either from a knowledge of the organisation of the retina (Masland, 2012) or by analogy with convolutional front ends in DNNs. This distance may be calculated from any of the other pixels in the ADE which then allows feature detectors of various shapes, or else it can be calculated from a centroid which encourages spherical feature detectors.

4.6. Unsupervised learning

To establish a simple form of homeostasis and sparsity as discussed in Section 2 we run through the training set (either in order or, preferably, drawing randomly from it so as to avoid order biases) and, for each training example, establish the ADEs that fire. If drawing randomly we can continue for more than the number of training cases. We accumulate the number of firing events per ADE and after an interval t (perhaps 2,000) compare that to the target number of firing events, e.g., 1% of the cases. If it is too high or low we increase or decrease the threshold accordingly. The end point is a threshold which ensures ≈1% firing on average.

During this process we can also carry out a simple Hebbian learning mechanism per synapse within each ADE. One version is related to a simple idea first explored by Hopkins et al. (2018) inspired by NMDA plateau potentials in a synaptic cluster. If the ADE fires then the smallest contributor to the sum which led to the threshold crossing has its longevity decremented by 1 and the largest contributor to the sum has its longevity increased by 1. After an interval t, if any single synapse has a longevity below a critical value it is replaced using the same mechanism as was used in the original setup of the ADE (as described above); a new pixel is chosen and its longevity is reset to the default value. This allows each ADE to home in on a feature. Neither this nor the threshold learning requires class information, hence this phase is "unsupervised".

4.7. Supervised learning

Now that we have all the ADs setup we can carry out the supervised learning using class information. First there is the choice of SBC architecture. In all cases 2D SBCs with one-hot class encoding are currently assumed. The number of SBCs can be chosen, each using either a pair of distinct ADs or using the same AD for both its row and its column decoder. For example, with 4 ADs each of different width n_i there are $\binom{4}{2} = 6$ SBCs where the AD on each row and column is a distinct combination. These SBCs recognise coincidences between ADEs of different widths, where the feature sizes differ. In addition, there are 4 possible SBCs that recognise coincidences between ADEs of the same width. These are half-size SBCs because only one half of the off-diagonal elements of the SBC are describing unique coincidences. An intelligent implementation will fit two of these half-size SBCs into the storage for one full-size SBC. So for example, the first 6 will be AD_1 * AD_2, AD_1 * AD_3, AD_1 * AD_4, AD_2 * AD_3, ... , and the last 4 will be AD_1 * AD_1, AD_2 * AD_2 etc. Other good results have been obtained using a simpler setup: 3x ADs with (6, 10, 12) synapses placed randomly at (+/−2, +/−3, +/−4) in x and y relative to a centroid chosen by M-H sampling. 3 SBCs are used, each using a different pair of the 3 ADs with no half-size SBCs.

The SBCs are now populated using a simple single-pass supervised learning mechanism that lies at the heart of the method. In a single pass through the training data all coincidences between ADEs cause the respective class bit to be set in the SBCs. A specific class bit may be set many times by different training examples, with the same outcome as if it were set only once by one training example. After a single pass through the training set, supervised learning is complete. An additional pass would, in any case, have no additional effect on the SBC contents unless training noise was being added to increase robustness as described in the next section though the differences are likely to be very small in realistic cases.

4.8. Inference

The inference and supervised learning mechanisms are very similar, accessing the relevant SBC locations in exactly the same

way, setting respective class bits during training but instead counting the set bits during inference. For inference an input case is acquired from a test set and presented to the ADs. For each coincidence between ADEs all class bits are read from the corresponding location in the relevant SBC, and the number of bits set for each class is summed across all SBCs. The highest sum indicates the inferred class of this test case.

If the classes are not one-hot encoded then there is an extra decoding phase required here to identify the most likely class from the accumulated bit counts.

5. Example results using MNIST and EMNIST

Some results are given below for two standard classification problems, the first very well-known and relatively easy, the second less so. They both provide input data as greyscale raster plots of handwritten digits/characters and require the correct classification over a test set once the training set has been digested by the learning mechanism.

In both of these cases, the basic *BitBrain* algorithm with one-hot class encoding, as described in the previous section, has been used. Improvements are possible using spatial jitter at the training stage, a technique termed *data augmentation* in the machine learning literature, but for simplicity and clarity we present raw results here. The test setup for these results is 4 ADs with different ADE widths {6, 8, 10, 12} where the subsampling pattern for each ADE is spatially clustered. The ADs each contain 2,048 ADEs and there are 10 2D SBC memories; 6 of which are full-size 2-way coincidences between different ADs, the other 4 being half-size memories containing coincidences within one AD as described above.

In each case we present results for varying amounts (including zero) of noise added independently per pixel during the training and/or testing phases. This noise can take one of two forms: Gaussian noise of the specified SD with maximum and minimum clamped at 255 and 0 respectively, and "Salt and Pepper" noise with a given probability of a pixel being replaced with 0 or 255, with each of these values being equally likely. Zero noise for both training and test is comparable to standard results. Noise added to the test set simulates imperfect inputs. Noise added to the training set helps to make inference more robust to test noise as can be seen in the plots.

For *BitBrain*, the uncertainty due to different random number seeds can be assumed at ≈0.1% on the Y axes which is too small to be represented by error bars so the thickness of the lines is a good guide. This represents another form of robustness for the learning process itself.

5.1. MNIST

This is the standard MNIST problem and data set (Deng, 2012) using 60,000 training images labelled with the 10 digit classes and 10,000 test images. In Figure 3, we give two views of the robustness performance of the setup described. In order to optimise expected performance in a real-world application a view must therefore be taken on the quality of the input data likely to be encountered. Perfect input data is unlikely in any realistic scenario (unlike benchmark testing), and this graceful degradation in real-world usage is one of the primary drivers for our interest in these ideas.

To give some idea of inference speed for this problem, an implementation of *BitBrain* was set up with 3 ADs each containing 2,048 ADEs driving 3 full-size 2D SBC memories, requiring 16 MB of memory for the total SBC footprint. Running on a 2020 MacBook Air laptop with a 3.2 GHz Apple Silicon CPU this took around 7 s for supervised training on 60,000 examples and 0.42 s for the 10,000 test inferences, delivering 96.6% accuracy with no training or test noise. This was single-threaded C code on the default compiler with no attempt to optimise beyond good coding practice, and no use was made of the GPU.

It is instructive to compare the robustness performance against some representative CNNs which represent a technology designed expressly for such image classification tasks. *LeNet-5* was an early breakthrough and reference designed for handwritten digit recognition (LeCun et al., 1998) which performs to a similar standard to our default *BitBrain* setup in the presence of noise-free inputs. *Efficient CapsNet* (Mazzia et al., 2021) is very recent and arguably close to state-of-the-art, so therefore a very challenging comparison.

Efficient CapsNet models were trained for a maximum of 100 epochs with ReLU activations, while the training setup for *LeNet-5* was a maximum of 100 epochs and sigmoidal activations. To save computational effort we used Tensorflow and some simplifications to the weight setup and training schedule. Also, there is no canonical Tensorflow implementation and the original *LeNet-5* paper uses a 32 × 32 version of the MNIST data. Together these are reasons why our noise-free results are not quite as good as the original results, but perhaps more importantly in this context they are comparable to *BitBrain* on noise-free data. Despite these small differences, we are confident that the important trends over training and test noise values will be unaffected.

In all cases we apply the same noise pattern per training image which is then frozen over training epochs. We call this *static* noise and the aim is to try and provide a fair comparison with *BitBrain* because during our key supervised learning phase each image is (in this study) only seen once and therefore contaminated with only one realisation of the noise distribution. This is not necessarily the case during our unsupervised learning phase, however we believe this is a secondary consideration. In any case, during a small number of test runs we have found that *LeNet-5* results were not significantly improved using *dynamic* noise where a different noise pattern per image is produced for every training epoch.

For the *LeNet-5* results there is considerable variation in the accuracy results with different random number seeds. We believe this is the combined effect of differing startup configurations and noise distributions over the training images. As a result, we have shown mean and SD error bars for these results from a small number of independent runs. It is worth noting that this sensitivity to setup conditions is another form of non-robust behaviours independent of the one that we are aiming to test here, but with its own practical implications. For *Efficient CapsNet* we have limited runs available due to time constraints; indicative error bars are given but these are less precise than those for *LeNet-5*. Figure 4 compares *BitBrain* and the two CNNs on the same Y-axis with the MNIST data set and Gaussian noise.

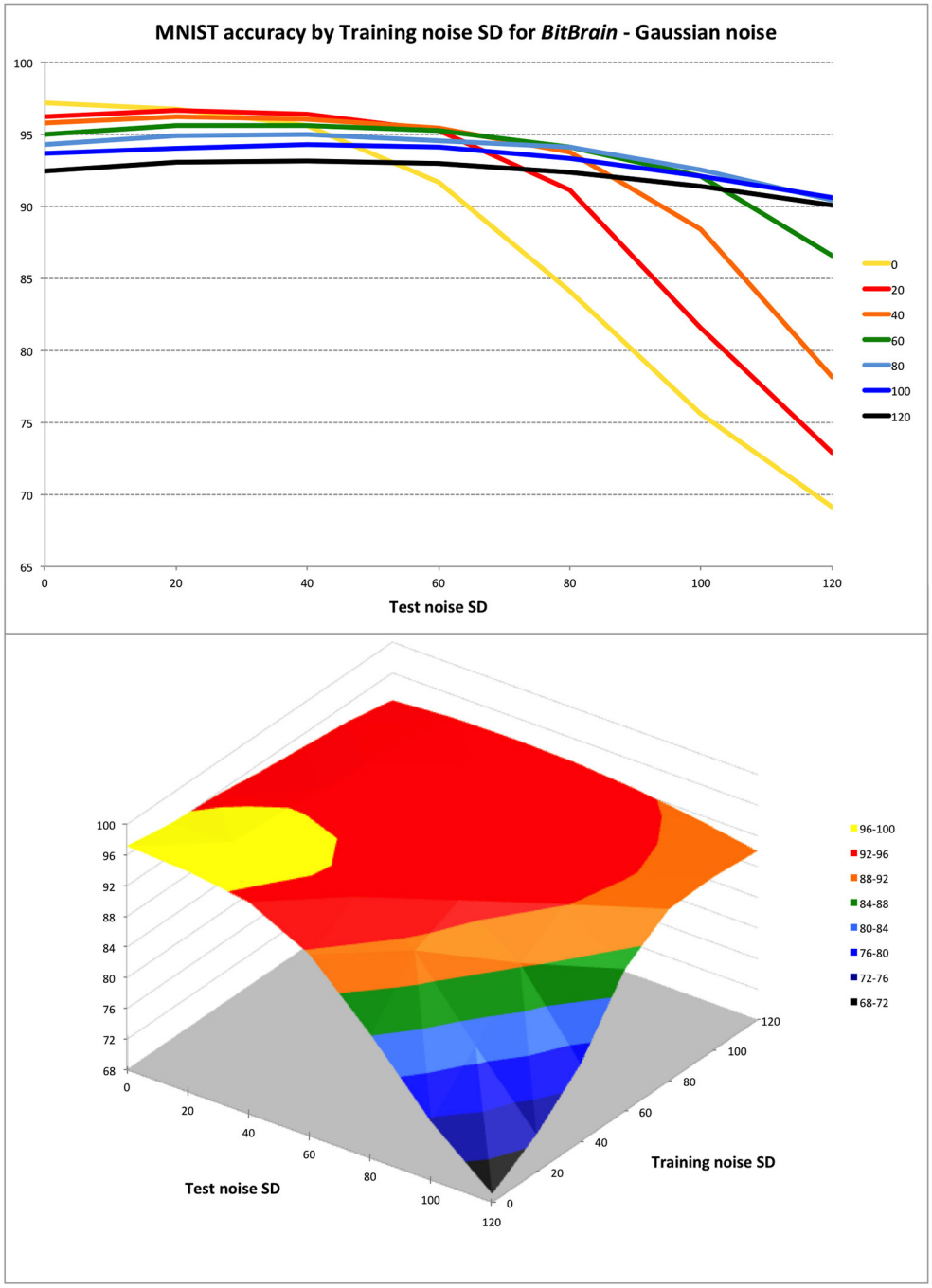


FIGURE 3
BitBrain performance against training and test Gaussian noise levels. Training noise SDs indicated in the legend for top plot and on second axis in surface plot. The trade-off between inference performance with perfect and noisy test data is clear. To perform well with inputs that are very noisy or otherwise imperfect, a small penalty must be accepted with perfect input data by training with appropriate amounts of noise. Over this range of test noise a training noise of 40–60 SD seems to be a good compromise; not penalising performance badly with perfect data whilst protecting against degradation with quite large quantities of test noise.

Clearly *LeNet-5* suffers badly in the presence of noise here but with an interesting pattern of the best test noise performance matching the same training noise setting, as if it has learned to recognise the appropriate signal-to-noise ratio. This pattern is even clearer in the middle panel of Figure 6 and has also been observed in independent work using a different CNN and where modifications of the training and test sets have been distortions other than noise (Adithya, 2022). This is suggestive of another kind of overfitting where the CNN is only learning to recognise data with one particular signal-to-noise ratio or contrast level, and is therefore lacking inferential robustness in realistic real world scenarios. It may be that this is a fruitful area of investigation

10.3389/fninf.2023.1125844

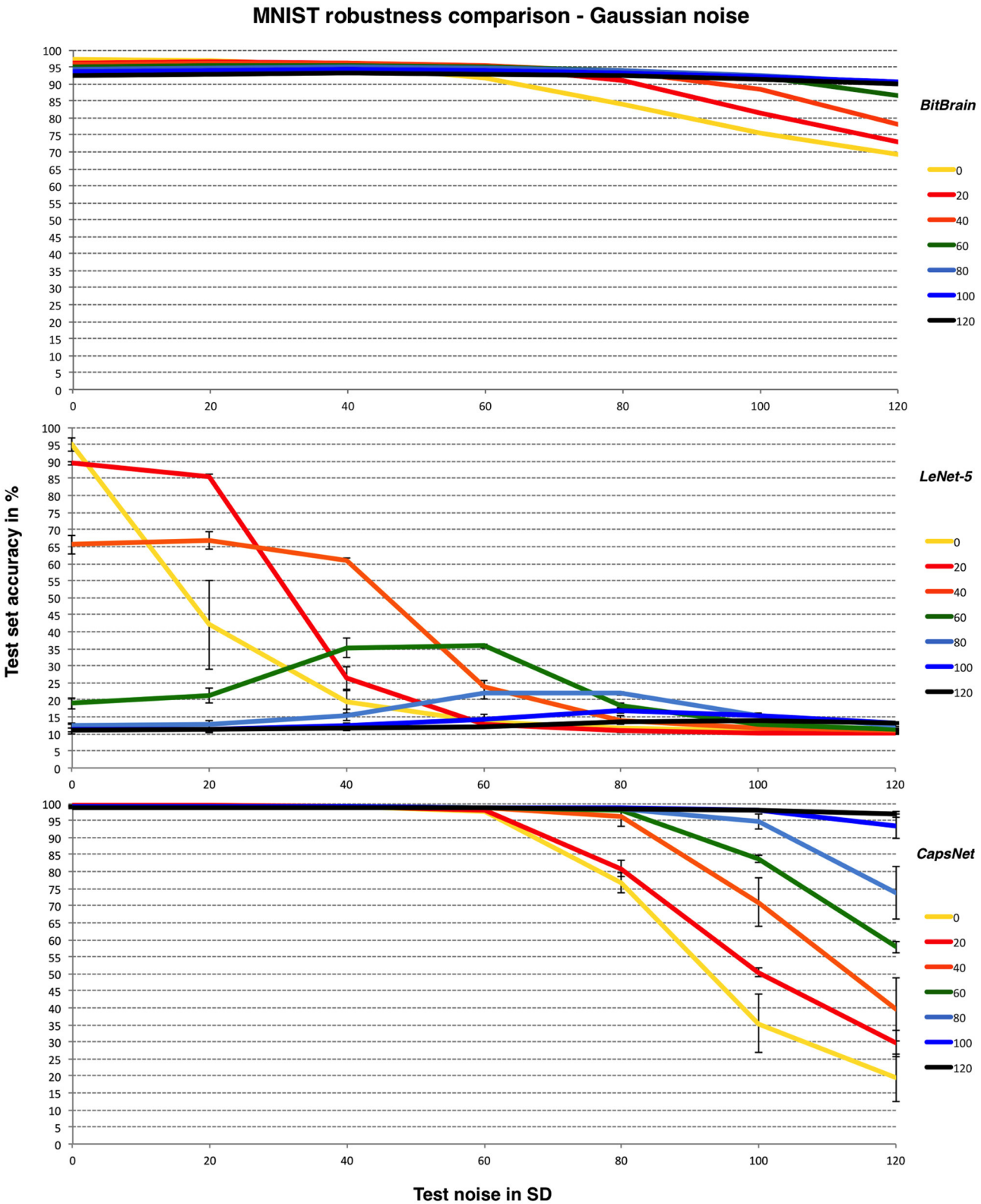


FIGURE 4
BitBrain **(top)** vs. *LeNet-5* **(middle)** vs. *Efficient CapsNet* **(bottom)** robustness comparison for MNIST and Gaussian noise with bounded pixels. Training noise SDs are the lines identified in the legend, test noise SD is on the X axis. *LeNet-5* results are mean and SD error bars from 8 independent runs. *CapsNet* results are mean and SD error bars from 3 independent runs.

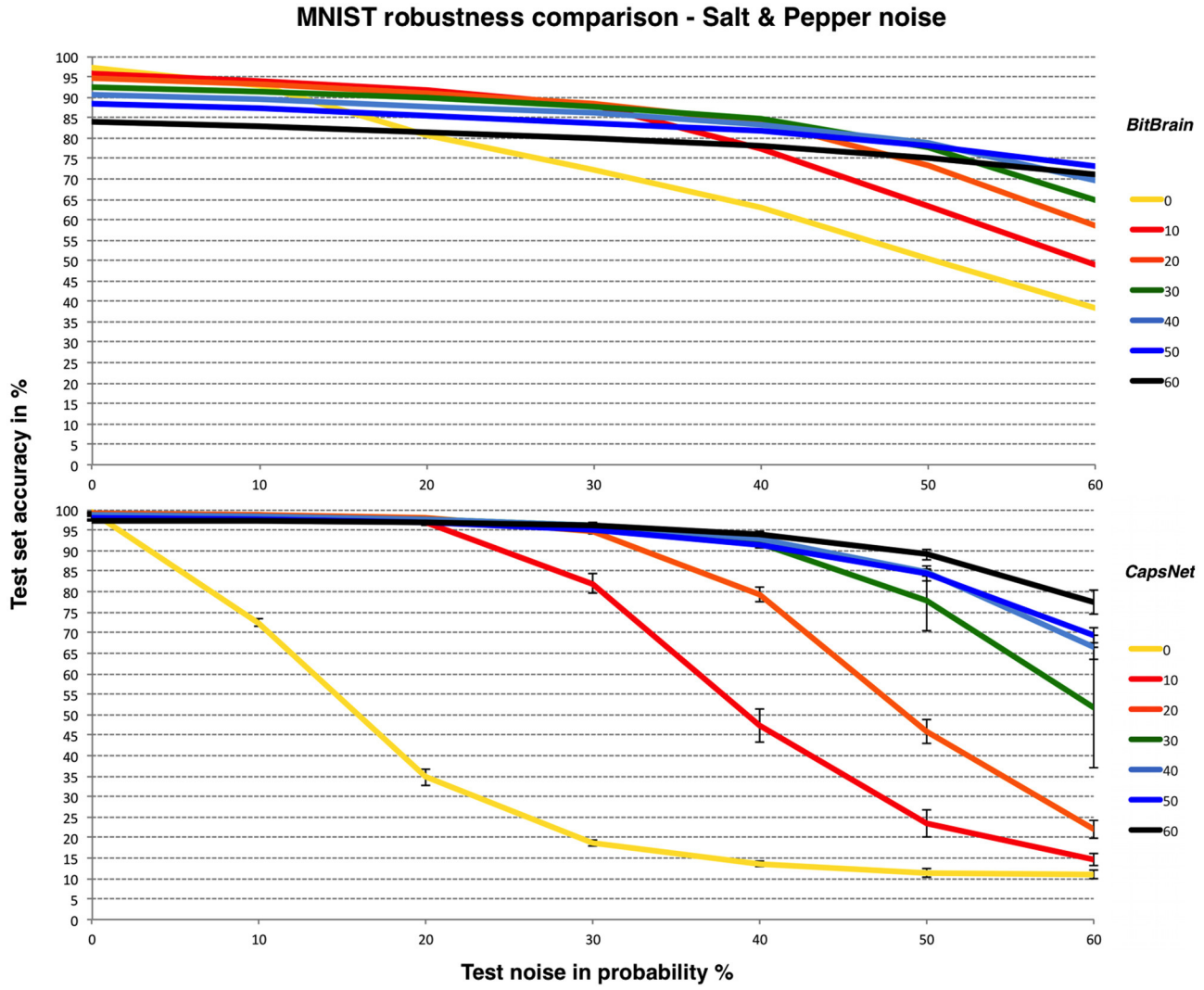


FIGURE 5
BitBrain **(top)** vs. *Efficient CapsNet* **(bottom)** robustness comparison for MNIST and Salt & Pepper noise. Training noise probabilities are the lines identified in the legend, test noise probability is on the X axis. *CapsNet* results are mean and SD error bars from 3 independent runs.

for future study. *CapsNet* is far more robust and in fact responds very well to high values of training noise. Presumably this acts as an effective regulariser which may be an interesting discovery. *BitBrain* is the least affected by different amounts of training noise at higher test noise levels but does not quite reach the accuracy levels of *CapsNet*.

