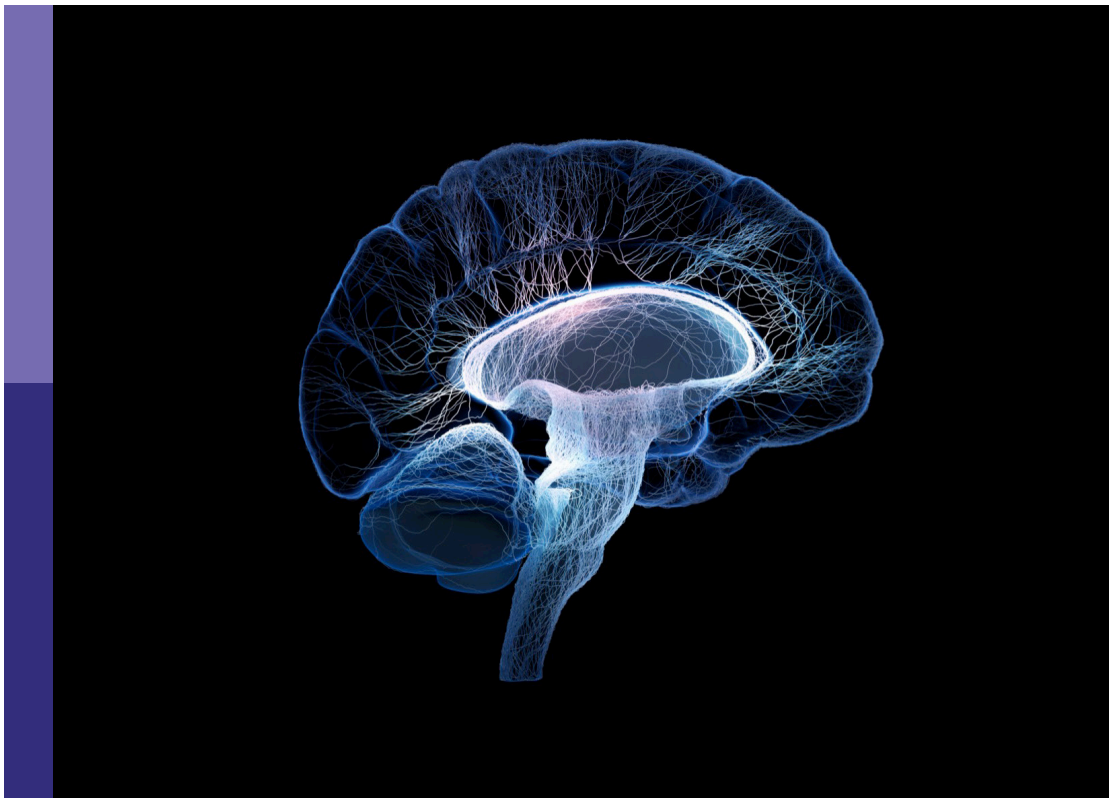
frontiers | Research Topics

Understanding and bridging the gap between neuromorphic computing and machine learning, volume II

Edited by
Huajin Tang, Lei Deng and Kaushik Roy

Published in
Frontiers in Neuroscience
Frontiers in Computational Neuroscience



ISSN 1664-8714
ISBN 978-2-8325-5363-3
DOI 10.3389/978-2-8325-5363-3

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

Understanding and bridging the gap between neuromorphic computing and machine learning, volume II

Topic editors

Huajin Tang — Zhejiang University, China

Lei Deng — Tsinghua University, China

Kaushik Roy — Purdue University, United States

Citation

Tang, H., Deng, L., Roy, K., eds. (2024). *Understanding and bridging the gap between neuromorphic computing and machine learning, volume II*. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-8325-5363-3

Table of contents

frontiers | Frontiers in Computational Neuroscience

TYPE Editorial
PUBLISHED 03 October 2024
DOI 10.3389/fncom.2024.1455530

OPEN ACCESS

EDITED BY
Nicolangelo Iannella,
University of Oslo, Norway

REVIEWED BY
Georgios Detorakis,
Independent Researcher, Irvine, United States
Tommaso Zanotti,
University of Modena and Reggio Emilia, Italy

*CORRESPONDENCE
Lei Deng
leideng@mail.tsinghua.edu.cn

RECEIVED 27 June 2024
ACCEPTED 09 September 2024
PUBLISHED 03 October 2024

CITATION
Deng L, Tang H and Roy K (2024) Editorial: Understanding and bridging the gap between neuromorphic computing and machine learning, volume II.
Front. Comput. Neurosci. 18:1455530.
doi: 10.3389/fncom.2024.1455530

Editorial: Understanding and bridging the gap between neuromorphic computing and machine learning, volume II

Lei Deng[1]*, Huajin Tang[2,3] and Kaushik Roy[4]

[1]Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing, China, [2]College of Computer Science and Technology, The State Key Lab of Brain-Machine Intelligence, Zhejiang University, Hangzhou, China, [3]MOE Frontier Science Center for Brain Science and Brain-Machine Integration, Zhejiang University, Hangzhou, China, [4]Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

KEYWORDS

spiking neural networks, neuromorphic computing, neuromorphic hardware, artificial neural networks, machine learning

Editorial on the Research Topic
Understanding and bridging the gap between neuromorphic computing and machine learning, volume II

Introduction

Pursuing intelligence is a long-term goal of the human, toward which two routes have been paved on the road: neuromorphic computing driven by neuroscience and machine learning driven by computer science (Pei et al., 2019). Spiking neural networks (SNNs) and neuromorphic chips (Basu et al., 2022; Christensen et al., 2022) dominate the neuromorphic computing domain, while artificial neural networks (ANNs) and machine learning accelerators (Deng et al., 2020) dominate the machine learning domain. Neuromorphic computing with efficient models and hardware has shown energy efficiency superiority (Renner et al., 2021), however, still lies in its infant stage and presents a gap in terms of accuracy and applications compared to the mature machine learning ecosystem.

To this end, we proposed a Research Topic, named "*Understanding and bridging the gap between neuromorphic computing and machine learning*," in Frontiers in Neuroscience and Frontiers in Computational Neuroscience in 2019, and have successfully published 14 articles on neuromorphic computing and machine learning (Deng et al., 2021). Encouraged by such positive impetus for the neuromorphic computing community, we relaunched the Research Topic in 2022. This time, we have accepted 11 submissions in the end. The scope of these works covers neuromorphic models and algorithms, hardware implementation, and programming frameworks.

Neuromorphic models and algorithms

SNNs encode information in spike events and process information using neural dynamics, which differ from ANNs. Due to the complicated spatiotemporal dynamics

and non-differentiable spike activities, the SNN domain uses plasticity-based unsupervised learning algorithms (Diehl and Cook, 2015) for a long time but suffers from low accuracy. To break the bottleneck of lacking effective learning algorithms, the sophisticated backpropagation method in machine learning has been introduced into SNNs (Lee et al., 2016; Wu et al., 2018), which greatly improves the performance of SNNs and thus extending the scope of neuromorphic models and applications (Yao et al., 2023). A comprehensive survey of the direct learning-based deep SNNs can be found in the review article from Guo et al., mainly categorized into accuracy improvement methods, efficiency improvement methods, and temporal dynamics utilization methods. Besides this review article, we accepted six articles on neuromorphic models and algorithms in this Research Topic, which are briefly summarized below.

With the extensive researches on SNN learning algorithms, how to combine the complementary advantages of bio-plausible unsupervised learning and powerful supervised learning is becoming an emerging and interesting Research Topic (Wu et al., 2022). Some learning rules for SNNs adopt a three-factor Hebbian update: a global error signal modulates local synaptic updates within each layer. Unfortunately, the global signal for a given layer requires processing multiple samples concurrently, but the brain only sees a single sample at a time. Daruwalla and Lipasti propose a new three-factor update rule where the global signal correctly captures information across samples via an auxiliary memory network. The auxiliary memory network can be trained a priori independently of the dataset being used with the primary network. This work presents an explicit connection between working memory and synaptic updates.

Most learning methods only consider the plasticity of synaptic weights. To improve the learning performance and biological plausibility, Wang proposes a new supervised learning algorithm for SNNs based on the typical SpikeProp method, in which both the synaptic weights and delays are adjustable parameters. Sun et al. further combine learnable delays, local skip-connections and an auxiliary loss term to enhance the accuracy and stability of SNNs, which is validated on spoken word recognition benchmarks.

Current SNNs can only focus on information within a short time period, which makes it difficult for them to make effective decisions based on global information. Chen et al. propose SNNs with working memory (SNNWM) to handle spike trains segment by segment, inspired by recent neuroscience advances. This model can help SNNs obtain global information and reduce the information redundancy between adjacent time steps. To better leverage the temporal potential of SNNs, Wu X. et al. propose a self-attention-based temporal-channel joint attention SNN (STCA-SNN) with end-to-end training by inferring attention weights along both temporal and channel dimensions concurrently. This method can model global temporal and channel information correlations, enabling the network to learn "what" and "when" to attend simultaneously.

High energy efficiency is a well-known advantage of SNNs, which is tightly associated with the sparse spike activities. To reduce the redundant spike counts, Fois and Girau propose weight-temporally coded representation learning (W-TCRL), which utilizes temporally coded inputs and leads to lower spike counts and improved energy efficiency. Furthermore, they introduce a novel spike-timing-dependent plasticity (STDP) learning rule for stable learning of relative latencies within the synaptic weight distribution. This work improves the image reconstruction error while achieving significantly higher sparsity in spike activities.