We thought it would be interesting to show a similar comparison between *BitBrain* and *CapsNet* for a different type of noise: "Salt and Pepper" as described at the start of this section. *LeNet-5* appears unable to produce consistent results with this form of noise over these ranges. This is shown in Figure 5.

5.2. EMNIST

EMNIST (Cohen et al., 2017) is a problem similar in nature to MNIST (i.e., 28 × 28 raster plots of greyscale digitised handwritten characters) but much more challenging. All digits and lower- and upper-case characters are used in the most comprehensive *by_Class* data set where the 62 classes are significantly unbalanced and several characters effectively alias each other, e.g.,

{o, O, 0}, {i, I, l, 1}, {s, S, 5}, {B, 8}

Figure 6 provides the results along with the CNNs as in the previous subsection.

Here we achieve results significantly better than the original results for noise-free operation (Cohen et al., 2017) with our basic mechanism, though more recent work has moved the achievable bound upwards by a few percent over ours (Baldominos et al., 2019) as can be seen from the CNN results here. The trade-off between noisy and noise-free test data here is clearer and accuracy generally much lower due to the nature of the problem. Again, adding an appropriate amount of training noise protects the initial performance effectively across a wide range of test noise though with a greater penalty for noise-free data.

10.3389/fninf.2023.1125844

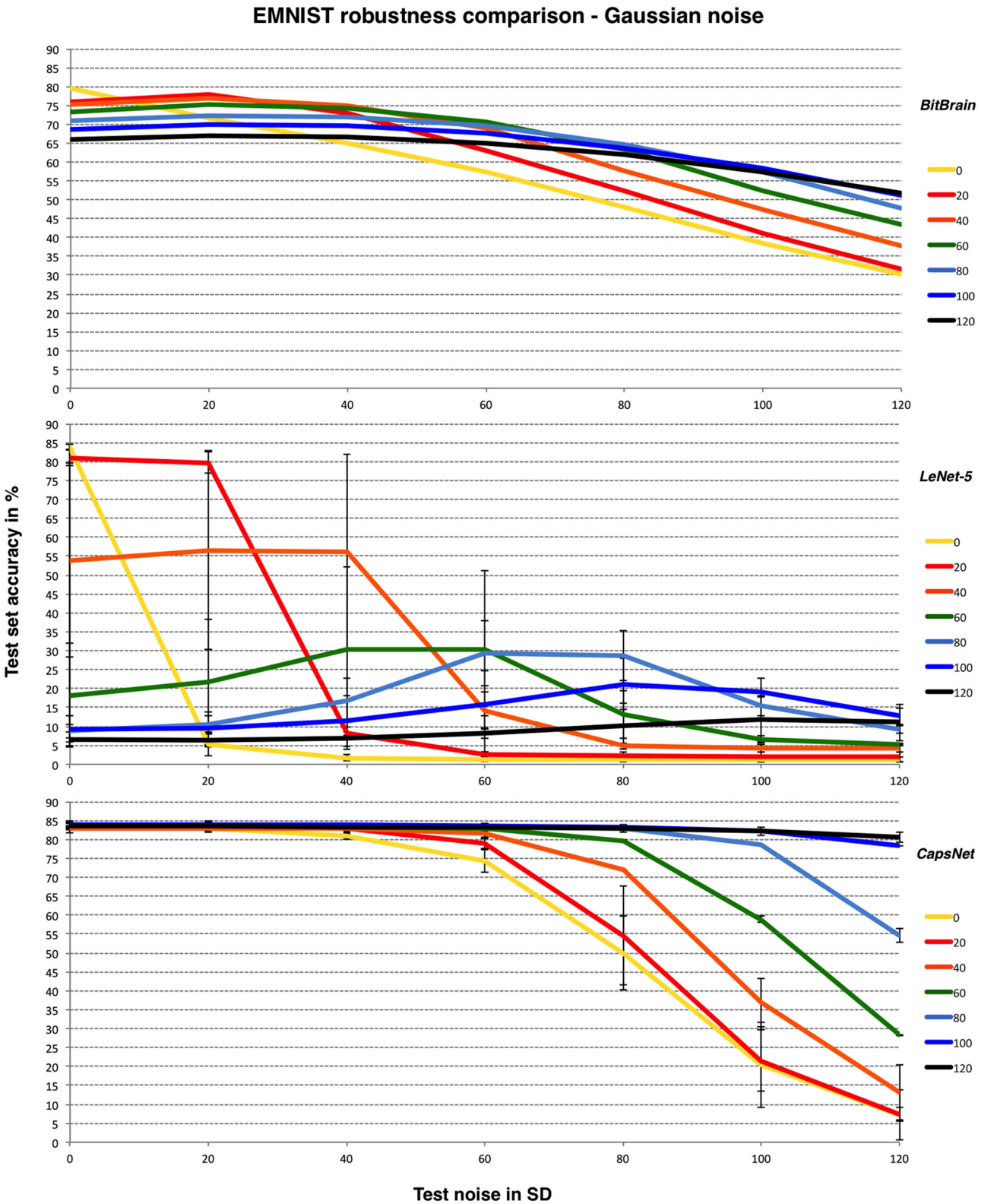


FIGURE 6
BitBrain **(top)** vs. *LeNet-5* **(middle)** vs. *Efficient CapsNet* **(bottom)** robustness comparison for EMNIST and Gaussian noise with bounded pixels. Training noise SDs are the lines identified in the legend, test noise SD is on the X axis. *LeNet-5* results are mean and SD error bars from 6 independent runs. *CapsNet* results are mean and SD error bars from 3 independent runs.

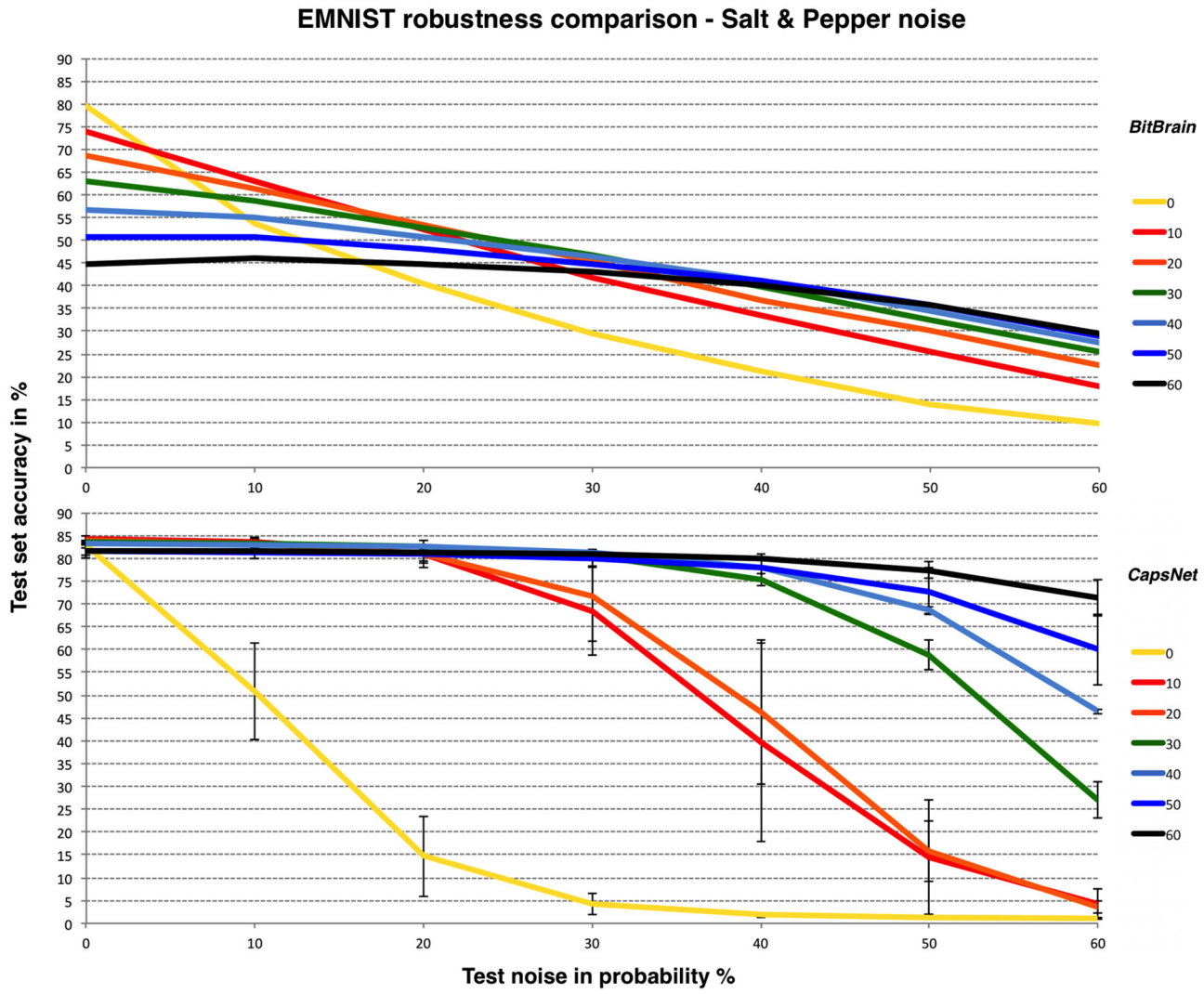


FIGURE 7
BitBrain **(top)** vs. *Efficient CapsNet* **(bottom)** robustness comparison for EMNIST and Salt & Pepper noise. Training noise probabilities are the lines identified in the legend, test noise probability is on the X axis. *CapsNet* results are mean and SD error bars from 3 independent runs.

Similar patterns are observed here. *LeNet-5* suffers most and has very high variability, *BitBrain* is least affected by different training noise settings at high test noise and *CapsNet* again performs very well with high training noise. Despite being completely unrelated technologies, both *BitBrain* and *CapsNet* respond well to high levels of training noise. We again compare *BitBrain* and *CapsNet* using Salt and Pepper noise in Figure 7.

The EMNIST problem combined with Salt & Pepper noise at these levels is obviously a significant challenge for both technologies, though again *CapsNet* with high levels of training noise performs very well.

5.3. Comparison with other single-pass ML methods

In this section, we present a summary of results from the ML literature about single-pass learning, where each sample from the training set is used only once and is not stored in memory. We investigate how *BitBrain* in its current form compares to a number of natively single-pass approaches (Wang et al., 2012, 2013; Zhou et al., 2016), well-established deep neural networks (He et al., 2016; Mazzia et al., 2021), and two simple CNNs with one and two convolutional layers trained with just a single epoch.[3] We continue to use MNIST here as the results are widely reported.

Although the state-of-the-art modern machine learning models often rely on deep networks which are successively trained over many epochs, simpler approaches which need only a single pass through the training set are still of interest to the community, especially in applications with limited resources. These single-pass approaches typically employ a form of online learning which allows them to process large datasets without the need for excessive computational resources. One particular approach—local online

3 CNN A—https://github.com/jiuntian/pytorch-mnist-example and CNN B—https://github.com/ya332/Simple-CNN-for-MNIST

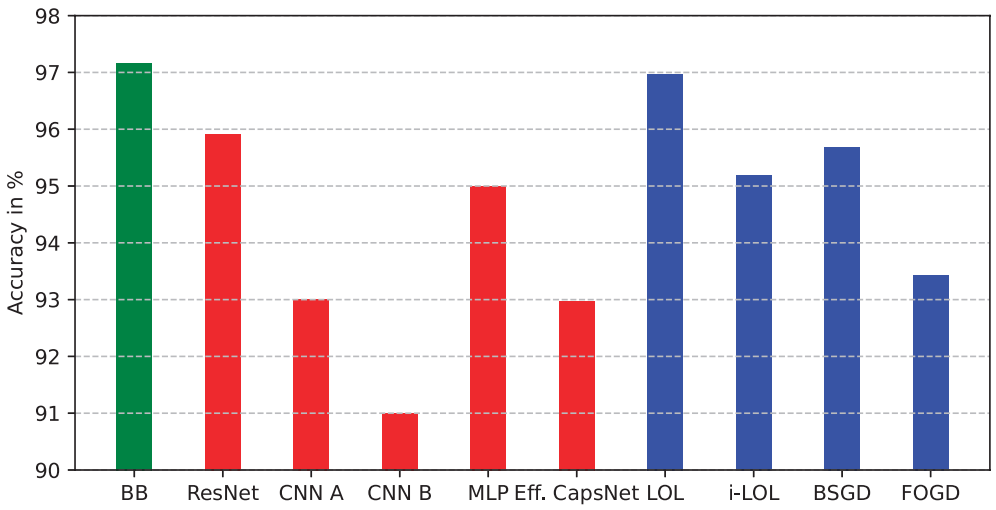


FIGURE 8
Performance comparison between *BitBrain* (green bar), deep neural networks (red bars), and natively single-pass learning approaches (blue bars). Accuracy in % on the test set.

TABLE 1 Two-class results for accuracy in % compared with *BitBrain* from Table 1 of Rai et al. (2009).

Task	libSVM	Perceptron	Pegasos1	Pegasos20	LASVM	StreamSVM1	StreamSVM2	*BitBrain*
0 vs. *1*	99.52	99.47	95.06	99.48	98.82	99.34	99.71	99.95
8 vs. *9*	96.57	95.90	69.41	90.62	90.32	84.75	94.70	98.49

The relevant parts of their table are replicated here with the best result for each case in red.

learning (LOL) (Zhou et al., 2016), proposes an extension of commonly used Passive-Aggresive (PA) method (Crammer et al., 2006) which updates the classifier sequentially based on the feedback from each data point in the training set. Unlike the PA and related approaches, the LOL allows for learning multiple local hyperplanes to non-linearly process sequential data in a one-pass manner. The authors also introduced a novel optimisation strategy which significantly improves the performance on classification tasks with multiple classes of patterns compared to previously proposed methods.

Figure 8 shows the single-pass performance of deep learning methods (red bars) and natively single-pass methods (blue bars) in comparison to *BitBrain* (green bar). The performance of *BitBrain* is visibly better than deep learning methods, which significantly underperform when trained with just a single epoch, as well as the online single-pass approaches. Notably, natively single-pass approaches also provide a better classification accuracy than more commonly used convolutional neural networks, however this discrepancy is likely to result from the fact that the training hyperparameters of the CNNs have not been adjusted adequately in such a limited training time.

Another single-pass comparison can be made with online methods for SVMs in two-class problems (Rai et al., 2009). In Table 1, results are provided from MNIST for discriminating *0* vs. *1* and *8* vs. *9* using a number of different algorithms.

5.4. Single-shot performance

In this section, we present some results that show learning performance as a function of n for MNIST and EMNIST with very small training sets from $n = 1$ per class upwards. These are shown in Figure 9 where error bars are one standard deviation from 10 repeats with different randomly generated subsets of the training cases. These results show that training data sets far smaller than are common in current machine learning applications can be useful in terms of generating inference accuracy well beyond chance. This will have implications for where *BitBrain* can be applied. It is also worth noting that the error bars are very small, even for $n = 1$. This indicates another aspect of robustness demonstrated by the *BitBrain* mechanism because it hardly seems to matter which training cases are chosen and anyone familiar with the MNIST and EMNIST data sets will know that the training cases can vary substantially.

6. Relationships to Kernel methods theory

BitBrain is a new idea and the underlying theory has to be developed further in order to catch up with the empirical results. This will help guide future directions for research and improve practical results and implementations. In Section 2 of the main document we discuss ideas from a number of fields which have informed this technology. In this section, we want to explore one of them further.

Kernel methods are based upon a matrix which is created by the similarities between data points in the training set. This is called the *Gram* or *Kernel* matrix which we will call $\boldsymbol{K}$. If there are n training data this matrix will be n x n positive definite symmetric (PDS) with "self-similarities" (however that is defined) along the diagonal and

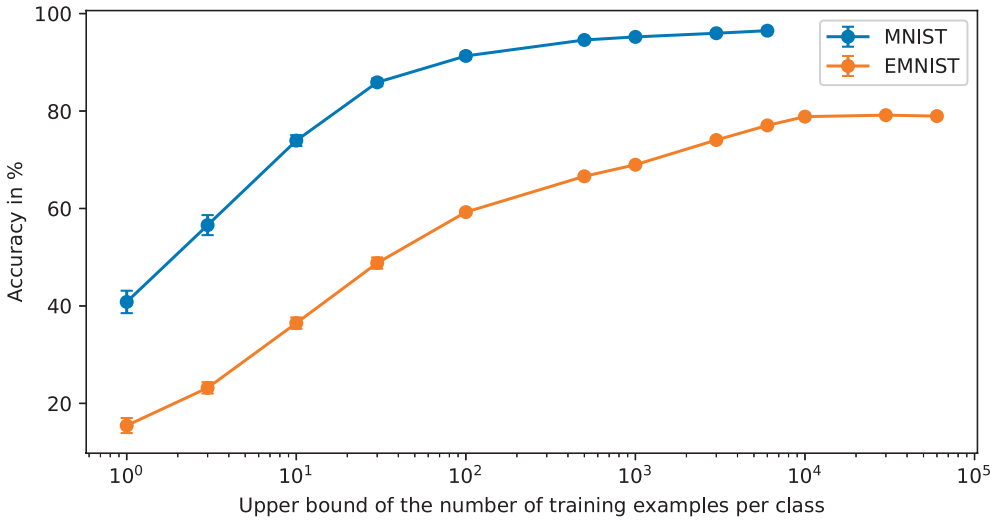


FIGURE 9
Accuracy as a function of the upper bound of the number of per class training examples. The X axis is less obvious for EMNIST because as we move further to the right some of the under-represented classes will have their training sets exhausted whilst other classes are still being subsampled. It may not be obvious but beyond 10,000 training examples on EMNIST the accuracy falls very slightly because the over-represented classes are still being added into the SBCs when there are none of the under-represented class training examples left which exacerbates the unbalanced nature of the data set.

all the off-diagonal entries being the similarities between different training data cases.

Similarities can be defined in many ways, the primary constraint being that they must generate a PDS Kernel matrix. A standard description in the kernel methods literature is that the elements of $\boldsymbol{K}$ are formed by dot products between features so that $\boldsymbol{K}_{ij} = < \phi(x_i), \phi(x_j) >$ where $\phi()$ is an arbitrary function that maps the input data x into a corresponding feature space. The choice of $\phi()$ is therefore key in order to make any given method appropriate for the data involved.

The "kernel trick" which provides potentially very large computational benefits for kernel methods is to find a closed-form function $k(x_i, x_j) = < \phi(x_i), \phi(x_j) >$ without having to calculate the (perhaps very high-dimensional) dot products required explicitly. A good example from Gaussian Process methods is the covariance function between two points in input space which can be of a very simple closed form whilst at the same time (i) guaranteeing a PDS Kernel matrix and (ii) expressing a very high-dimensional feature space that can be parameterised and adapted easily but which never needs to be explicitly calculated (Rasmussen and Williams, 2006).

6.1. A simple multi-class Kernel-based classification (KBC) method

Probably the most straightforward KBC method that can be applied effectively to problems with multiple classes is called *Least-squares Classification* (LSC) (Rasmussen and Williams, 2006) and various versions are compared by Rifkin and Klautau (2004). In its simplest form, assume that $\boldsymbol{K}$ is formed from the training data and that there are $c = 10$ classes (e.g., for the 10 MNIST digits) with each training case labelled with one of the set {0, 1, ... 8, 9}. Now make c "dummy targets" $\boldsymbol{y}_0$-$\boldsymbol{y}_9$ which are of length n and in each case contain a zero for training cases where the label doesn't match their subscript and a one where it does. So in this case there are about 90% zeroes and 10% ones for each $\boldsymbol{y}_i$. Now generate 10 "hat" vectors $\boldsymbol{h}_0$-$\boldsymbol{h}_9$ of length n which are essentially weights (both positive and negative) used for assessing any new case and which class it corresponds to. The algebra[4] of this is:

$$h_i = K^{-1} y_i \tag{4}$$

So now the class of a new data case can be inferred. For each case form the vector $\boldsymbol{k}_\star$ of length n which gives the similarity (exactly as defined when $\boldsymbol{K}$ was formed between training cases) between the new case and all n training cases. This is like forming a new row/column of $\boldsymbol{K}$. Now create the dot product of $\boldsymbol{k}_\star$ with each $\boldsymbol{h}_i$ to produce c "class indicator" values, i.e.,

$$class\ indicator_i = k_*.h_i \tag{5}$$

In the ideal case this would produce c-1 class indicators = 0 and 1 class indicator = 1, with the index of the latter providing the class inference. In reality, imprecision in the similarity metric, noise in the training and test data and other issues will make these results approximate, but a simple and robust mechanism for class choice is to pick the largest value. This intuitively describes "overall similarity to training cases of class$_i$" in the similarity metric defined by the kernel function chosen.