Hardware implementation

Neuromorphic models enjoy low computational costs owing to the binary spike representation and sparse operations. However, directly executing SNNs on GPUs without tailored optimization is inefficient. Neuromorphic hardware is designed for the efficient execution of SNNs via event-driven computing (Merolla et al., 2014). In this Research Topic, we accepted three articles on hardware implementation of SNNs in this Research Topic.

Probabilistic sampling is an effective approach for making SNNs achieve Bayesian inference, but also a time-consuming operation on conventional computing architectures. To address this problem, Li et al. design a specific accelerator on FPGA to improve the execution of SNN sampling models by parallelization. The streaming pipelining and array partitioning operations are used to achieve acceleration with the lowest resource consumption. The Python productivity for Zynq (PYNQ) framework is combined to efficiently migrate models onto FPGA. This work promises implementing complex probabilistic model inference in embedded systems.

Huang et al. propose the MAC array for the acceleration of SNN inference, which is a parallel architecture on each processing element of SpiNNaker 2 (Liu et al., 2018). The authors further investigate the parallel acceleration algorithms for collaborating with multi-core MAC arrays. The proposed Echelon Reorder model information densification algorithm can achieve efficient spatiotemporal load balancing and optimization performance with the help of the adapted multi-core two-stage splitting and authorization deployment strategies. This work expands the application scope of the general sparse matrix-matrix multiplication (SpGEMM) issue to SNNs.

The decentralized manycore architecture with high computing parallelism and memory locality is widely adopted by neuromorphic chips. However, its fragmented memories and decentralized execution lowers the resource utilization and processing efficiency. Wang et al. propose the mapping limit concept which points out the resource saving upper limit during logical and physical mapping when deploying neural networks onto neuromorphic chips. A closed-loop mapping strategy with an asynchronous 4D model partition for logical mapping and a Hamilton loop algorithm (HLA) for physical mapping are elaborated. Their methods and performance gains are validated on the TianjicX neuromorphic chip (Ma et al., 2022), which is helpful for building a general and efficient mapping framework for neuromorphic hardware.

Programming frameworks

Software is one of the key components in the ecosystem of neuromorphic computing (Fang et al., 2023), which is sometimes

more important than the hardware itself because it determines how much practical efficiency we can gain from the peak efficiency of hardware. In this Research Topic, we accepted one article on the programming framework for neuromorphic models in this Research Topic.

Wu Z. et al. introduce a user-friendly brain-inspired deep learning (BIDL) framework for generalized and lightweight spatiotemporal processing (STP). Researchers can use the framework to construct deep neural networks which leverage neural dynamics for processing spatiotemporal information and ensure high accuracy. The framework is compatible for various types of spatiotemporal data such as videos, dynamic vision sensor (DVS) signals, 3D medical images, and natural languages. Moreover, BIDL incorporates several optimizations such as iteration representation, state-aware computational graph, and built-in neural functions for easy deployment on GPUs and neuromorphic chips. By facilitating the exploration of different neural models and enabling global-local co-learning, BIDL shows potential to drive future advancements in bio-inspired research.

Conclusion

Neuromorphic computing is a neuroscience-driven domain in pursuing brain-like intelligence, which is an important route distinct from machine learning. Although neuromorphic systems have not yet demonstrated superior performance over machine learning systems in main stream intelligent tasks, we believe it can be significantly improved when the neuromorphic ecosystem is constructed and becomes iterative between algorithms, models, hardware, software, and benchmarks. This Research Topic is a quite minor step. We hope future works can really bridge the gap between neuromorphic computing and machine learning, along the way to reach the long-term goal of mimicking brain 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

LD: Writing – original draft. HT: Writing – review & editing. KR: Writing – review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was supported in part by the National Natural Science Foundation of China (Nos. 62276151 and 62106119), CETC Haikang Group-Brain Inspired Computing Joint Research Center, and Chinese Institute for Brain Research, Beijing.

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.

The author(s) declared that they were an editorial board member of Frontiers, at the time of submission. This had no impact on the peer review process and the final decision.

Publisher's note

References

Basu, A., Deng, L., Frenkel, C., and Zhang, X. (2022). "Spiking neural network integrated circuits: a review of trends and future directions," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*. Newport Beach, CA: IEEE.