4 For actual computation it is both faster and numerically more stable to decompose K (e.g., using the Cholesky decomposition) and then solve for y_i using a backsubstitution rather than forming the inverse explicitly and then multiplying. This decomposition only needs to be done once but it is $O(n^3)$ and so clearly for large training sets is a time-consuming linear algebra operation. The solutions to form each h_i are $O(n^2)$ and only need to be done once.

6.2. How does *BitBrain* resemble and differ from KBC methods?

Now think about all possible 2D coincidences from a data input as one long bit vector. For example, assume there are 3 ADs each of length $w = 2{,}048$ and 3 2D SBC memories to capture all the full-size coincidences. "Unrolling" the 2D bit positions from this setup would make one vector, $\boldsymbol{b}$, of length $2{,}048^2$ x 3 = 12,582,912 bits. This ignores the class bit depth, which will be addressed later. The number of possible bits turned on in $\boldsymbol{b}$ ranges from 0 to 12,582,912. Assume a 1% firing probability per AD and a random distribution for the active bits, then the expected number of bits turned on $\approx 20.48^2$ x 3 = 1,258.3.

Now consider two different inputs. These will turn on different sets of bits in $\boldsymbol{b}$. Taking the logical AND of these two vectors is also, of course, a dot product, which can then be written in exactly the form $< \phi(x_i), \phi(x_j) >$ as described at the start of Section 6. This can be thought of as an overlap between bit vectors or an intersection between bit sets. As it can be expressed as a dot product it is a valid similarity measure which will produce a PDS Kernel matrix, with $\phi()$ here being a projection of the image data *via* the ADEs into a sparse 12,582,912D binary space

It may be instructive to try and demystify the $\boldsymbol{h}_i$ vectors somewhat. The upper panel of Figure 10 is a plot from the first 600 MNIST cases, sorted by label so that the first ones are 0s, then 1s, etc. As expected by the form of the $\boldsymbol{y}_i$ vectors, these 'gate' the weightings for their own labels. However, there are subtleties as well such as substantial differences between same label cases (caused by the interrelated relative similarities encoded in $\boldsymbol{K}^{-1}$) and some less trivial negative weights as in case 183, which presumably means that case (which is a **2**) is particularly dissimilar to a/some **1** case(s), hence the negative weight.

Using the above setup and the mechanisms described in Section 4.6 we achieved 98.6% on MNIST, which is the best result so far using these ADEs. Table 2 shows all the class indicator values for two correctly assigned sample outputs. The first is obviously a clear result with good confidence and the second less so.

6.3. The relationship to *BitBrain*

In *BitBrain* the same vector $\boldsymbol{b}$ is formed for every data case but used in a different way. For every training image the class label is stored for every active bit position of $\boldsymbol{b}$ where it is not already set. This assumes that $\boldsymbol{b}$ now has a 2nd depth dimension of length c (for one-hot encoding), or alternatively think of c vectors $\boldsymbol{b}_i$ which are analogous in some sense to the KBC "hat" vectors $\boldsymbol{h}_i$. The second perspective corresponds more directly to the kernel definitions and is used in what follows.

The essence of understanding the relationship between these two apparently distinct methods is to see how the supervised learning memory write mechanism relates to Equation (4), and how the read operation for inference relates to Equation (5). They are clearly performing related tasks, albeit in very different ways. A current hypothesis is that the *BitBrain* mechanism is forming an empirical approximation to the kernel function $k(x_i, x_j)$ described earlier, so that the expensive $\mathbf{O}(n^2)$ and $\mathbf{O}(n^3)$ operations required in Equations (4) and (5) are now converted into memory writes and reads over the training data as a whole.

What is not yet clear is exactly how that relationship can be derived, but perhaps some progress can be made. A starting point is the observation that *Mercer's Theorem* means a kernel function can be appropriately decomposed into a summation over products of orthonormal functions, i.e.,

$$\kappa(x, z) = \sum_{j=1}^{m} \phi_j(x)\phi_j(z) \tag{6}$$

So that, for example, the $\phi_j()$ could be eigenvectors of the $\phi()$ function that defines the feature space. A suggestion is that x is a vector of binary values, $m = length(b)$ and $\phi_j(x) = x[j]$ where [ ] indexes into the vector and therefore returns one of {0, 1}. Kernel functions such as this are described by Shawe-Taylor and Cristianini (Shawe-Taylor and Cristianini, 2004) in Sections 9.5 and 9.7 as set kernels and by Odone et al. (2005) as histogram intersection kernels which turn out to have a natural link to L^1 distance in their Equation (15). Their use in image processing problems is described by Raginsky and Lazebnik (2009). So it is possible that the c vectors $\boldsymbol{b}_i$ are related to this summation, where the summation index is now over their length. As required above, each bit of $\boldsymbol{b}_i$ would by definition be (approximately) orthonormal in a large, sparse binary vector.

6.4. Differences from Kernel methods

Unfortunately, we don't have direct access to the full Kernel matrix $\boldsymbol{K}$ which describes the similarities between all training cases and so cannot explicitly form Equations (4) and (5) which are essential for LSC. However, we have generated a union over all the bit patterns found from the training set and stored them in the relevant $\boldsymbol{b}_i$ according to their class information. It seems that this sampling and storage mechanism has taken the place of Equation (4) and then Equation (5) is being approximated by the overlap of the bit pattern from a new data input which we can call $\boldsymbol{o}_*$ (analogous to $\boldsymbol{k}_*$) and the stored bit pattern in each $\boldsymbol{b}_i$ (analogous to $\boldsymbol{h}_i$).

So what can we say about this? For one thing, in the kernel method the similarities are unambiguously calculated between cases. In *BitBrain* the similarities are calculated between $\boldsymbol{o}_*$ and either the bit pattern projections over $length(\boldsymbol{b})$ or the c classes, depending on your perspective. It may be useful to think about this in terms of set theory. The cardinality of the whole bit set = $length(\boldsymbol{b})$. The subsets for each class are defined by the active bits in each $\boldsymbol{b}_i$ let's define these subsets as $\mathbf{S}\{\boldsymbol{b}_i\}$. The relationship between any new point and the stored training cases for $class_i$ is defined by the intersection of the active bits in $\boldsymbol{o}_*$ with those of $\mathbf{S}\{\boldsymbol{b}_i\}$. We can directly call this a similarity by appealing to Equation (6). This gives us half of Equation (5) but where do we get the equivalent of $\boldsymbol{h}_i$ and how do we deal with the different domain over which the dot product is calculated?

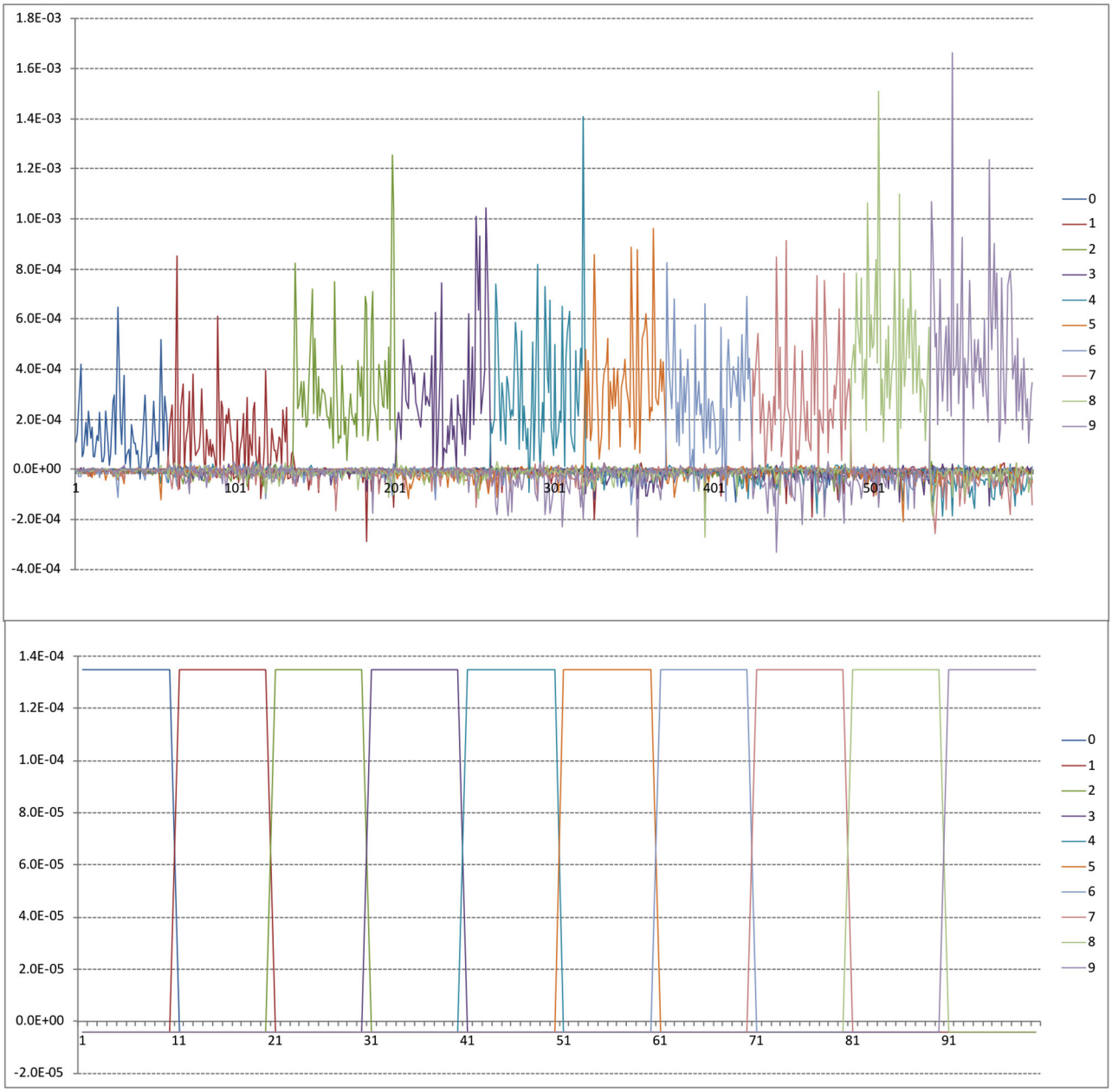


FIGURE 10
Upper panel is sample $\boldsymbol{h}_i$ vectors for first 600 MNIST training cases. Lower panel is synthetic $\boldsymbol{h}_i$ vectors for $n = 100$ as described in text.

To address this question it might be useful to consider the structure of the $\boldsymbol{h}_i$. If in the kernel setup all cases are equally similar to cases in their own class, and equally but less similar to cases in other classes then $\boldsymbol{h}_i$ just looks like a gating variable. A synthetic simulation for $n = 100$ is shown in the lower panel of Figure 10, sorted by class. This was created by a $\boldsymbol{K}$ matrix with 3,000 on the diagonal (i.e., self-similarity or the number of bits activated by a case), 500 off diagonal for cases of the same class and 50 off diagonal for cases of different classes.

It's interesting to compare this with the upper panel of Figure 10 where varying mean similarities by class and within-class similarities between specific cases produce a very complex pattern of weightings. In this case the exact (and unrealistic) uniformity of similarity provides a very clear weighting pattern. Arguably, what we are doing with *BitBrain* is the same as this but gated across c classes or $length(\boldsymbol{b})$ depending on your perspective.

It's also worth bearing in mind that the $\mathbf{S}\{\boldsymbol{b}_i\}$ will intersect with each other in potentially complex ways. An intersection between any number of $\mathbf{S}\{\boldsymbol{b}_i\}$ simply means the subset of memory positions in all the SBC memories where those class bits are all set. Figure 11 gives a matrix plot for the number of bits in each $\mathbf{S}\{\boldsymbol{b}_i\}$ on the diagonal and the two-way intersections between these subsets off

TABLE 2 The class indicator values for two sample outputs.

Digit	ID 3357 is a 5	ID 5680 is a 3
0	0.008482	0.113866
1	0.099251	0.008050
2	−0.003518	0.023137
3	−0.058747	0.414228
4	−0.058338	−0.005619
5	1.087230	0.063122
6	−0.040066	0.010855
7	−0.063182	0.029908
8	0.058970	−0.026898
9	0.039974	0.019438

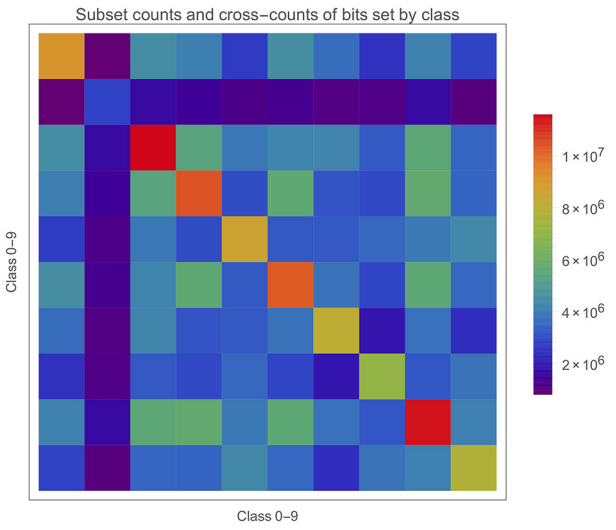


FIGURE 11
Cardinality of subsets and intersections for *length*(***b***) = 33.5M in this case.

the diagonal. The intersections are substantial, but does it matter or is it inevitable? What about three- and higher- (i.e., up to *c*-) way intersections?

We can observe the different cardinalities of each $\mathbf{S}\{\boldsymbol{b}_i\}$ on the diagonal which is particularly low for class **1** and high for classes **2** and **8**. However, directly correcting for this (e.g., a naive multiplication by their inverse ratio) makes prediction accuracy worse, and so the cardinality is not all that matters. One observation is that **1**s might be more similar, leading to fewer bits being set in $\mathbf{S}\{\boldsymbol{b}_1\}$ because the same bits are constantly being activated but cannot be stored more than once. The relatively low values of $\boldsymbol{h}_i$ in Figure 10 for class **1** would tend to support this. This may not matter in inference if the test cases have a similar behaviours and are well-separated in feature space from the other classes. Presumably, "well-separated" is some function of the intersections (of various orders) between all the $\mathbf{S}\{\boldsymbol{b}_i\}$.

For precise inference it would be ideal to have no intersections at all, but minimising them is a much more realistic aim. That would suggest a very clear difference in the bit count values between classes assuming that test cases follow the training case patterns, which is a realistic assumption that is often required in machine learning. Achieving this kind of behaviours where classes of interest are separated as widely as possible is a standard problem in optimal experimental or sampling design (Shewry and Wynn, 1987). However, working out how to get to that position in this problem is not easy. One would need to ensure that entirely different patterns of bits are set by the different classes which would mean consistently different combinations of ADEs firing per class, and it's not currently clear how this could be achieved. Some early work tried learning each $\frac{1}{10}^{\text{th}}$ of the ADEs (i.e., the feature detectors) on digit data distribution separately by class or differences between classes at the unsupervised stage, but this did not provide any significant improvement in prediction performance.

To visualise the intersections of active bits in $\boldsymbol{o}_*$ with those of $\mathbf{S}\{\boldsymbol{b}_i\}$, Table 3 shows the first results from the same setup as Figure 11. The number of active bits in the test case shows that the ADEs are tending to fire at slightly more than the 1% target rate, at least assuming complete randomness.

There are several things of interest here. Firstly the correct class has been chosen each time (this sample is from a 97.12% performance, so not such a surprise). Secondly is that almost all the active bits in a test case intersect with the bits in the relevant $\mathbf{S}\{\boldsymbol{b}_i\}$. Of course this is good but such a high overlap is somewhat surprising and this seem to be quite general across both test cases and classes. Thirdly, the size of overlap with the wrong classes is also much higher than expected. Let's take the first case for example and look at class **9** instead. The overlap with $\mathbf{S}\{\boldsymbol{b}_9\}$ = 9,704. *card* ($\mathbf{S}\{\boldsymbol{b}_9\}$) = 7,859,278 which gives as a proportion of all possible bits a 23.42% occupancy. There are 3,803,472 bits shared between $\mathbf{S}\{\boldsymbol{b}_7\}$ and $\mathbf{S}\{\boldsymbol{b}_9\}$ (see Figure 11) and $card(\mathbf{S}\{\boldsymbol{b}_7\})$ = 7,075,762. So we can say that this two-way intersection is about half of each subset! In other words, for any memory position where one of these class bits has been set there is about a 50% chance that the other is set as well. **7** and **9** are likely to be a worse than average case as they are often one of the higher value pairs in the confusion matrix.

7. Discussion

In this section, we aim to cover some loose ends and discussion points that surround what is basically a simple idea that we have tried to describe in a straightforward way.

Although we have focused on the benefits of *BitBrain* some may wonder what is to be paid for these. One discussion has been about how much data needs to be stored in the SBC memories and that a comparison should be made with methods that use this amount of parameter space. We disagree with this perspective, for the following reasons. We see *BitBrain* as a non-parametric method. There is no definitive

TABLE 3 The first results from the same setup as Figure 11 with classes {0, ... , 9}, *length*(***b***) = 33.5M.

card(*o**)	*class*	0	1	2	3	4	5	6	7	8	9
12,168	7	4,876	2,909	6,529	6,916	7,291	6,398	2,769	12,055	5,295	9,704
21,011	2	13,624	8,495	20,348	16,114	4,068	12,939	14,645	2,930	13,362	3,179
7,320	1	2,551	7,257	5,965	5,115	5,764	5,160	4,531	6,147	5,962	5,010
15,664	0	15,409	1,941	9,625	8,997	5,459	10,297	11,040	7,434	5,724	7,152
14,110	4	6,816	2,625	9,796	6,231	13,919	7,156	8,005	9,553	7,969	11,495
8,597	1	2,442	8,515	6,577	5,184	6,452	4,983	4,287	7,522	6,843	5,335
10,804	4	3,005	3,087	5,594	6,073	9,974	6,007	4,667	6,638	7,824	7,644

card(***o****) is the number of active bits in the test case, *class* is the class for the test case, and the remaining columns show the intersections of the test case with each of the training classes.

description of a non-parametric model but this is as good as any.[5]

> Non-parametric machine learning algorithms try to make assumptions about the data given the patterns observed from similar instances. For example, a popular non-parametric machine learning algorithm is the K-Nearest Neighbor algorithm that looks at similar training patterns for new instances. The only assumption it makes about the data set is that the training patterns that are the most similar are most likely to have a similar result.

Another parallel is found here then with Kernel methods which—expressed in the form described in Section 6—are clearly non-parametric in nature as they use the data directly for their inference, not parameters that have been somehow inferred from the data. There may sometimes be hyperparameters involved (for example the local width of the kernel in Kernel regression) but nevertheless the predictor is formed from a relatively simple function of the data themselves.

BitBrain is clearly working in a similar way, and in fact the only scalar parameters learned are during the unsupervised learning stage where ADE thresholds are found which give an approximately correct average firing probability. These are local 1D searches that are cheap and easy to carry out in parallel and with a well-defined optimum point. There are also some differences, for example it is not the data *themselves* that are used but a high-dimensional projection of them *via* $\phi()$. We don't believe this changes anything though in relation to the fundamentally non-parametric nature of the inference. So the SBC memories are storing "a direct encoding of the data in a form which makes it suitable for inference", not "parameters".

It should also be said that there are versions of the method with a much lower memory footprint. For example, if only memory positions containing a single class bit after supervised learning are saved. In the context of the discussion in Section 6 this means activated parts of the bit space where there are no intersections between class subsets. This excludes the vast majority of memory positions and also means that less space is needed for each. One may achieve 2-4 orders of magnitude saving in storage with a demonstrated small (≈3% on MNIST) loss of inference performance but also a significant gain in inference speed, though hash tables or some other mechanism will be needed for the required sparse storage unless specialised hardware is available and this will somewhat complicate the implementation. It may be that in some applications these trade-offs are fully justified.

Another unknown is the capability of the current and basic implementation on the most challenging problems. We are unlikely to compete with deep networks with billions or trillions of parameters at this point, but in defence of the method it was never designed for this. It remains to be seen how the more advanced versions discussed below fare in such problems.

7.1. Future directions

BitBrain is a very new method and there is a lot left to explore in order to understand and improve behaviours. In this subsection, we briefly describe a number of ideas that are either under consideration or being actively investigated.

It has been observed that various ADEs will tend to fire together more than they should by chance, whereas the ideal would appear to be independence amongst the ADE firing patterns. There are various arguments from both information theory and SDM theory to support this aim, though it is by no means proven. Some mechanism for modifying or replacing ADEs which are too similar would therefore appear to be a useful mechanism during the unsupervised learning phase. Earlier work in unsupervised learning may be relevant here (Atick and Redlich, 1993; Bell and Sejnowski, 1995; Linsker, 2005).

There can be several subtleties and variations for the basic supervised learning phase described in Section 4.7. We might, e.g.,

- Only write a bit probabilistically which will reduce memory occupancy and potentially increase inferential robustness.
- Add noise to the training data to facilitate inferential robustness (as seen in Figures 3–7).
- "Jitter" or otherwise augment the training data with elastic deformations and rotations.
- Use *N-of-M* codes within each memory location to specify the class encoding and decoding.

5 https://deepai.org/machine-learning-glossary-and-terms/non-parametric-model

- Enforce a strict *N-of-M* code for each AD activation by choosing the *N* ADEs that exceed their threshold by the largest amount.

There will be weightings of the class bit sums which perform better than the default (uniform) case. The simplest weighting is to element-wise multiply by a vector the same length as the number of classes. A more flexible weighting which can take into account—and perhaps correct for—complex inter-correlation patterns between the classes is to post-multiply the class bit counts by a square matrix and use the resulting vector for class choice. In either of these cases, the challenge is to know what this vector or matrix should be!

Two types of continuous learning are being considered. Firstly, dealing with data that changes over time: perhaps the degradation of a sensor input device or a genuine change in patterns in the system of interest. As new labelled data arrives over time these can be added to the SBC memories as in the supervised phase. A count would be kept of the number of bits added and at a chosen interval this same number of bits are randomly removed from the SBCs. This will maintain the memory occupancy at the chosen level. Over time, even if the nature of the input data has diverged significantly from the original training set this very simple mechanism will adapt automatically to changing circumstances and retain predictive performance on the most recent inputs. The key decision is how quickly to adapt and this will differ for each problem. The second possibility is that of adding new classes, which apart from the administration of the SBC memories would be almost automatic. In both of these cases there may be an argument for another phase of unsupervised learning (or perhaps occasional updating) but this would not necessarily be required. Contrast the simplicity of these solutions with the problems facing most other ML methods.

An application where these ideas might be fruitfully applied is in the ML sensor 2.0 paradigm (Warden et al., 2022) which we feel is a natural fit for *BitBrain* for the following reasons:

- A very simple interface is required to provide security and engineering modularity.
- The impact on model building, training, software development and integration. For example, the speed with which a specific sensor could be taught its own custom model on a training set with *BitBrain* and thereby sidestep issues about production variability and the sharing of large and complex pre-trained models.
- Continuous learning within the black box will generally be a problem but not for *BitBrain*.
- Inference that is robust to sensor degradation and environmental variability is considered essential.
- Specific and neuromorphic H/W is seen as the future of such low energy/always on devices.

The unsupervised learning mechanism described above has been shown to work well, however there may be other approaches using work from image processing in ML which can provide a useful alternative mechanism. Convolutional neural nets have been shown to provide very good results on challenging problems and in many cases the convolutional front end can be reused across problems of a similar nature. This could allow us to go directly from acquiring the data to the single-pass learning mechanism, and at the same time sidestep the issue of needing to relearn the front end in continuous learning problems.

We have only discussed single channel image data in this paper. We are keen to expand beyond this into any type of data and preliminary results are very promising. For example:

- Multi-channel (such as RGB) image or volumetric data (e.g., in medical imaging).
- DNA, IP or other engineering/biological/pharmaceutical codes with no obvious locality structure.
- Uni- or multi-variate time series including real-time data from event-based sensors so that temporal as well as spatial patterns can be classified.

There are a number of very interesting questions to answer as we expand the technique into these areas.

Just as in other areas of AI and ML, layers of inference and/or hierarchies can be very powerful extensions of a basic learning and inference mechanism. We believe that the same may be true of *BitBrain*. The question will be: how to connect the SBCs together? For example, in forming layers of SBCs, we would need some form of 'output' from the upper layer which is not the class itself. This would then feed the next layer as input. There are a number of interesting possibilities currently under consideration. This also provides the option that information flow can be feedback as well as feedforward, and therefore the opportunity for more end-to-end style learning mechanisms.

Finally, we are interested in experimenting with robustness in the presence of input perturbation of more realistic forms than simple Gaussian or Salt and Pepper noise added to the data.

There are so many possibilities here that separate papers will be required to address them all.

7.2. Neuromorphic interpretation

All computational operations for *BitBrain*—whether in training or inference—are small integer addition/multiplication or bitwise operations, which can be performed in one cycle on a RISC architecture with low energy use and often also leverage efficient SIMD units. This is rarely the case in typical ANN computations. The energy benefits of this difference will vary widely but an order of magnitude is not unrealistic.

While the *BitBrain* architecture can be mapped onto existing neuromorphic computational devices, there are likely to be greater gains in performance and energy-efficiency from mapping it into neuromorphic hardware specifically designed to support it. Figure 2 is highly suggestive of a possible hardware implementation of an SBC, comprising a 2D array of nodes where each node incorporates a number of SRAM cells to store the class bits. This 2D array has a row AD and a column AD, where each AD is a linear array of ADEs. The vector of input values is broadcast across all the ADEs, each of which selects its chosen inputs, computes

its activation and, if this exceeds its threshold, fires its output. Wherever two active ADE outputs meet across the 2D array the corresponding node is activated for a read or write operation. Writing can use a class indicator broadcast across the 2D array; reading is trickier as it involves counting the 1s for each class across all of the activated nodes, which could be achieved using analogue techniques or perhaps by serially pulsing the active row ADEs and counting the 1s by column.

7.2.1. *BitBrain* implementation on SpiNNaker

On some of the existing neuromorphic devices, such as Loihi and TrueNorth (Akopyan et al., 2015; Davies et al., 2018), the ADEs could potentially be mapped onto individual units resembling simple spiking neurons. The SpiNNaker system (Furber and Bogdan, 2020) offers a number of ways to directly take advantage of the sparsity and parallelism inherent in *BitBrain* thanks to its ability to execute non-neural simulations. SpiNNaker is a digital neuromorphic platform which can host up to 1 million general purpose processors in a single cluster, thus allowing for an extremely high number of calculations to be executed in parallel. The SpiNNaker platform, as well as the discussed implementation of the *BitBrain* algorithm, are designed to allow for robust and loss-tolerant routing of messages. Therefore, a loss of information from faulty nodes is redundant for the computation.

There are many different ways to implement such a system on SpiNNaker. In this section, we will discuss one such implementation which consists of two types of application vertices: ADE and SBC vertices. Here, an application vertex is understood as a particular type of application running on an independent core in the SpiNNaker system. These applications carry out certain computational tasks, and then communicate between each other *via* multicast packets which contain certain type of payload specific to the application. In the extremely parallel implementation which assumes a *BitBrain* instance with 4 ADs of length 2,048, the ADE vertices alone would require 8,192 cores to be employed. Additionally, 6 cores would be needed for the SBC vertices which receive firing patterns and interact with the SBC memory. We found that such massively parallel implementations are sub-optimal, due to the large number of messages which need to be processed. Moreover, we found that computation done by the SBC vertices requires more time to be completed, therefore further parallelisation of ADE processing does not bring expected gains in performance.

For the purpose of this paper we decided to focus on an implementation which parallelises not only the ADE calculations but also writing and reading from the SBC memories. Note that this in an early implementation and we plan to do a more thorough study to understand and explore the many options, as there is so much flexibility in how to implement the algorithm on SpiNNaker that finding the best balance between, e.g., data movement, AD calculations, SBC calculations, message types and quantity, memory access patterns is a large and sometimes counter-intuitive task. Moreover, the SpiNNaker architecture is based on rather old processors ARM9 with clock rate of 200 MHz.

We found that it is the most beneficial to divide each SBC into 64 cores which only receive messages from a small subset of ADE vertices, see the upper panel of Figure 12. Thus, we use 384 cores in total for reading and writing from the SBC memory. We then divide the ADE computation into further 128 cores. With this approach we've been able to recreate the performance we have previously recorded on conventional processors, and achieve an inference time of approximately 48s, or ≈4.8 ms per data point. Notably, the performance accuracy of our implementation is higher than that of other approaches previously implemented on SpiNNaker, such as Liquid State Machines (LSM) (Patiño-Saucedo et al., 2022). However, it is important to emphasise that these previous attempt typically used a variant of MNIST dataset—Neuromorphic MNIST (Orchard et al., 2015), which requires additional preprocessing steps and operates on temporal inputs, thus cannot be compared directly.

The ADE cores are responsible for calculating the firing patterns of a small subset of pixels in the image. When activated, an ADE vertex sends a message to the set of its corresponding SBC vertices, see the lower panel of Figure 12. In turn, the SBC vertices collect the incoming messages, and calculate the feature coincidences. The updates to the application vertices are performed on a timer interrupt which occurs every simulation time step. The length of this time step is determined by the complexity of the calculations which need to be performed and he messages that need to be processed. On average the SBC vertices receive ≈60 activation messages from the ADE vertices, with an additional 16 messages which are meant to indicate that the ADE vertices have finished their part of the job. On the other hand, the ADE vertices receive 192 messages per one image in the dataset.

The training and testing sets, as well as other data structures required to implement a *BitBrain* instance are stored in two types of memory: SDRAM (shared slow memory) which contains the SBC memories and the full set of training/testing data, and DTCM (fast local memory) which contains the simulation parameters, routing keys, address decoder thresholds, firing patterns, and test labels (in inference mode). Additionally, we allow the ADE vertices to transfer the training/testing data for each example into the DTCM memory, while the system is waiting for the SBC vertices to interact with their respective SBC memories. This approach allows us to reduce the inference time further by approximately ≈25%.

Each of the SBC vertices has a recording channel. In the inference mode the count of SBC activations per class is being recorded for each example in the test set. In the training mode the SBC memories are recorded only once after the whole training set has been processed. The recordings are then accessed by the host machine in order to save the trained SBC memories, or calculate the inference accuracy and build a confusion matrix.

8. Conclusions

We have introduced an innovative working mechanism (the *SBC memory*) and surrounding infrastructure (*BitBrain*) based upon a novel synthesis of ideas from sparse coding, computational neuroscience and information theory that support single-pass learning, accurate and robust inference, and the potential for continuous adaptive learning. We have

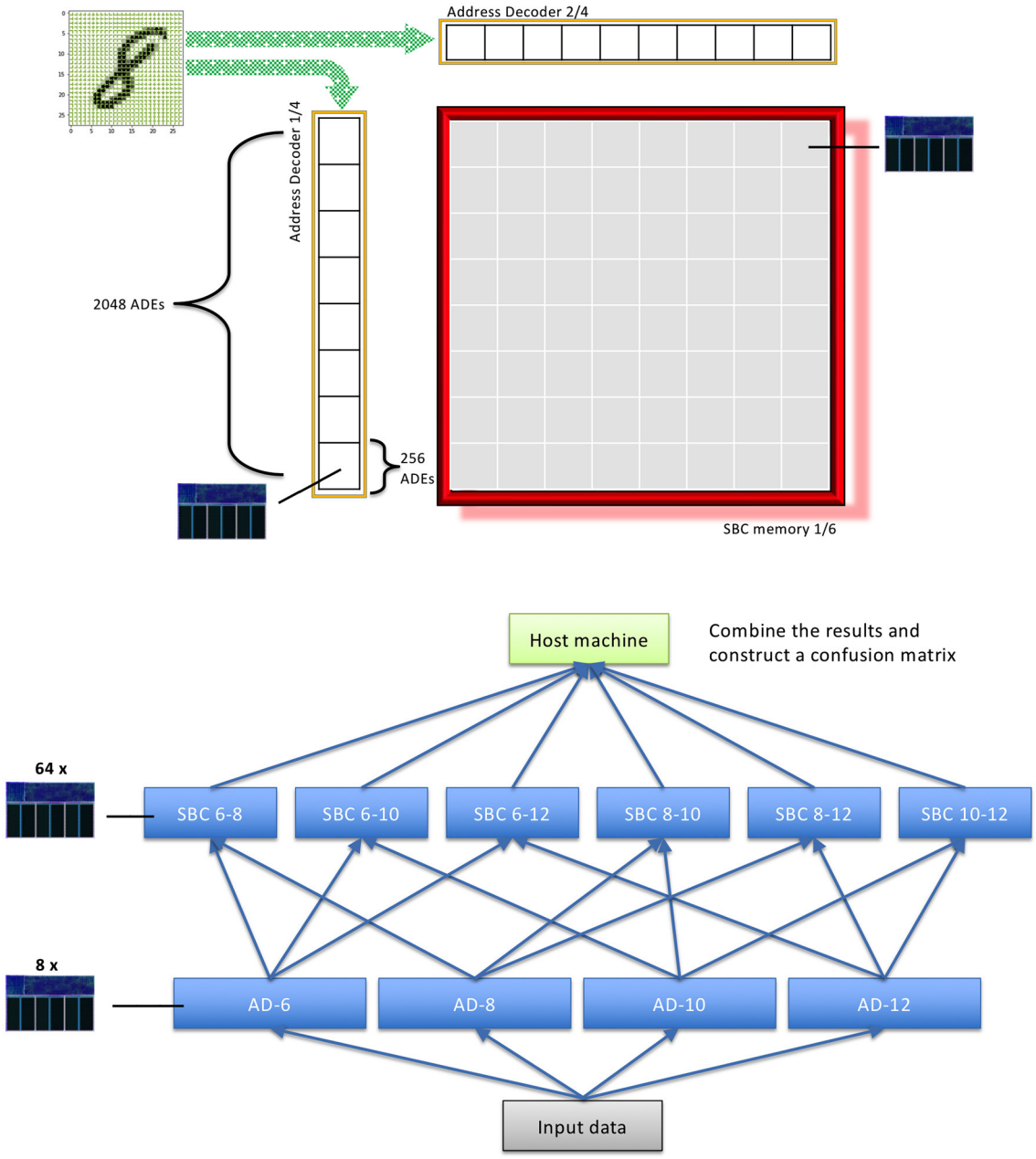


FIGURE 12
An example of distributed implementation of *BitBrain* on SpiNNaker neuromorphic platform. In this example, we divide the SBC computation into 64 parts per SBC memory. Additionally, each Address Decoder is split into 8 cores consisting of 256 ADEs. The panel below illustrates the message routing between different application vertex types in a parallelised implementation of *BitBrain*.

demonstrated the efficacy of these concepts on the MNIST and EMNIST benchmarks and shown that the proposed inference mechanism has very low training costs and is robust to noise.

Clearly these ideas are not yet fully developed, and theoretical advances as well as practical experience are likely to provide further gains in performance and efficiency. There are various ways that the mechanisms can be reconfigured, for example to reduce the SBC memory requirements by using an efficient compressed sparse matrix storage format, and/or by reducing the number of classes stored at each potential coincidence node.

Even at this early stage of development, *BitBrain* displays state-of-the-art performance in one-shot learning tasks combined with intrinsic robustness and fast inference. The Sparse Binary Coincidence memories upon which it is based may be large, but are simple bit arrays set to mark principal feature coincidences. This mechanism supports continuous on-line learning provided that the memories do not become over full, which may be ensured by incorporating some form of random "forgetting" .

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

MH and SF created and carried out testing of the original ideas. EJ and JF produced the comparative CNN results. JF implemented *BitBrain* on the SpiNNaker hardware. All authors contributed to the article and approved the submitted version.

Funding

This work was supported by the European Union Flagship Human Brain Project Specific Grant Agreement 3 (H2020 945539).

Acknowledgements

We thank Petruț Bogdan for help with *LaTeX* and *git*, along with various members of the SpiNNaker group (in particular Andrew Rowley) and Basabdatta Bhattacharya for comments on an earlier draft and a discussion of the comparative results.

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.

References

Adithya, A. (2022). *Are Biologically-Inspired Filters Robust Against Image Distortion and Noise?* Technical report, Brain-inspired Neural Networks Lab, Dept of Computer Science and Information Systems, BITS Pilani Goa Campus.

Ahmad, S., and Hawkins, J. (2016). *How Do Neurons Operate on Sparse Distributed Representations? A Mathematical Theory of Sparsity, Neurons and Active Dendrites*. Technical report, Numenta.

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. Computer Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Atick, J. J., and Redlich, A. N. (1993). Convergent algorithm for sensory receptive field development. *Neural Comput.* 5, 45–60. doi: 10.1162/neco.1993.5.1.45

Austin, J. (1998). *RAM-Based Neural Networks*. World Scientific.

Baldominos, A., Sáez, Y., and Isasi, P. (2019). A survey of handwritten character recognition with MNIST and EMNIST. *Appl. Sci.* 2019, 3169. doi: 10.3390/app9153169

Bell, A. J., and Sejnowski, T. J. (1995). An information-maximization approach to blind separation and blind deconvolution. *Neural Comput.* 7, 1129–1159. doi: 10.1162/neco.1995.7.6.1129

Bernardo, J. M., and Smith, A. (2000). *Bayesian Theory*. Chichester: John Wiley & Sons Ltd.

Branco, T., and Häusser, M. (2010). The single dendritic branch as a fundamental functional unit in the nervous system. *Curr. Opin. Neurobiol.* 20, 494–502. doi: 10.1016/j.conb.2010.07.009

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). "EMNIST: extending MNIST to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2921–2926.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *J. Mach. Learn. Res.* 7, 551–585. Available online at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-33646371466&partnerID=40&md5=a5ba8b961c124786647232b89322edfc

Dahmen, D., Layer, M., Deutz, L., Dąbrowska, P. A., Voges, N., von Papen, M., et al. (2022). Global organization of neuronal activity only requires unstructured local connectivity. *eLife* 11, e68422. doi: 10.7554/eLife.68422

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, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* 29, 141–142. doi: 10.1109/MSP.2012.2211477

Edwards, S., Grove, D., and Wynn, H., editors (2000). *Statistics for Engine Optimization*. Wiley.

Furber, S., Bainbridge, J., Cumpstey, M., and Temple, S. (2004). Sparse distributed memory using *N-of-M* codes. *Neural Netw.* 17, 1437–1451. doi: 10.1016/j.neunet.2004.07.003

Furber, S., and Bogdan, P. (eds.). (2020). *SpiNNaker: A Spiking Neural Network Architecture*. Boston, MA; Delft.

Furber, S., Brown, G., Bose, J., Cumpstey, J., Marshall, P., and Shapiro, J. (2007). Sparse distributed memory using rank-order neural codes. *IEEE Trans. Neural Netw.* 18, 648–659. doi: 10.1109/TNN.2006.890804

Govindarajan, A., Israely, I., Huang, S.-Y., and Tonegawa, S. (2011). The dendritic branch is the preferred integrative unit for protein synthesis-dependent LTP. *Neuron* 69, 132–146. doi: 10.1016/j.neuron.2010.12.008

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hopkins, M., Pineda-García, G., Bogdan, P. A., and Furber, S. B. (2018). Spiking neural networks for computer vision. *Interface Focus* 8, 20180007. doi: 10.1098/rsfs.2018.0007

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge: Cambridge University Press.

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge, MA: The MIT Press.

Kastellakis, G., Cai, D. J., Mednick, S. C., Silva, A. J., and Poirazi, P. (2015). Synaptic clustering within dendrites: an emerging theory of memory formation. *Prog. Neurobiol.* 126, 19–35. doi: 10.1016/j.pneurobio.2014.12.002

Larkum, M. E., and Nevian, T. (2008). Synaptic clustering by dendritic signalling mechanisms. *Curr. Opin. Neurobiol.* 18, 321–331. doi: 10.1016/j.conb.2008.08.013

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 46, 2278–2324. doi: 10.1109/5.726791

Linsker, R. (2005). Improved local learning rule for information maximization and related applications. *Neural Netw.* 18, 261–265. doi: 10.1016/j.neunet.2005.01.002

London, M., and Häusser, M. (2005). Dendritic computation. *Annu. Rev. Neurosci.* 28, 503–532. doi: 10.1146/annurev.neuro.28.061604.135703

Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. *Proc. IEEE* 102, 860–880. doi: 10.1109/JPROC.2014.2310593

Masland, R. H. (2012). The neuronal organization of the retina. *Neuron* 76, 266–280. doi: 10.1016/j.neuron.2012.10.002

Mazzia, V., Salvetti, F., and Chiaberge, M. (2021). Efficient-CapsNet: capsule network with self-attention routing. *Sci. Rep.* 11, 14634. doi: 10.1038/s41598-021-93977-0

Mel, B. (1992). NMDA-based pattern discrimination in a modeled cortical neuron. *Neural Comput.* 4, 502–517. doi: 10.1162/neco.1992.4.4.502

Neal, R. M. (1996). *Priors for Infinite Networks*. New York, NY: Springer.

Odone, F., Barla, A., and Verri, A. (2005). Building kernels from binary strings for image matching. *IEEE Trans. Image Process.* 14, 169–180. doi: 10.1109/TIP.2004.840701

O'Hagan, A., and Kendall, M. (1994). *Kendall's Advanced Theory of Statistics: Bayesian Inference*. Edward Arnold.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9, 437. doi: 10.3389/fnins.2015.00437

Papoutsi, A., Kastellakis, G., Psarrou, M., Anastasakis, S., and Poirazi, P. (2014). Coding and decoding with dendrites. *J. Physiol.* 108, 18–27. doi: 10.1016/j.jphysparis.2013.05.003

Patiño-Saucedo, A., Rostro-Gonzalez, H., Serrano-Gotarredona, T., and Linares-Barranco, B. (2022). Liquid state machine on spinnaker for spatio-temporal classification tasks. *Front. Neurosci.* 16, 819063. doi: 10.3389/fnins.2022.819063

Phadke, M. S. (1989). *Quality Engineering Using Robust Design*. Englewood Cliffs, NJ: Prentice Hall.

Raginsky, M., and Lazebnik, S. (2009). "Locality-sensitive binary codes from shift-invariant kernels," in *Advances in Neural Information Processing Systems*, Vol. 22, eds Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (Curran Associates, Inc.).

Rai, P. H. III, and Venkatasubramanian, S. (2009). "Streamed learning: one-pass SVMs," in *IJCAI International Joint Conference on Artificial Intelligence*.

Rasmussen, C. E., and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Reike, F., Warland, D., van Steveninck, R., and Bialek, W. (1996). *Spikes: Exploring the Neural Code*. Cambridge, MA: MIT Press.

Richards, B. A., and Lillicrap, T. P. (2019). Dendritic solutions to the credit assignment problem. *Curr. Opin. Neurobiol.* 54, 28–36. doi: 10.1016/j.conb.2018.08.003

Rifkin, R., and Klautau, A. (2004). In defense of one-vs-all classification. *J. Mach. Learn. Res.* 5, 101–141. doi: 10.5555/1005332.1005336

Shawe-Taylor, J., and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Shewry, M., and Wynn, H. (1987). Maximum entropy sampling. *J. Appl. Stat.* 14, 165–170.

Sivia, D. S., and Skilling, J. (2006). *Data Analysis - A Bayesian Tutorial, 2nd Edn*. Oxford University Press.

Stuart, G., Spruston, N., and Häusser, M. (2016). *Dendrites*. Oxford University Press.

Thorpe, S., and Gautrais, J. (1998). *Rank Order Coding*. Boston, MA: Springer US.

Tishby, N., Pereira, F. C., and Bialek, W. (1999). *The Information Bottleneck Method*.

Wang, J., Hoi, S., Zhao, P., Zhuang, J., and Liu, Z.-Y. (2013). "Large scale online kernel classification," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 1750–1756.

Wang, Z., Crammer, K., and Vucetic, S. (2012). Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training. *J. Mach. Learn. Res.* 13, 3103–3131. Available online at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84869463516&partnerID=40&md5=9360a34f58214aeb816a6dc03587f775

Warden, P., Stewart, M., Plancher, B., Banbury, C., Prakash, S., Chen, E., et al. (2022). Machine learning sensors. *arXiv preprint arXiv:2206.03266*. Available online at: https://arxiv.org/abs/2206.03266

Yang, S., Gao, T., Wang, J., Deng, B., Lansdell, B., and Linares-Barranco, B. (2021a). Efficient spike-driven learning with dendritic event-based processing. *Front. Neurosci.* 15, 601109. doi: 10.3389/fnins.2021.601109

Yang, S., Wang, J., Hao, X., Li, H., Wei, X., Deng, B., et al. (2021b). Bicoss: toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 2801–2815. doi: 10.1109/TNNLS.2020.3045492

Zhou, Z., Zheng, W.-S., Hu, J.-F., Xu, Y., and You, J. (2016). One-pass online learning: a local approach. *Pattern Recogn.* 51, 346–357. doi: 10.1016/j.patcog.2015.09.003

frontiers | Frontiers in Neuroinformatics

TYPE Original Research
PUBLISHED 04 May 2023
DOI 10.3389/fninf.2023.1144301

OPEN ACCESS

EDITED BY
Bert Offrein,
IBM Research - Zurich, Switzerland

REVIEWED BY
Michael Wynn Hopkins,
The University of Manchester, United Kingdom
Leslie Samuel Smith,
University of Stirling, United Kingdom

*CORRESPONDENCE
Md Abdullah-Al Kaiser
✉ mdabdull@usc.edu

†These authors have contributed equally to this work

RECEIVED 14 January 2023
ACCEPTED 13 April 2023
PUBLISHED 04 May 2023

CITATION
Kaiser MA-A, Datta G, Wang Z, Jacob AP, Beerel PA and Jaiswal AR (2023) Neuromorphic-P^{2}M: processing-in-pixel-in-memory paradigm for neuromorphic image sensors.
Front. Neuroinform. 17:1144301.
doi: 10.3389/fninf.2023.1144301

Neuromorphic-P^{2}M: processing-in-pixel-in-memory paradigm for neuromorphic image sensors

Md Abdullah-Al Kaiser[1,2*†], Gourav Datta[1†], Zixu Wang[1], Ajey P. Jacob[2], Peter A. Beerel[1,2] and Akhilesh R. Jaiswal[1,2]

[1]Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA, United States, [2]Information Sciences Institute, University of Southern California, Los Angeles, CA, United States

Edge devices equipped with computer vision must deal with vast amounts of sensory data with limited computing resources. Hence, researchers have been exploring different energy-efficient solutions such as near-sensor, in-sensor, and in-pixel processing, bringing the computation closer to the sensor. In particular, in-pixel processing embeds the computation capabilities inside the pixel array and achieves high energy efficiency by generating low-level features instead of the raw data stream from CMOS image sensors. Many different in-pixel processing techniques and approaches have been demonstrated on conventional frame-based CMOS imagers; however, the processing-in-pixel approach for neuromorphic vision sensors has not been explored so far. In this work, for the first time, we propose an asynchronous non-von-Neumann analog processing-in-pixel paradigm to perform convolution operations by integrating *in-situ* multi-bit multi-channel convolution inside the pixel array performing analog multiply and accumulate (MAC) operations that consume significantly less energy than their digital MAC alternative. To make this approach viable, we incorporate the circuit's non-ideality, leakage, and process variations into a novel hardware-algorithm co-design framework that leverages extensive HSpice simulations of our proposed circuit using the GF22nm FD-SOI technology node. We verified our framework on state-of-the-art neuromorphic vision sensor datasets and show that our solution consumes ∼ 2× lower backend-processor energy while maintaining almost similar front-end (sensor) energy on the IBM DVS128-Gesture dataset than the state-of-the-art while maintaining a high test accuracy of 88.36%.

KEYWORDS

neuromorphic, processing-in-pixel-in-memory, convolution, address event representation, hardware-algorithm co-design, DVS gesture

1. Introduction

Today's widespread video acquisition and interpretation applications [e.g., autonomous driving (Beltrán et al., 2020), surveillance (Xie et al., 2021), object detection (Jiao et al., 2022), object tracking (Wu et al., 2021), and anomaly detection (Mansour et al., 2021)] are fueled by CMOS image sensors (CIS) and deep learning algorithms. However, these computer vision systems suffer from energy inefficiency and throughput bottlenecks (Chai, 2020) that stem from the transmission of a high volume of data between the sensors at the edge and processors in the cloud. For example, smart glasses (e.g., Meta AR/VR glasses,

Google classes, etc.) drain the battery within 2–3 h when used for intensive computer vision tasks (LiKamWa et al., 2014). Although significant technological and system-level advancements exist in both CMOS imagers (Maheepala et al., 2020) and deep neural networks (Goel et al., 2020), the underlying energy inefficiency arises due to the physical separation of sensory and processing hardware. Hence, developing novel energy-efficient hardware for resource-constrained computer vision applications has attracted significant attention in the research community.

Many researchers implement the first few computation tasks of machine vision applications close to the sensor to reduce the energy consumption of massive data transfer (Zhou and Chai, 2020). These approaches can be categorized into three types (1) near-sensor processing, (2) in-sensor processing, and (3) in-pixel processing. The near-sensor processing approach places a digital signal processor or machine learning accelerator close to the sensor chip. In Pinkham et al. (2021), a dedicated near-sensor processor led to a 64.6% drop in inference energy for MobileNetV3. In Eki et al. (2021), a 3D stacked system consisting of a CNN inference processor and a back-side illuminated CMOS image sensor demonstrated an energy efficiency of 4.97 TOPS/W. Near-sensor computing can improve energy efficiency by reducing the data transfer cost between the sensor chip and the cloud/edge processor; however, the data traffic between the sensor and near-sensor processor still consumes significant amounts of energy.

In contrast, the in-sensor approach utilizes an analog or digital signal processor at the periphery of the sensor chip. For instance, RedEye (LiKamWa et al., 2016) uses analog convolution processing before the sensor's analog-to-digital conversion (ADC) blocks to obtain a 5.5× reduction in sensor energy. Moreover, a mixed-mode in-sensor tiny convolution neural network (CNN) (Hsu et al., 2022) yielded a significant reduction in bandwidth and, in particular, reduced power consumption associated with the ADC. To fully remove the ADC energy overhead, Chen et al. (2019) processed the raw analog data from the CMOS image sensor using an on-chip completely analog binary neural network (BNN) that leverages switched capacitors. Using energy-efficient analog computing was also explored in Ma et al. (2019), which proposes a novel current-mode analog low-precision BNN. Furthermore, SleepSpotter (Lefebvre et al., 2021) implemented energy-efficient current-domain on-chip MAC operations. Nevertheless, this solution still requires the potentially-compressed raw analog data to be streamed through column-parallel bitlines from the sensor nodes to the peripheral processing networks. In general, these in-sensor approaches significantly reduce the energy overhead of analog-to-digital converters; however, they still suffer from the data transfer bottleneck between the sensor and peripheral logic.

On the other hand, the in-pixel processing approach integrates computation capabilities inside the pixel array to enable early processing and minimize the subsequent data transmission. For instance, a low-voltage in-pixel convolution operation has been proposed in Hsu et al. (2020) that utilizes a current-based digital-to-analog converter (DAC) to implement weights and pulse-width-modulated (PWM) pixels. Moreover, a single instruction multiple data (SIMD) pixel processor array (PPA) (Bose et al., 2020) can perform parallel convolution operations within the pixel array by storing the weights of the convolution filters in registers within the in-pixel processing elements. In addition, the direct utilization of the photodetector current to compute the binary convolution can yield 11.49 TOPS/W energy efficiency (Xu et al., 2020). Furthermore, Xu et al. (2021) performs classification tasks on the MNIST dataset by generating the in-pixel MAC results of the first BNN layer and exhibits 17.3 TOPS/W energy efficiency. In addition, a processing-in-pixel-in-memory paradigm for CIS reported an 11× energy-delay product (EDP) improvement on the Visual Wake Words (VWW) dataset (Datta et al., 2022c). Follow-up works by the same authors have demonstrated 5.26× and 3.14× reduction in energy consumption on hyperspectral image recognition (Datta et al., 2022e) and multi-object tracking in the wild (Datta et al., 2022d), respectively. In summary, due to the embedded pixel-level processing elements, the in-pixel processing approach can outperform energy and throughput compared to in-sensor and near-sensor processing solutions.

Most of the research works on different energy-efficient CIS approaches (near-sensor, in-sensor, and in-pixel processing) are focused on conventional frame-based imagers. However, many researchers are now exploring the use of event-driven neuromorphic cameras or dynamic vision sensors (DVS) (Lichtsteiner et al., 2008; Leñero-Bardallo et al., 2011) for different neural network applications, including autonomous driving (Chen et al., 2020), steering angle prediction (Maqueda et al., 2018), optical flow estimation (Zhu et al., 2018), pose re-localization (Nguyen et al., 2019), and lane marker extraction (Cheng et al., 2020), due to their energy, latency, and throughput advantages over traditional CMOS imagers. The DVS pixel generates event spikes based on the change in light intensity instead of sensing the absolute pixel-level illumination in conventional CMOS imagers. Thus, DVS pixels filter out the redundant information from a visual scene and produce sparse asynchronous events. These sparse events are communicated off-chip using the address event link protocol (Lin and Boahen, 2009). By avoiding the analog-to-digital conversion of the absolute pixel intensity and frame-based sensing method, DVS exhibits higher energy efficiency, lower latency, and higher throughput than frame-based alternatives. Moreover, the dynamic range of the DVS pixel is higher than the conventional CMOS imagers; hence, the DVS camera can adapt to the illumination level of the scene due to its logarithmic receptor. These advantages motivate a paradigm shift toward neuromorphic vision sensors for vision-based applications.

These DVS cameras are often coupled with spiking convolution neural networks (CNN) that natively accept asynchronous input events. Traditionally, time is decomposed into windows, and the number of spikes that occur in each time window is accumulated independently for each pixel creating multi-bit inputs to a spiking CNN. The first spiking CNN layer thus consists of digital MAC operations (not accumulations because the input is multi-bit instead of binary), unlike the subsequent spiking CNN layers that consist of more energy-efficient accumulations that operate on spikes (Datta and Beerel, 2022; Datta et al., 2022b). To improve the energy efficiency of such a DVS system, this paper explores in-pixel processing by performing MAC operations in the analog domain within the pixel array. In particular, we have developed a novel energy-efficient neuromorphic processing-in-pixel-in-memory (P^2M) computing paradigm in which we implement the first spiking CNN layer using embedded transistors that model the multi-bit multi-channel weights and enable massively parallel

in-pixel spatio-temporal MAC operations. Because the DVS event spikes are asynchronous in nature, we perform the multiply operation by accumulating the associated weight each time a pixel event occurs. We threshold the accumulated value at the end of each time window to produce a binary output activation and reset the accumulator in preparation for the next time window. To support multiple input filters operating on individual pixels, we parallelize this operation and simultaneously operate on all channels (and all pixels). This charge-based in-pixel analog MAC operation exhibits higher energy efficiency than its digital off-chip counterpart. Moreover, the sparse binary output activations are communicated utilizing a modified address-event representation (AER) protocol, preserving the energy benefit of the workload sparsity. In addition, we have developed a hardware-algorithm co-design framework incorporating the circuit's non-linearity, process variation, leakage, and area consideration based on the GF22nm FD-SOI technology node. Finally, we have demonstrated the feasibility of our hardware-algorithm framework utilizing state-of-the-art neuromorphic event-driven datasets (e.g., IBM DVS128-Gesture, NMNIST) and evaluated our approach's performance and energy improvement. We incur a ∼5% accuracy drop in these datasets because our charge-based P^{2}M approach does not capture the conventional notion of membrane potential for the first CNN layer. This lack of membrane potential is due to the limited time a passive analog capacitor can effectively store charge without significant leakage. However, this problem can be mitigated using non-volatile memories (Jaiswal and Jacob, 2021) that we plan to explore in our future work.

The key contributions of our work are as follows:

1. We propose a novel neuromorphic-processing-in-pixel-in-memory (Neuromorphic-P^{2}M) paradigm for neuromorphic image sensors, wherein multi-bit pixel-embedded weights enable massively parallel spatio-temporal convolution on input events inside the pixel array.
2. Moreover, we propose non-von-Neumann charge-based energy-efficient in-pixel asynchronous analog multiplication and accumulation (MAC) units and incorporate the non-idealities and process variations of the analog convolution blocks into our algorithmic framework.
3. Finally, we develop a hardware-algorithm co-design framework considering hardware constraints (non-linearity, process variations, leakage, area consideration), benchmark the accuracy, and yield a $\sim 2\times$ improvement in backend-processor energy consumption on the IBM DVS128-Gesture dataset with a $\sim$ 5% drop in test accuracy.

The remainder of the paper is organized as follows. Section 2 describes the circuit implementation, operation, and manufacturability of our proposed Neuromorphic-P^{2}M approach. Section 3 explains our hardware-algorithm co-design approach and hardware constraints on the first layer of the neural network model. Section 4 demonstrates our experimental results on two event-driven DVS datasets and evaluates the accuracy and performance metrics. Finally, Section 5 presents the concluding remarks.

2. P^{2}M circuit implementation

This section presents the critical hardware innovations and implementation of our proposed neuromorphic-P^{2}M approach. Figure 1 illustrates the representative chip stack and computing flow for the first convolution layer utilizing our proposed neuromorphic-P^{2}M architecture. The top die consists of DVS pixels and generates ON (OFF) events based on the increase (decrease) in input light contrast level. A DVS pixel consists of a logarithmic receptor, source-follower buffer, capacitive-feedback difference amplifier, and two comparators (Lichtsteiner et al., 2008; Leñero-Bardallo et al., 2011; Son et al., 2017). The generated events (ON and OFF) per pixel are communicated to the bottom die via pixel-level hybrid Cu-to-Cu interconnects. The bottom die contains the weights and energy-efficient charge-based analog convolution blocks. Each DVS pixel's output channel (ON-channel and OFF-channel) is connected to a transistor in the bottom die that implements a multi-bit weight (e.g., $w_{1,ON}$, $w_{1,OFF}$, etc.) to perform the multiplication (e.g., $I_{1,ON} \times w_{1,ON}$, $I_{1,OFF} \times w_{1,OFF}$, etc.) operation. The positive and negative weights are implemented by utilizing the pMOS and nMOS transistors, respectively. Each kernel (corresponding to the filter of the spiking CNN model) accumulates its weighted multiplication of input events on an analog memory (capacitor) asynchronously when an ON or OFF event occurs in the input DVS pixel. As the input spikes are binary, the accumulation voltage either steps up (positive weight) or down (negative weight) by an amount, depending on the weight values. The accumulation continues for a fixed time period (simulation time length for each event stream of our neural network model), and after that, the summed voltage is compared with the threshold (using a comparator or skewed inverter) to generate the output activation signal (e.g., O_{ACT}) of each kernel for the next layer. A similar computing flow is used across the different kernels throughout the sensor array.

The operations of our proposed neuromorphic-P^{2}M can be divided into three phases. These are:-

1. Reset Phase: During the reset phase, the accumulation capacitor of each kernel is precharged to $0.5V_{DD}$ so that the accumulation voltage can step up or down within the supply rail depending on positive or negative weights, respectively.
2. Convolution Phase: In the convolution phase, the multi-bit weight-embedded pixels and the accumulation capacitor of each kernel perform multiplication and accumulation (MAC) operations in the analog domain for a fixed period of time. After that, the final accumulated voltage of each kernel is compared with a threshold voltage to (potentially) generate the output activation spike for the next layer.
3. Read Phase: Finally, during the read phase, the output activations of different kernels are sequentially read utilizing the asynchronous Address-Event Representation (AER) read scheme.

More details on each step, including their hardware implementations, will be explained below.

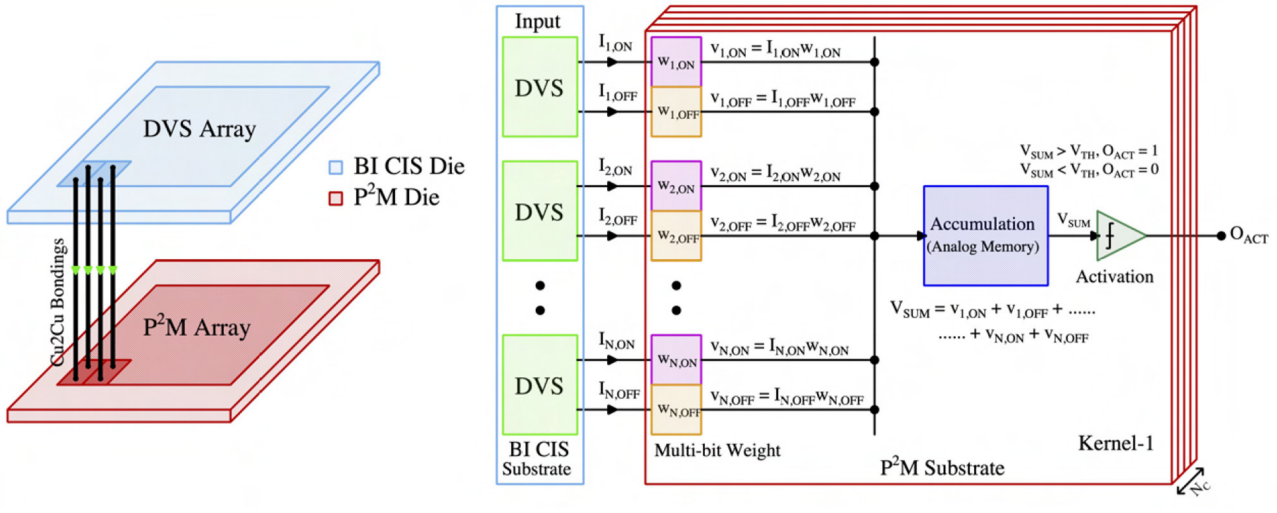


FIGURE 1
The representative 3D chip stack and computing flow diagram of our proposed neuromorphic-P^{2}M architecture.

2.1. Multi-bit weight embedded pixels

As illustrated in Figure 2, positive and negative weights of the first spiking CNN layer have been implemented by utilizing pMOS and nMOS transistors connected with supply voltages V_{DD} and ground, respectively. For a positive (negative) weight, the voltage across the kernel's capacitor (C_K) charges (discharges) from $0.5V_{DD}$ to V_{DD} (ground) as a function of weight values and the number of input DVS events. The weight values can be tuned by varying the driving strength ($\frac{W}{L}$ ratio) of the weight transistors (M_W). A high-V_T pMOS in positive weight implementation (nMOS in negative weight implementation) (M_{EN}) is activated during the convolution phase to enable the multiplication and accumulation operations on the kernel's capacitor (C_K) and remains off during the reset phase. The weight transistor (M_W) is chosen to have a high-V_T to limit the charging (for positive weight) or discharging (for negative weight) current to avoid capacitor saturation. Moreover, each DVS pixel includes a delayed self-reset circuit (consisting of a current-starved inverter chain and AND gate) to prevent voltage saturation on the capacitor (C_K) by limiting the event pulse duration. A switch transistor (M_{SW}) controlled by the DVS event spike is used to isolate the kernel's capacitor (C_K) from the weight transistor (M_W) to reduce the leakage. The switching transistor (M_{SW}) will be activated only when there are input DVS spikes, hence, ensuring the asynchronous MAC operation on the kernel's capacitor (C_K). Furthermore, to reduce the leakage, a kernel-dependent (as leakage is a function of transistor's geometry, hence, leakage amount is dependent on the kernel's weights) the current source (I_{NULL}) is connected with the accumulation capacitor (C_K) that flows in the opposite direction of the leakage current to nullify the leaky behavior of the capacitor. The number of weight transistors associated with a kernel depends on the size of the kernel (e.g., for a kernel size of 3 × 3, there will be a total of 18 weight transistors considering the ON and OFF-channel). Each kernel's weight transistors are connected to one accumulation capacitor (C_K).

Note that the weights cannot be re-programmed after manufacturing. However, it is common to use pre-trained weights for the first few layers as low-level feature extractors in modern neural network models (Jogin et al., 2018). Hence, the fixed weights of our proposed architecture do not limit its application for a wide range of machine-vision tasks. Moreover, we can also replace the transistor by utilizing a non-volatile memory device [e.g., Resistive Random Access Memory (RRAM), Phase Change Memory (PCM), Magnetic Random Access Memory (MRAM)] to add reconfigurability in our neuromorphic-P^{2}M approach.

To incorporate the circuit's non-ideality in our algorithmic model, we have simulated the output characteristics of the positive and negative weights for the different numbers of input event spikes using the GF 22nm FD-SOI node. Figure 3 represents the output voltage change on the accumulation capacitor (ΔV_{OUT}) as a function of the normalized weight transistor's $\frac{W}{L}$ ratios and different numbers of input event spikes. The figures show that the accumulated voltage can step up (for positive weights) and down (for negative weights), and the size of the step is dependent on the weight transistor's $\frac{W}{L}$ ratio. However, the step size dependency is non-linear, and the non-linearity is larger when the weights are large, and the pre-step voltage is close to the supply rails. This can be attributed to the fact that the weight transistors (M_W) enter the triode region when their drain-to-source voltage is low, causing the charging (discharging) current to drop compared to the typical saturation current. However, the number of input events is sparse for the DVS dataset, and having large weight values for all the weights in a kernel is highly unlikely for a neural network model. Hence, the weight transistors' non-linear characteristics do not cause significant accuracy issues in our algorithmic model. Besides, the circuit's asymmetry due to utilizing different types of transistors (pMOS for positive weights and nMOS for negative weights) is also captured and included in our algorithmic model.

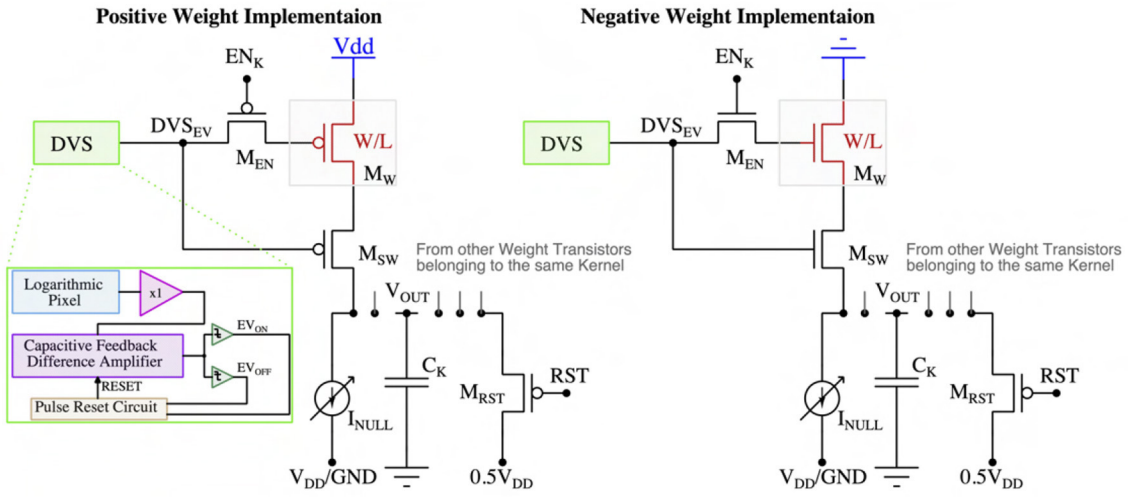


FIGURE 2
Embedded multi-bit positive and negative weight implementation.

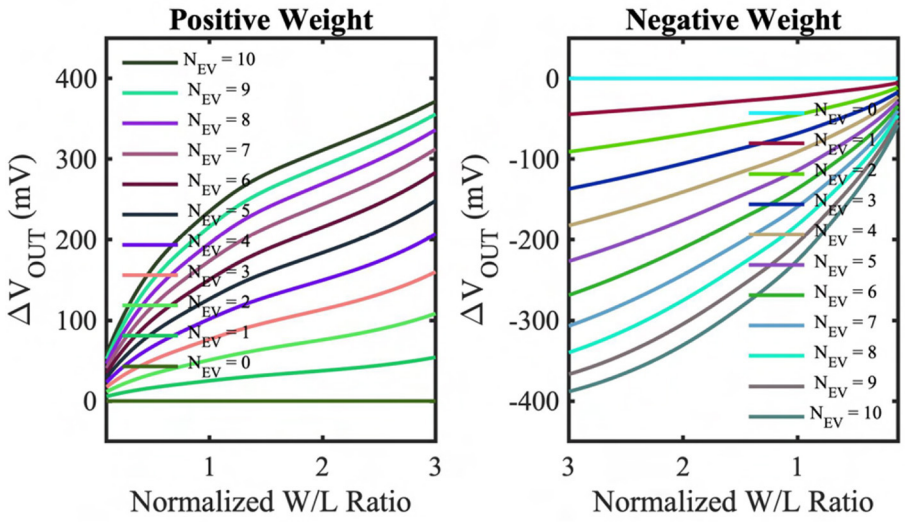


FIGURE 3
Output accumulation voltage change (ΔV_{OUT}) from the reset voltage of the kernel capacitor (C_K) as a function of normalized weight (normalized transistor $\frac{W}{L}$ ratio) and input event spikes simulated on GF 22nm FD-SOI node for positive and negative weights.

2.2. *In-situ* multi-pixel multi-channel convolution operation

In the first spiking CNN layer, we must perform *spatio-temporal* MAC operations across multiple channels *simultaneously* for each kernel. Figure 4 illustrates our proposed neuromorphic-P^{2}M architecture. The left sub-figure represents an array of DVS pixels (each white rectangular box includes multiple DVS pixels arranged in rows and columns) consisting of multiple channels distributed spatially. Each DVS pixel is connected with multiple weight transistors of the analog MAC blocks depending on the number of channels and stride (e.g., each DVS pixel will be connected with four sets of analog MAC blocks for a stride of 2). Each channel performs analog MAC operations asynchronously for a fixed temporal window (the length of each algorithmic time step). For instance, assume the kernel size is 3 × 3, and each kernel has 5 different channels that are represented by the white rectangular boxes in the left sub-figure. The right sub-figure exhibits the zoomed version of the 3 × 3 kernel with 5 different channels. Each channel has a dedicated accumulation capacitor (e.g., C_{Ki}, where i = 1, 2, ... 5) and a local bitline so that charge can accumulate across all the different channels at the same time. Depending on the kernel size, multiple weight transistors (both positive and negative) are connected to its kernel-dedicated accumulation capacitor using the local bitline of each channel. In this example, 18 weight transistors (kernel size = 3 × 3 and for the ON and OFF channels of the

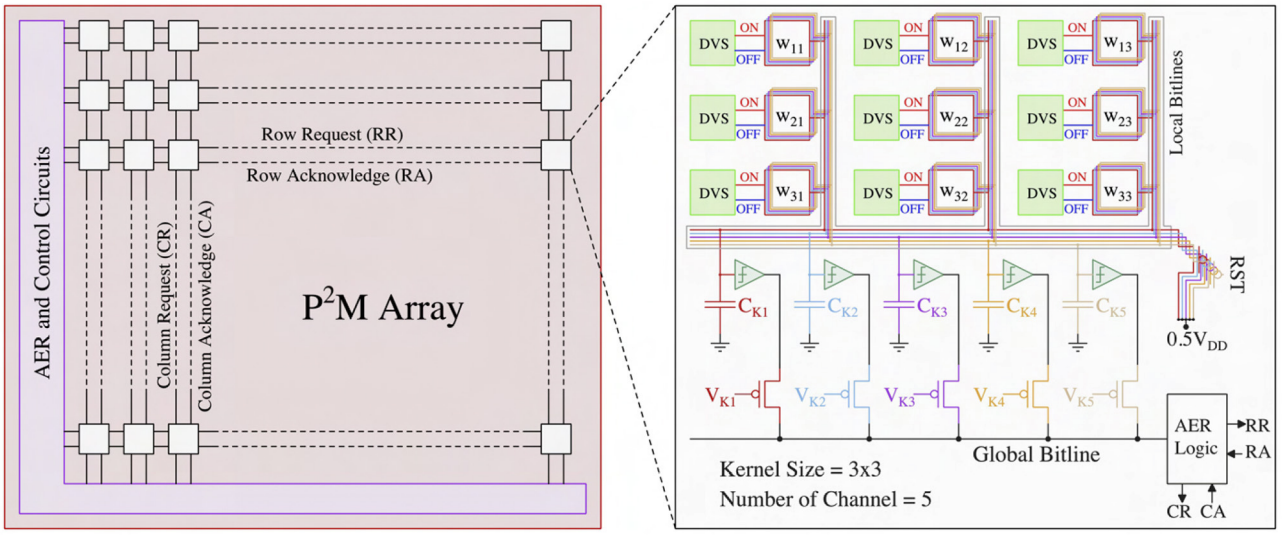


FIGURE 4
Neuromorphic-P^{2}M array block diagram with peripheral control circuits and multi-channel configuration of our proposed neuromorphic-P^{2}M architecture.

DVS pixels) are connected with a single kernel capacitor. The per-channel accumulation capacitor and local bitline shared among the kernel's weight transistors enable simultaneously and massively parallel spatio-temporal MAC operations across different channels. The multiplication results (fixed amount of charge transfer to kernel capacitor from V_{DD} or from kernel capacitor to GND as a function of positive or negative weight depending on the weight values) accumulate on the kernel capacitor for a fixed temporal window (length of each algorithmic time step). These analog MAC operations are asynchronous and parallel across all the kernels for all the input feature maps (DVS pixels) throughout the sensor array. Finally, a thresholding circuit compares the final accumulated voltage on each channel's capacitor with a reference voltage to generate the output activation spike. Output activations from different channels are multiplexed (controlled by V_{K1}, V_{K2}, etc.) to communicate with the AER read circuits at the periphery (left sub-figure) through the kernel-level AER logic block (right sub-figure). The row request (RA) and row acknowledge (RA) signals are shared along the rows, and the column request (CR) and column acknowledge (CA) signals are shared along the columns. After the read operation (described in Section 2.3), the kernel's accumulation capacitor is reset to $0.5V_{DD}$ by the reset transistor (M_{RST}) shown in Figure 5. Note that the reset operation implies no propagation of the voltage accumulated on the kernel's capacitor from one time step to subsequent time steps. Thus, the kernel capacitor voltage is unlike the typical representation of the membrane potential found in the literature (Datta et al., 2021, 2022a,b), which is conserved across time steps. Taking into cognizance the above behavior, for the first layer of the network, we ensure our algorithmic framework includes thresholding and reset operation across time steps, thus faithfully representing the circuit behavior in algorithmic simulations.

The frequency of the reset operation is based on the amount of time the capacitor can hold the charge without significant leakage. To minimize the capacitor leakage, we use high-V_T weight transistors, a switching transistor (M_{SW}) to disconnect the kernel's capacitor from the weight transistors, and kernel-dependent nullifying current source (I_{NULL}) (shown in Figure 2). According to our HSpice simulations, in the worst-case scenario (all weights are maximum in the kernel, which is very unlikely in the neural network model), the voltage on the accumulation capacitor deviates due to leakage from its ideal value by a mere 22 mV over a significantly longer duration of time (e.g., 1 ms). Based on the reset frequency, the length of each algorithmic time step of our neural network model has been set to 1 ms for the first layer.

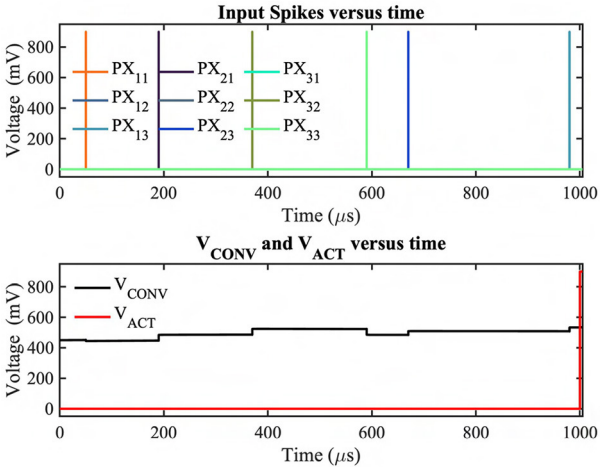


FIGURE 5
A random convolution operation with output activation spike simulated on GF 22nm FD-SOI node.

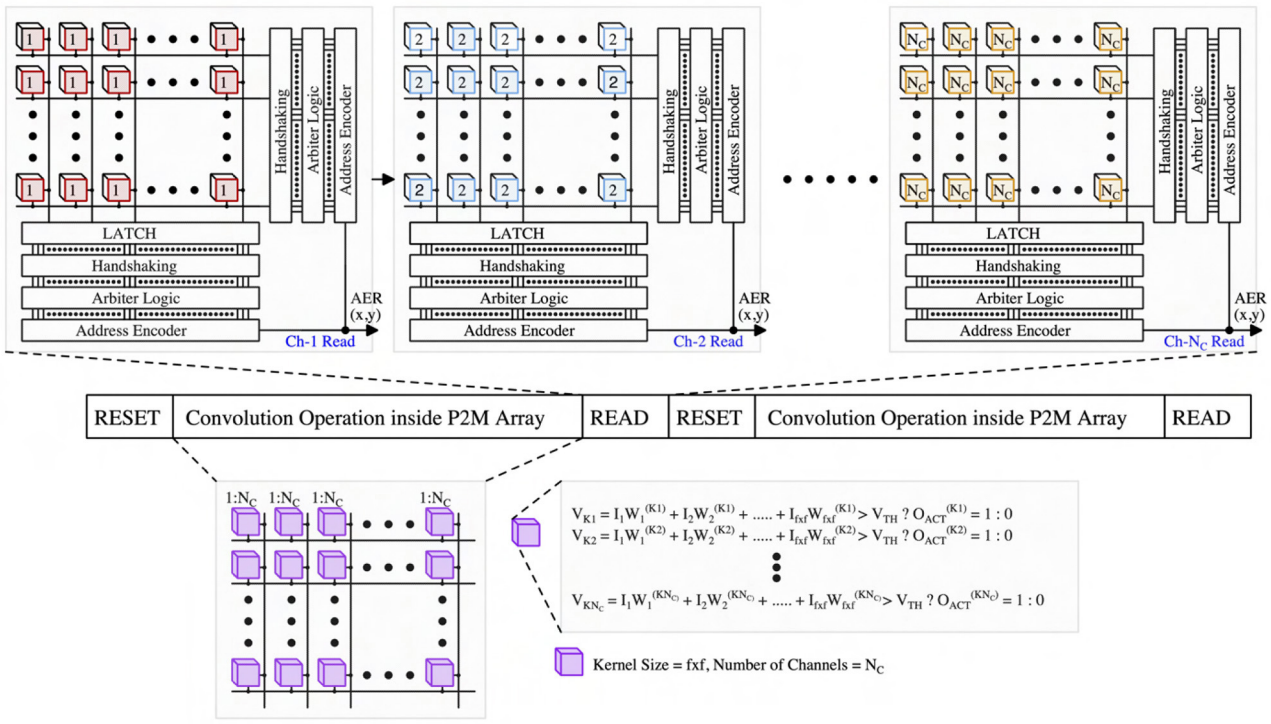


FIGURE 6
AER read scheme of our proposed neuromorphic-P^{2}M architecture.

Figure 5 illustrates an asynchronous convolution and output activation spike generation example of our proposed neuromorphic-P^{2}M using the GF22nm FD-SOI technology node considering random inputs and weights. For this simulation, a kernel size of 3 × 3 has been considered. The weights, the instant of the events, and the number of events per DVS pixel are generated randomly. For the test HSpice simulation, 1 ms simulation time has been considered according to our algorithmic framework; hence, all the output events from DVS pixels within this time period will be multiplied with their weights and accumulated on the Kernel's accumulation capacitor before being compared with a fixed threshold voltage. The top subplot exhibits that the DVS pixels (e.g., PX_{11}, PX_{21}, etc.) are generating the event spikes at different time instants. PX_{13}, PX_{21}, PX_{23}, PX_{32} are connected with positive weights, whereas the other pixels are connected with negative weights. It may also be noted that a few pixels (e.g., PX_{12}, PX_{22}, PX_{31}) do not generate any event during this time frame. This test simulation also considers these no-event generation scenarios to mimic the actual dataset sparsity. From the bottom subplot, it can be observed that the convolution output (V_{CONV}) of our analog MAC circuit is updating (charging or discharging) for each input event spike. When the weight is positive (negative), the accumulation voltage steps up (down) depending on the weight value. Finally, after the fixed temporal window, the convolution output has been compared with the threshold voltage. If the convolution output is higher than the threshold voltage, the comparator will generate an output activation spike (V_{ACT}) for the next layer for each kernel.

2.3. P^{2}M address-event representation (AER) read operation

In this sub-section, we propose modifications to the standard AER scheme in a manner so that it can be compatible with the presented asynchronous processing in-pixel computations. We are utilizing the asynchronous AER read-out scheme (Boahen, 2004) to read the output activations from the first convolution layer (mapped onto the DVS pixels using our proposed neuromorphic-P^{2}M paradigm). The representative read scheme is illustrated in Figure 6. Our P^{2}M architecture can support multiple numbers of channels (e.g., N_C) as required by the spiking CNN model. The outputs of the channels (thresholded output activation spikes) are read sequentially throughout the P^{2}M array in an asynchronous manner. At a time, one channel is being asserted of the P^{2}M array by activating V_{Ki} sequentially, where i = 1, 2, ... N_C (shown in Figure 4). Kernel-level AER logic block shared among different channels for each spatial feature map generates the row, and column request signals whenever an output activation spike exists in the kernel. For AER reading, row-parallel techniques can be used where it latches all the events generated in a single row and read them sequentially (Boahen, 2004). The peripheral address encoders (row and column encoders) of the AER read circuits output the x and y addresses of the output activation in parallel. Moreover, while performing the read operation, we can also pipeline the next reset and convolution phases without waiting for the read phase to be completed by adding a transistor between the kernel capacitor and the comparator. The comparator output can be stored on the

dynamic node for a short period of time, or even we can use a small holding capacitor to hold the output activation for a sufficient amount of time considering the read operation. As the output activations are sparse and AER read can be completed within a few μs windows, we can also utilize our architecture to perform the convolution and read phase in parallel. Besides, performing the in-pixel convolution operation reduces the output activation map size as a function of the kernel size and number of strides. In addition, we also do not need to send an extra bit to define the polarity of the event (ON or OFF-event), similar to the base DVS systems. As a result, the required number of address bits that need to be communicated off-chip has been reduced from the base DVS system. Hence, our P^{2}M architecture maintains the energy benefit of a sparse system due to utilizing the AER read scheme along with lower off-chip communication energy cost due to generating fewer address bits per output activation.

2.4. Process integration and area consideration

Figure 7 exhibits the representative illustration of a heterogeneously integrated system featuring our proposed neuromorphic-P^{2}M paradigm. Our proposed system can be divided into two key dies, i) a backside illuminated CMOS image sensor (BI-CIS) consisting of the DVS pixels and biasing circuitry, and ii) a die containing multi-bit multi-channel weight transistors, accumulation capacitors, comparators, and AER read circuits. Figure 4 shows that for each spatial feature (DVS pixels), the algorithm requires multiple channels that incur higher area due to multiple weight transistors and one accumulation capacitor per channel. However, due to the advantages of heterogeneous integration, our bottom die can be fabricated on an advanced technology node compared to the top die (BI-CIS). Hence, multiple channels in the bottom die can be accommodated and aligned with the top die without any area overhead while maintaining the neural network model accuracy. It may be noted that typical DVS pixels are larger due to the inclusion of a capacitive feedback difference amplifier. The overall system can be fabricated by a wafer-to-wafer bonding process using pixel-level hybrid Cu2Cu interconnects (Kagawa et al., 2016; Miura et al., 2019; Seo et al., 2021). Each DVS pixel has two Cu2Cu interconnects for its ON and OFF-channel, respectively. Considering the DVS pixel area of 40 $\mu m \times 40\ \mu m$ (Lichtsteiner et al., 2008) for 128 $\times$ 128 sensor array, Cu2Cu hybrid bonding pitch of 1 μm (Kagawa et al., 2020) and the analog convolution elements (weight transistors, comparators, accumulation capacitors) area in GF22nm FD-SOI node, our neuromorphic-P^{2}M architecture can support a maximum of 128 and 32 channels with a kernel size of 3 $\times$ 3 for stride 2 and 1, respectively. However, 32 channels with stride 2 have been utilized in our algorithmic framework. Such kernel-parallel MAC structure allows us to enable *in-situ* convolution operation without needing weight transfer from a different physical location; thus, this method does not lead to any data bandwidth or energy bottleneck.

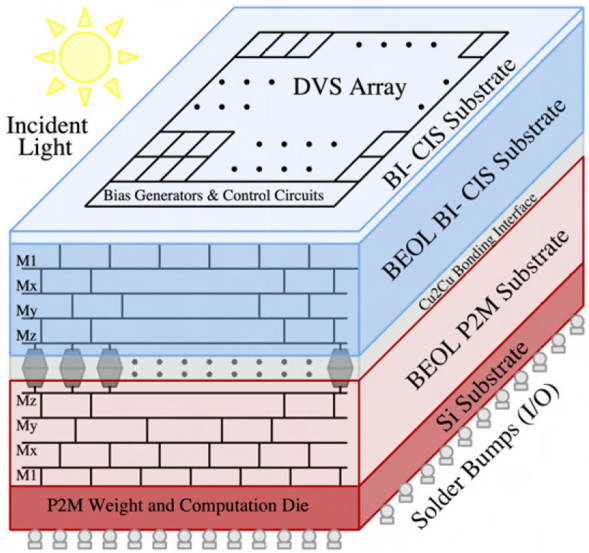


FIGURE 7
Representative illustration of a heterogeneously integrated system featuring neuromorphic-P^{2}M paradigm utilizing Cu2Cu bonding.

3. P^{2}M-constrained algorithm-hardware co-design

This section presents our algorithmic framework implementation guided by our proposed neuromorphic-P^{2}M architecture. The in-pixel charge-based analog convolution generates non-ideal non-linear convolution; in addition, process variation yields a deviation of the convolution result from the ideal output. Moreover, leakage poses constraints on the maximum length of each algorithmic time step, and the area limits the number of channels utilized per each spatial feature map. The hardware-algorithm co-design framework of our proposed neuromorphic-P^{2}M approach has been illustrated in Figure 8. More details on including non-idealities, process variation, leakage, and area effects in our algorithmic framework are given in the following subsections.

3.1. Custom convolution for the first layer modeling circuit non-linearity and process variation

From an algorithmic perspective, the first layer of a spiking CNN is a linear convolution layer followed by a non-linear activation unit. In our neuromorphic-P^{2}M paradigm, we have implemented the weights utilizing voltage accumulation through appropriately sized transistors that are inherently non-linear. As a result, any analog convolution circuit built on transistor devices will exhibit non-ideal non-linear behavior. Hence, to suppress the non-linearity, we have tuned our weights (transistor's geometry) in

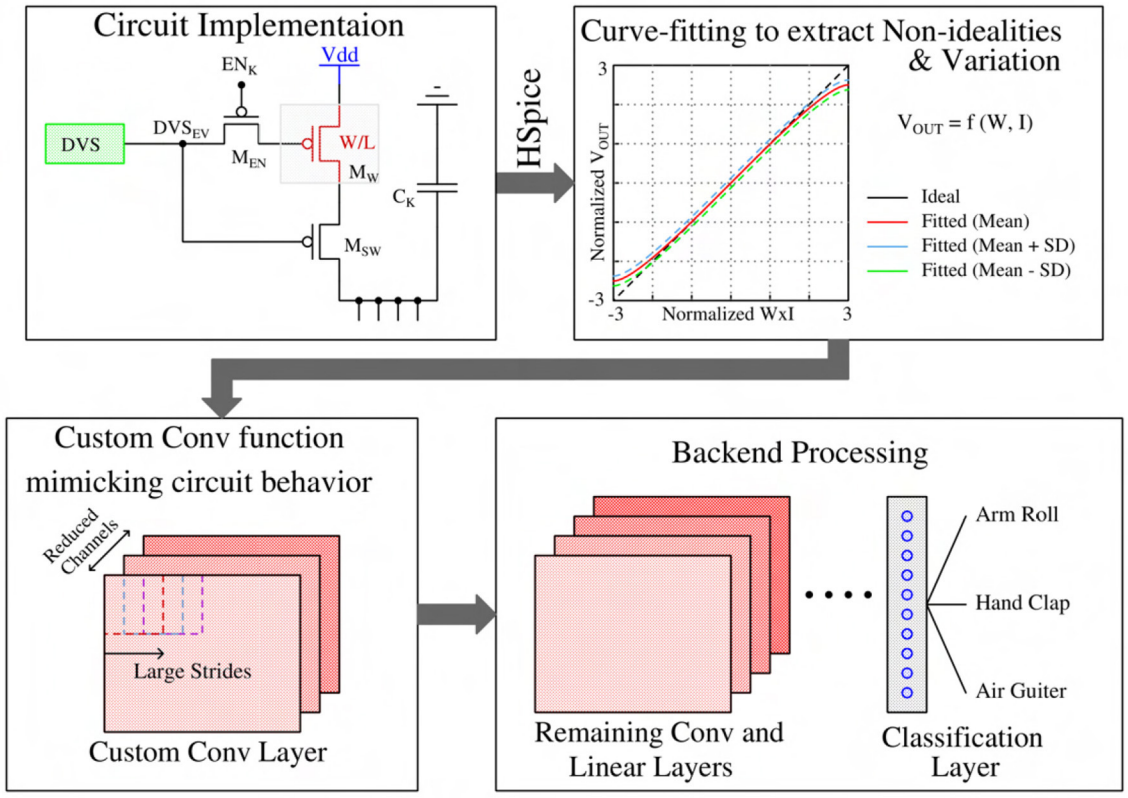


FIGURE 8
Hardware-algorithm co-design framework to enable our proposed neuromorphic-P^{2}M approach.

a non-linear manner in such a way that the output accumulation voltage steps can increase or decrease linearly for positive and negative weights, respectively. However, the nonlinearity is also a function of the drain-to-source voltage of the weight transistors. In our scheme, we are charging or discharging the kernel's capacitor during the computation phase. Depending on the weight values, the charging and discharging current are functionally dependent on the drain-to-source voltage. Hence, when the accumulation voltage on the V_{OUT} node (shown in Figure 2) gets larger (smaller) for the positive (negative) weights, the transistor enters into the triode region; hence, the charging or discharging current reduces. Besides, the same positive and negative weight values cannot ensure the same change in voltage accumulation due to device asymmetry (pMOS for positive weight implementation and nMOS for negative weight implementation). Furthermore, due to process variation, the transistor's geometry cannot be fabricated precisely; hence, the convolution output current can also vary due to process variation. Considering all these non-linear non-ideal behaviors and process variations, we extensively simulated our proposed P^{2}M paradigm for a wide range of input spikes and weights combinations considering leakage and around 3-sigma variation using GF22nm FD-SOI technology node. Figure 9 illustrates the resulting HSpice results with a standard deviation bar, i.e., the normalized convolution output voltages per pixel corresponding to a range of weights and input number of spikes have been modeled using a behavioral curve-fitting function. Note, for the scatter plot, we have used 100 μs temporal window for the convolution phase to save the total circuit simulation time as we have to run 1,000 Monte-Carlo simulations for each combination of weights and the number of input spikes. In our algorithmic framework, a random Gaussian sample value has been generated between the mean $\pm$ standard deviation for each particular normalized weight times input event value to capture the effects of the process variation. For the fixed simulation time for the event stream, in each Kernel, the accumulation output voltage per pixel is calculated first, then added to the other pixel's accumulation voltage inside the kernel to calculate the final output. The algorithmic framework was then used to optimize the spiking CNN training for the event-driven neuromorphic datasets. Besides the above-mentioned non-ideality and variation effects, thermal noise and temperature variation may affect the inference performance. The thermal noise of the circuit can also be modeled as zero-mean Gaussian distribution (Gow et al., 2007). Hence, this can be incorporated by adding an appropriate standard deviation with the mean and standard deviation for the process variation in our framework. Moreover, temperature variation can increase or decrease the step size on each kernel's accumulation capacitor. Large deviation (higher than 3-sigma of the process variation) from the nominal step size due to temperature can affect the classification accuracy.

To validate our Hspice simulations generated curve-fitting function's prediction accuracy, we have tested 1,000 random cases. In these test cases, we have used a kernel size of 3 $\times$ 3, where

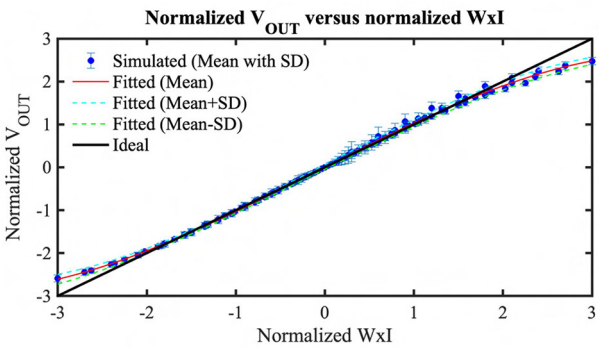


FIGURE 9
A scatter plot with standard deviation comparing the pixel output voltage to ideal multiplication value of Weights × Input activation (Normalized W × I).

the weight values are generated randomly. Moreover, the number of input event spikes and time instants for the input spikes are also randomly generated. Note these random tests utilize 100 μ s of simulation time for the convolution phase to reduce the total simulation time. As mentioned earlier, utilizing kernel-dependent nullifying current source, high-V_T weight transistors and a switch to disconnect the kernel's capacitor exhibits a maximum of 22 mV error in the worst-case scenario. Hence, random HSpice tests ignoring a long time (1 ms of simulation time length for each event stream of our neural network model) will not incur any significant accuracy issues for these 1,000 random tests. Among 1,000 random tests, only 100 test results (for clear visibility) are shown in Figure 10. The figure shows the curve-fitted mean and mean± standard deviation predictions of our proposed analog MAC operations with HSpice-generated simulation results. We have used a 3rd order single variable (normalized weight times input event spikes) polynomial to generate the curve fitting functions (mean, mean ± standard deviation) considering 0.55% mean RMSE of our analog MAC to minimize the computation complexity in our algorithmic framework while maintaining high accuracy. It can clearly be seen that the predicted mean output follows the Hspice results closely, and the HSpice outputs fall between the mean± standard deviation value.

3.2. Circuit-algorithm co-optimization of spiking CNN backbone subject to P^{2}M constraints

In our proposed neuromorphic-P^{2}M architecture, we have utilized a kernel-dedicated capacitor to enable instantaneous and massively parallel spatio-temporal convolution operation across different channels. We need a kernel-dedicated capacitor to preserve the temporal information of input DVS spikes across different channels simultaneously. Moreover, there is a direct trade-off between the acceptable leakage and capacitance value (a large capacitor incurs a large area; however, it results in lower leakage). Almost 47% of the area in our P^{2}M array is occupied by the capacitors. Hence, we have reduced the number of channels in our spiking CNN models compared to the baseline neural architecture not to incur any area overhead while preserving the model accuracy. In addition, the leakage also limits the length of each algorithmic time step in our algorithmic framework. We have also reduced the time length in our neural network model to minimize the kernel-dependent leakage error of our custom first convolution layer. Moreover, to reduce the amount of data transfer between the P^{2}M architecture and the backend hardware processing of the remaining spiking CNN layers, we have avoided the max pooling layer and instead used a stride of 2 in the P^{2}M convolutional layer. Lastly, we incorporate the Monte Carlo variations in the proposed non-linear custom convolutional layer explained above in our algorithmic framework. In particular, we have estimated the mean and standard deviation of the output of the custom convolutional layer from extensive circuit simulations. We then train our spiking CNN with the addition of the standard deviation as noise to the mean output of the convolutional layer. This noise addition during training is crucial to increase the robustness of our spiking CNN models, as otherwise, our models would incur a drastic drop in test accuracy.

In this work, we have evaluated our P^{2}M paradigm on complex neuromorphic datasets where each event is at least a few seconds long. Hence, with a timestep length of 1 ms, we will require more than thousands of total time steps to train our SNNs with these neuromorphic datasets. This is impractical (it would require more than a year to train one SNN model on the DVS Gesture dataset) in modern GPUs typically used for training SNNs. To mitigate this problem, we only employed a small timestep length (1 ms) in our first P^{2}M-implemented layer where the weights are kept frozen while the remaining layers implemented outside the sensor are trained with a large timestep length that leads to a small number of total time steps. The weights in the P^{2}M-implemented layer are obtained from a baseline SNN where all the layers have the same large timestep length. Thus, the length of the timestep impacts the trainability of the SNNs. It also affects the temporal information injected into the SNN, i.e., as the length of the timestep reduces, the SNN can extract more fine-grained temporal features, which can potentially improve the inference performance at the cost of reduced trainability. We expect to reduce energy consumption by increasing the time step length as that would inject a smaller number of spikes into the network (a constant large incoming synaptic input can emit more spikes if the number of time steps is increased, i.e., the time step length is decreased).

4. Experimental results

4.1. Benchmarking dataset and model

This article focuses on the potential use of P^{2}M for event-driven neuromorphic tasks where the goal is to classify each video sample captured by the DVS cameras. In particular, we evaluate our P^{2}M approach on two large-scale popular neuromorphic benchmarking datasets.

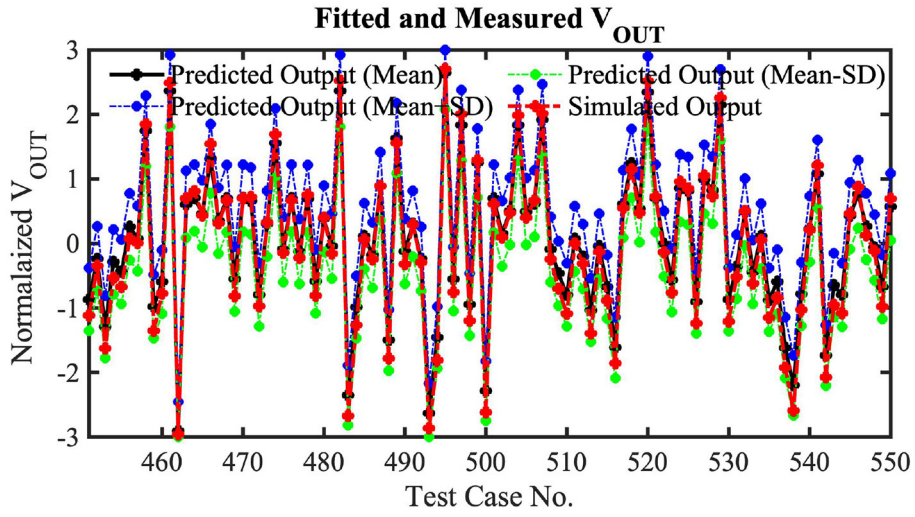


FIGURE 10
One hundred random HSpice simulation results for 3 × 3 Kernel benchmarking with the fitted equations.

4.1.1. DVS128-gesture

The IBM DVS128-Gesture (Amir et al., 2017) is a neuromorphic gesture recognition dataset with a temporal resolution in μs range and a spatial resolution of 128×128. It consists of 11 gestures (1,000 samples each), such as hand clap, arm roll, etc., recorded from 29 individuals under three illumination conditions, and each gesture has an average duration of 6 s. To the best of our knowledge, it is the most challenging open-source neuromorphic dataset with the most precise temporal information.

4.1.2. NMNIST

The neuromorphic MNIST (Orchard et al., 2015) dataset is a converted dataset from MNIST. It consists of 50K training images and 10K validation images. We preprocess it in the same way as in N-Caltech 101. We resize all our images to 34×34.

For these datasets, we apply a 9:1 train-valid split. We use the Spikingjelly package (Fang et al., 2020) to process the data and integrate them into a fixed time interval of 1 ms based on the kernel's capacitor retention time supported by our neuromorphic-P^{2}M circuit. However, such a small integration time would lead to a large number of time steps for the neuromorphic datasets considered in this work whose input samples are at least a few seconds long. This would significantly exacerbate the training complexity. To mitigate this concern, we first pre-train a spiking CNN model with a large integration time in the order of seconds (i.e., with a small number of time steps) without any P^{2}M circuit constraints. We then decrease the integration time of the first spiking convolutional layer for P^{2}M implementation and integrate the spikes in the second interval such that the network from the second layer processes the input with only a few time steps. We fine-tune this network from the second layer while freezing the first layer since training the first layer significantly increases the memory complexity due to a large number of time steps. This is because the gradients of the first layer need to be unrolled across all the time steps.

We use four convolutional layers, followed by two linear layers at the end with 512 and 10 neurons, respectively. Each convolutional layer is followed by a batch normalization layer, spiking LIF layer, and max pooling layer.

4.2. Classification accuracy

We evaluated the performance of the baseline and P^{2}M custom spiking CNN models on the two datasets illustrated above in Table 1. Note that all these models are trained from scratch. As we can see, the custom convolution model does not incur any significant drop in accuracy for any of the two datasets. However, removing the state variable, i.e., the membrane potential in the first layer, leads to an average ∼ 5% drop in test accuracy. This might be because of the loss in the temporal information of the input spike integration from the DVS camera. Additional P^{2}M constraints, such as less number of channels and increased strides in the first convolutional layer (see Section 3.2), hardly incur any additional drop in accuracy. Overall, our P^{2}M-constrained models lead to an average 5.2% drop in test accuracy across the two datasets.

4.3. Analysis of energy consumption

We develop a circuit-algorithm co-simulation framework to characterize the energy consumption of our baseline and P^{2}M-implemented spiking CNN models for neuromorphic datasets. Note that we do not evaluate the latency of our models since that would depend heavily on the underlying hardware architecture and data flow of the backend hardware (i.e., the hardware processing the remaining layers of the CNN, excluding the first layer that is processed using our P^{2}M paradigm). The frontend energy ($E_{frontend}$) is comprised of sensor energy (E_{sens}) and communication energy (E_{com}), while the backend energy

TABLE 1 Comparison of the test accuracy of our P^{2}M enabled spiking CNN models with the baseline spiking CNN counterparts, where "MP" denotes membrane potential, "Custom conv." denotes the incorporation of the non-ideal model to the ML algorithmic framework, and "Reduced dimensionality" denotes the reduction in the number of channels in the first convolutional layer.

Dataset	MP 1st layer	Custom conv.	Reduced dimensionality	Accuracy (%)
DVS128-Gesture	✓	×	×	93.40
DVS128-Gesture	×	×	×	88.78
DVS128-Gesture	×	✓	×	88.54
DVS128-Gesture	×	✓	✓	88.36
NMNIST	✓	×	×	98.10
NMNIST	×	×	×	93.68
NMNIST	×	✓	×	93.44
NMNIST	×	✓	✓	93.12

TABLE 2 Energy estimation for different hardware components.

Model type	Sensing energy (mJ) (E_{sens})	Comm energy (pJ/bit) ($e_{comm} = e_{sens-to-tx} + e_{tx}$)	MAC energy (pJ) (e_{mac})	MAdds energy (pJ) (e_{ac})
P^{2}M (ours)	26.588	4.1	1.568	0.03
Baseline	26.032	4.1	1.568	0.03

The energy values are measured for designs in 22 nm CMOS technology. Note, the sensing energy (E_{sens}) of our model includes the convolution energy for P^{2}M as the convolution is performed as a part of the sensing operation. The communication energy (e_{comm}) includes both the energy consumption of sending the address bits from the sensor to the transmitter ($e_{sens-to-tx}$) and wireless transmitter energy (e_{tx}). For e_{mac} and e_{ac}, we convert the corresponding value in 45 nm to that of 22 nm by following standard scaling strategy (Stillmaker and Baas, 2017).

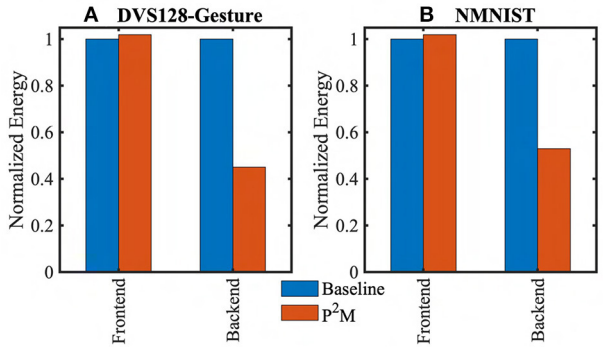


FIGURE 11
Comparison of the energy consumption between baseline and P^{2}M implementations of spiking CNNs to process neuromorphic images from **(A)** DVS128-Gesture, and **(B)** NMNIST datasets.

($E_{backend}$) to process the SNN layers (excluding the first layer for the P^{2}M implementation) is primarily composed of the accumulation operations incurred by the spiking convolutional layers (E_{ac}) and the parameter read (E_{read}) costs. Assuming T denotes the total number of time steps and s denotes the sparsity. The energy components can be approximated as follows:

$$E_{frontend} \approx \underbrace{e_{event} * N_{event} + E_{bias}}_{E_{sens}} + \underbrace{(e_{sens-to-tx} + e_{tx}) * N_{event}}_{E_{com}} \quad (1)$$

$$E_{backend} \approx \underbrace{e_{ac} * N_{ac} * s * T}_{E_{ac}} + \underbrace{e_{read} * N_{read}}_{E_{read}} \quad (2)$$

Here, e_{event} represents per-pixel sensing energy, N_{event} denotes the number of events communicated from the sensor to the backend, and E_{bias} is the biasing energy for the DVS pixel array considering the dataset duration. In addition, $e_{sens-to-tx}$ is the communication energy to send the address bits from the sensor node to the transmitter, and e_{tx} is the wireless transmission energy to the backend. Note that the first convolutional layer of the SNN in the baseline implementation requires MAC operations, and hence, we need to replace e_{ac} with the MAC energy e_{mac} and use s=1. For a spiking convolutional layer that takes an input $\mathbf{I} \in R^{h_i \times w_i \times c_i}$ and weight tensor $\boldsymbol{\theta} \in R^{k \times k \times c_i \times c_o}$ to produce output $\mathbf{O} \in R^{h_o \times w_o \times c_o}$, the N_{ac} (Datta et al., 2021, 2022b; Kundu et al., 2021a,b) and N_{read} can be computed as

$$N_{ac} = h_o * w_o * k^2 * c_i * c_o \quad (3)$$

$$N_{read} = k^2 * c_i * c_o \quad (4)$$

The energy values we have used to evaluate $E_{frontend}$ and $E_{backend}$ are presented in Table 2. While E_{sens} and $e_{sense-to-tx}$ are obtained from our circuit simulations, e_{tx} is obtained from Lin et al. (2021), and e_{ac} and e_{read} are obtained from Kang et al. (2018). Figure 11 shows the comparison of energy costs for standard vs P^{2}M-implemented spiking CNN models for the DVS

datasets. In particular, P^{2}M can yield a backend energy reduction of up to $\sim$ 2$\times$ with the cost of 2% increase in frontend energy only. This reduction primarily comes from the reduced energy consumption in the backend since we offload the compute of the first convolutional layer of the SNN. This layer consumes more than 50% of the total backend energy since it involves expensive MAC operations (due to event accumulation before convolution computation), which consume $\sim$32$\times$ more energy compared to cheap accumulate operations (Horowitz, 2014) with 32-bit fixed point representation. Thus, the proposed neuromorphic-P^{2}M paradigm enables *in-situ* availability of the weight matrix within the array of DVS pixels (reducing the energy overhead associated with the transfer of weight matrix) while also significantly reducing energy-consumption of MAC operations by utilizing massively parallel non-von-Neumann analog processing-in-pixel.

5. Conclusion

We have proposed and implemented a novel in-pixel-in-memory processing paradigm for neuromorphic event-based sensors in this work. To the best of our knowledge, this is the first proposal to enable massively parallel, energy-efficient non-von-Neumann analog processing-in-pixel for neuromorphic image sensors using novel weight-embedded pixels. Instead of generating event spikes based on the change in contrast of scenes, our proposed solutions can directly send the low-level output features of the convolutional neural network using a modified address event representation scheme. By leveraging advanced 3D integration technology, we can perform *in-situ* massively parallel charge-based analog spatio-temporal convolution across the pixel array. Moreover, we have incorporated the hardware (non-linearity, process variation, leakage) constraints of our analog computing elements as well as area consideration (limiting the maximum number of channels of the first neural network layer) into our algorithmic framework. Our P^{2}M-enabled spiking CNN model yields an accuracy of 88.36% on the IBM DVS128-Gesture dataset and achieved $\sim$ 2$\times$ backed energy reduction compared to the conventional system.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://research.ibm.com/interactive/dvsgesture/, https://www.garrickorchard.com/datasets/n-mnist.

Author contributions

MK proposed the use of P^{2}M for the neuromorphic vision sensor and developed the corresponding circuit simulation framework. GD developed the baseline and P^{2}M-constrained algorithmic framework with the help of ZW. APJ and ARJ proposed the idea of P^{2}M. MK and GD wrote the majority of the paper. ARJ and PAB supervised the research and reviewed the manuscript extensively. All authors reviewed the manuscript. All authors contributed to the article and approved the submitted version.

Funding

This work was funded in part by the DARPA HR00112190120 award.

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.

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)*, 7388–7397.

Beltrán, J., Guindel, C., Cortés, I., Barrera, A., Astudillo, A., Urdiales, J., et al. (2020). "Towards autonomous driving: a multi-modal 360° perception proposal," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)* (IEEE), 1–6.

Boahen, K. A. (2004). A burst-mode word-serial address-event link-I: Transmitter design. *IEEE Trans. Circ. Syst. I Regular Pap.* 51, 1269–1280. doi: 10.1109/TCSI.2004.830703

Bose, L., Dudek, P., Chen, J., Carey, S. J., and Mayol-Cuevas, W. W. (2020). "Fully embedding fast convolutional networks on pixel processor arrays," in *European Conference on Computer Vision* (Springer), 488–503.

Chai, Y. (2020). In-sensor computing for machine vision. *Nature* 579, 32–33 doi: 10.1038/d41586-020-00592-6

Chen, G., Cao, H., Conradt, J., Tang, H., Rohrbein, F., and Knoll, A. (2020). Event-based neuromorphic vision for autonomous driving: a paradigm shift for bio-inspired visual sensing and perception. *IEEE Signal Process. Mag.* 37, 34–49. doi: 10.1109/MSP.2020.2985815

Chen, Z., Zhu, H., Ren, E., Liu, Z., Jia, K., Luo, L., et al. (2019). Processing near sensor architecture in mixed-signal domain with cmos image sensor of convolutional-kernel-readout method. *IEEE Trans. Circ. Syst. I Reg. Pap.* 67, 389–400. doi: 10.1109/TCSI.2019.2937227

Cheng, W., Luo, H., Yang, W., Yu, L., and Li, W. (2020). Structure-aware network for lane marker extraction with dynamic vision sensor. *arXiv preprint arXiv:2008.06204*. doi: 10.48550/arXiv.2008.06204

Datta, G., and Beerel, P. A. (2022). "Can deep neural networks be converted to ultra low-latency spiking neural networks?" in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 718–723.

Datta, G., Deng, H., Aviles, R., and Beerel, P. A. (2022a). Towards energy-efficient, low-latency and accurate spiking LSTMs. *arXiv preprint arXiv:2110.05929*. doi: 10.48550/arXiv.2210.12613

Datta, G., Kundu, S., and Beerel, P. A. (2021). "Training energy-efficient deep spiking neural networks with single-spike hybrid input encoding," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Datta, G., Kundu, S., Jaiswal, A. R., and Beerel, P. A. (2022b). ACE-SNN: algorithm-hardware co-design of energy-efficient & low-latency deep spiking neural networks for 3D image recognition. *Front. Neurosci.* 16, 815258. doi: 10.3389/fnins.2022.815258

Datta, G., Kundu, S., Yin, Z., Lakkireddy, R. T., Mathai, J., Jacob, A. P., et al. (2022c). A processing-in-pixel-in-memory paradigm for resource-constrained tinyml applications. *Sci. Rep.* 12:14396. doi: 10.1038/s41598-022-17934-1

Datta, G., Kundu, S., Yin, Z., Mathai, J., Liu, Z., Wang, Z., et al. (2022d). P2M-DeTrack: processing-in-Pixel-in-Memory for energy-efficient and real-time multi-object detection and tracking. *arXiv preprint arXiv:2205.14285*. doi: 10.1109/VLSI-SoC54400.2022.9939582

Datta, G., Yin, Z., Jacob, A., Jaiswal, A. R., and Beerel, P. A. (2022e). Toward efficient hyperspectral image processing inside camera pixels. *arXiv preprint arXiv:2203.05696*. doi: 10.48550/arXiv.2203.05696

Eki, R., Yamada, S., Ozawa, H., Kai, H., Okuike, K., Gowtham, H., et al. (2021). "9.6 a 1/2.3 inch 12.3 mpixel with on-chip 4.97 tops/w CNN processor back-illuminated stacked CMOS image sensor," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)* (IEEE), 154–156.

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly

Goel, A., Tung, C., Lu, Y.-H., and Thiruvathukal, G. K. (2020). "A survey of methods for low-power deep learning and computer vision," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)* (IEEE), 1–6.

Gow, R. D., Renshaw, D., Findlater, K., Grant, L., McLeod, S. J., Hart, J., et al. (2007). A comprehensive tool for modeling cmos image-sensor-noise performance. *IEEE Trans. Electron Dev.* 54, 1321–1329. doi: 10.1109/TED.2007.896718

Horowitz, M. (2014). "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 10–14.

Hsu, T.-H., Chen, G.-C., Chen, Y.-R., Lo, C.-C., Liu, R.-S., Chang, M.-F., et al. (2022). "A 0.8 v intelligent vision sensor with tiny convolutional neural network and programmable weights using mixed-mode processing-in-sensor technique for image classification," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)* (IEEE), 1–3.

Hsu, T.-H., Chen, Y.-R., Liu, R.-S., Lo, C.-C., Tang, K.-T., Chang, M.-F., et al. (2020). A 0.5-v real-time computational cmos image sensor with programmable kernel for feature extraction. *IEEE J. Solid State Circ.* 56, 1588–1596. doi: 10.1109/JSSC.2020.3034192

Jaiswal, A., and Jacob, A. P. (2021). *Integrated Pixel and Two-Terminal Non-Volatile Memory Cell and an Array of Cells for Deep In-sensor, In-Memory Computing*. US Patent 11195580.

Jiao, L., Zhang, R., Liu, F., Yang, S., Hou, B., Li, L., et al. (2022). New generation deep learning for video object detection: a survey. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 3195–3215. doi: 10.1109/TNNLS.2021.3053249

Jogin, M., Madhulika, M., Divya, G., Meghana, R., and Apoorva, S. (2018). "Feature extraction using convolution neural networks (CNN) and deep learning," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (IEEE), 2319–2323.

Kagawa, Y., Fujii, N., Aoyagi, K., Kobayashi, Y., Nishi, S., Todaka, N., et al. (2016). "Novel stacked CMOS image sensor with advanced cu2cu hybrid bonding," in *2016 IEEE International Electron Devices Meeting (IEDM)* (IEEE), 8–4.

Kagawa, Y., Hashiguchi, H., Kamibayashi, T., Haneda, M., Fujii, N., Furuse, S., et al. (2020). "Impacts of misalignment on 1μm pitch cu-cu hybrid bonding," in *2020 IEEE International Interconnect Technology Conference (IITC)* (IEEE), 148–150.

Kang, M., Lim, S., Gonugondla, S., and Shanbhag, N. R. (2018). An in-memory VLSI architecture for convolutional neural networks. *IEEE J. Emerg. Select. Top. Circ. Syst.* 8, 494–505. doi: 10.1109/JETCAS.2018.2829522

Kundu, S., Datta, G., Pedram, M., and Beerel, P. A. (2021a). "Spike-thrift: towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 3953–3962.

Kundu, S., Datta, G., Pedram, M., and Beerel, P. A. (2021b). Towards low-latency energy-efficient deep snns via attention-guided compression. *arXiv preprint arXiv:2107.12445*. doi: 10.48550/arXiv.2107.12445

Lefebvre, M., Moreau, L., Dekimpe, R., and Bol, D. (2021). "7.7 a 0.2-to-3.6 tops/w programmable convolutional imager soc with in-sensor current-domain ternary-weighted mac operations for feature extraction and region-of-interest detection," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)* (IEEE), 118–120.

Leñero-Bardallo, J. A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2011). A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE J. Solid State Circ.* 46, 1443–1455. doi: 10.1109/JSSC.2011.2118490

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128x128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circ.* 43, 566–576. doi: 10.1109/JSSC.2007.914337

LiKamWa, R., Hou, Y., Gao, J., Polansky, M., and Zhong, L. (2016). Redeye: analog convnet image sensor architecture for continuous mobile vision. *ACM SIGARCH Comput. Arch. News* 44, 255–266. doi: 10.1145/3007787.3001164

LiKamWa, R., Wang, Z., Carroll, A., Lin, F. X., and Zhong, L. (2014). "Draining our glass: an energy and heat characterization of google glass," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, 1–7.

Lin, J., and Boahen, K. (2009). "A delay-insensitive address-event link," in *2009 15th IEEE Symposium on Asynchronous Circuits and Systems* (IEEE), 55–62.

Lin, L., Ahmed, K. A., Salamani, P. S., and Alioto, M. (2021). "Battery-less IoT sensor node with pll-less wifi backscattering communications in a 2.5-μw peak power envelope," in *2021 Symposium on VLSI Circuits* (IEEE), 1–2.

Ma, T., Jia, K., Zhu, X., Qiao, F., Wei, Q., Zhao, H., et al. (2019). "An analog-memoryless near sensor computing architecture for always-on intelligent perception applications," in *2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)* (IEEE), 150–155.

Maheepala, M., Joordens, M. A., and Kouzani, A. Z. (2020). Low power processors and image sensors for vision-based iot devices: a review. *IEEE Sensors J.* 21, 1172–1186. doi: 10.1109/JSEN.2020.3015932

Mansour, R. F., Escorcia-Gutierrez, J., Gamarra, M., Villanueva, J. A., and Leal, N. (2021). Intelligent video anomaly detection and classification using faster rcnn with deep reinforcement learning model. *Image Vision Comput.* 112, 104229. doi: 10.1016/j.imavis.2021.104229

Maqueda, A. I., Loquercio, A., Gallego, G., García, N., and Scaramuzza, D. (2018). "Event-based vision meets deep learning on steering prediction for self-driving cars," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 5419–5427.

Miura, T., Sakakibara, M., Takahashi, H., Taura, T., Tatani, K., Oike, Y., et al. (2019). "A 6.9 μm pixel-pitch 3d stacked global shutter cmos image sensor with 3m cu-cu connections," in *2019 International 3D Systems Integration Conference (3DIC)* (IEEE), 1–2.

Nguyen, A., Do, T.-T., Caldwell, D. G., and Tsagarakis, N. G. (2019). "Real-time 6dof pose relocalization for event cameras with stacked spatial LSTM networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9, 437. doi: 10.3389/fnins.2015.00437

Pinkham, R., Berkovich, A., and Zhang, Z. (2021). Near-sensor distributed DNN processing for augmented and virtual reality. *IEEE J. Emerg. Select. Top. Circ. Syst.* 11, 663–676. doi: 10.1109/JETCAS.2021.3121259

Seo, M.-W., Chu, M., Jung, H.-Y., Kim, S., Song, J., Lee, J., et al. (2021). "A 2.6 e-rms low-random-noise, 116.2 mw low-power 2-mp global shutter cmos image sensor with pixel-level adc and in-pixel memory," n *2021 Symposium on VLSI Technology* (IEEE), 1–2.

Son, B., Suh, Y., Kim, S., Jung, H., Kim, J.-S., Shin, C., et al. (2017). "4.1 a 640× 480 dynamic vision sensor with a 9μm pixel and 300meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (IEEE), 66–67.

Stillmaker, A., and Baas, B. (2017). Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm. *Integration* 58, 74–81. doi: 10.1016/j.vlsi.2017.02.002

Wu, X., Li, W., Hong, D., Tao, R., and Du, Q. (2021). Deep learning for unmanned aerial vehicle-based object detection and tracking: a survey. *IEEE Geosci. Remote Sens. Mag.* 10, 91–124. doi: 10.1109/MGRS.2021.3115137

Xie, J., Zheng, Y., Du, R., Xiong, W., Cao, Y., Ma, Z., et al. (2021). Deep learning-based computer vision for surveillance in its: evaluation of state-of-the-art methods. *IEEE Trans. Vehicular Technol.* 70, 3027–3042. doi: 10.1109/TVT.2021.3065250

Xu, H., Lin, N., Luo, L., Wei, Q., Wang, R., Zhuo, C., et al. (2021). Senputing: an ultra-low-power always-on vision perception chip featuring the deep fusion of sensing and computing. *IEEE Trans. Circ. Syst. I Reg. Pap.* 69, 232–243. doi: 10.1109/TCSI.2021.3090668

Xu, H., Nazhamaiti, M., Liu, Y., Qiao, F., Wei, Q., Liu, X., et al. (2020). "Utilizing direct photocurrent computation and 2d kernel scheduling to improve in-sensor-processing efficiency," in *2020 57th ACM/IEEE Design Automation Conference (DAC)* (IEEE), 1–6.

Zhou, F., and Chai, Y. (2020). Near-sensor and in-sensor computing. *Nat. Electron.* 3, 664–671. doi: 10.1038/s41928-020-00501-9

Zhu, A. Z., Yuan, L., Chaney, K., and Daniilidis, K. (2018). EV-flowNet: self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*. doi: 10.48550/arXiv.1802.06898

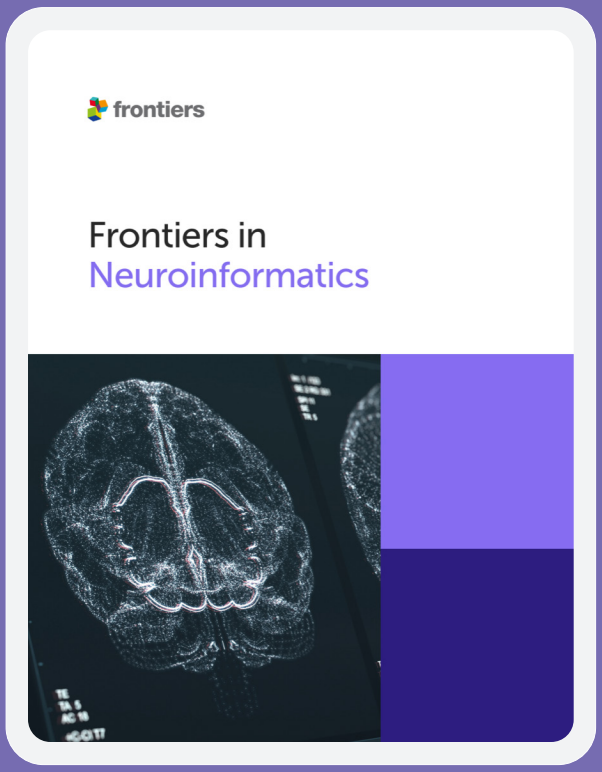
Frontiers in Neuroinformatics

Leading journal supporting neuroscience in the information age

Part of the most cited neuroscience journal series, developing computational models and analytical tools used to share, integrate and analyze experimental data about the nervous system functions.

Discover the latest Research Topics

See more →



Frontiers

Avenue du Tribunal-Fédéral 34
1005 Lausanne, Switzerland
frontiersin.org

Contact us

+41 (0)21 510 17 00
frontiersin.org/about/contact