Christensen, D. V., Dittmann, R., Linares-Barranco, B., Sebastian, A., Le Gallo, M., Redaelli, A., et al. (2022). 2022 roadmap on neuromorphic computing and engineering. *Neuromorph. Comput. Eng.* 2:e022501. doi: 10.1088/2634-4386/ac4a83

Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proc. IEEE* 108, 485–532. doi: 10.1109/JPROC.2020.2976475

Deng, L., Tang, H., and Roy, K. (2021). Understanding and bridging the gap between neuromorphic computing and machine learning. *Front. Comput. Neurosci.* 15:665662. doi: 10.3389/978-2-88966-742-0

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

Fang, W., Chen, Y., Ding, J., Yu, Z., Masquelier, T., Chen, D., et al. (2023). SpikingJelly: an open-source machine learning infrastructure platform for spike-based intelligence. *Sci. Adv.* 9:eadi1480. doi: 10.1126/sciadv.adi1480

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:228000. doi: 10.3389/fnins.2016.00508

Liu, C., Bellec, G., Vogginger, B., Kappel, D., Partzsch, J., Neumärker, F., et al. (2018). Memory-efficient deep learning on a SpiNNaker 2 prototype. *Front. Neurosci.* 12:416510. doi: 10.3389/fnins.2018.00840

Ma, S., Pei, J., Zhang, W., Wang, G., Feng, D., Yu, F., et al. (2022). Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots. *Sci. Robot.* 7:eabk2948. doi: 10.1126/scirobotics.abk2948

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

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Renner, A., Sheldon, F., Zlotnik, A., Tao, L., and Sornborger, A. (2021). The backpropagation algorithm implemented on spiking neuromorphic hardware. *arXiv preprint arXiv:2106.07030*. doi: 10.21203/rs.3.rs-701752/v1

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:323875. doi: 10.3389/fnins.2018.00331

Wu, Y., Zhao, R., Zhu, J., Chen, F., Xu, M., Li, G., et al. (2022). Brain-inspired global-local learning incorporated with neuromorphic computing. *Nat. Commun.* 13:65. doi: 10.1038/s41467-021-27653-2

Yao, M., Zhao, G., Zhang, H., Hu, Y., Deng, L., Tian, Y., et al. (2023). Attention spiking neural networks. *IEEE Trans. Pat. Anal. Machine Intell.* 45, 9393–9410. doi: 10.1109/TPAMI.2023.3241201

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 18 May 2023
DOI 10.3389/fnins.2023.1168864

OPEN ACCESS

EDITED BY
Huajin Tang,
Zhejiang University, China

REVIEWED BY
Liliana Ibeth Barbosa Santillan,
University of Guadalajara, Mexico
Joseph Friedman,
The University of Texas at Dallas, United States

*CORRESPONDENCE
Jing Pei
✉ peij@mail.tsinghua.edu.cn

RECEIVED 18 February 2023
ACCEPTED 26 April 2023
PUBLISHED 18 May 2023

CITATION
Wang S, Yu Q, Xie T, Ma C and Pei J (2023) Approaching the mapping limit with closed-loop mapping strategy for deploying neural networks on neuromorphic hardware. *Front. Neurosci.* 17:1168864. doi: 10.3389/fnins.2023.1168864

Approaching the mapping limit with closed-loop mapping strategy for deploying neural networks on neuromorphic hardware

Song Wang, Qiushuang Yu, Tiantian Xie, Cheng Ma and Jing Pei*

Department of Precision Instrument, Center for Brain-Inspired Computing Research (CBICR), Tsinghua University, Beijing, China

The decentralized manycore architecture is broadly adopted by neuromorphic chips for its high computing parallelism and memory locality. However, the fragmented memories and decentralized execution make it hard to deploy neural network models onto neuromorphic hardware with high resource utilization and processing efficiency. There are usually two stages during the model deployment: one is the logical mapping that partitions parameters and computations into small slices and allocate each slice into a single core with limited resources; the other is the physical mapping that places each logical core to a physical location in the chip. In this work, we propose the mapping limit concept for the first time that points out the resource saving upper limit in logical and physical mapping. Furthermore, we propose a closed-loop mapping strategy with an asynchronous 4D model partition for logical mapping and a Hamilton loop algorithm (HLA) for physical mapping. We implement the mapping methods on our state-of-the-art neuromorphic chip, TianjicX. Extensive experiments demonstrate the superior performance of our mapping methods, which can not only outperform existing methods but also approach the mapping limit. We believe the mapping limit concept and the closed-loop mapping strategy can help build a general and efficient mapping framework for neuromorphic hardware.

KEYWORDS

neuromorphic chip, logical mapping, physical mapping, mapping limit, closed-loop mapping

1. Introduction

Deep neural networks (DNNs) have made a series of breakthroughs in many fields. With the exponential growth (Vaswani et al., 2017; Gholami et al., 2021) of parameters and computations of DNN models, the memory and computational costs are unaffordable for conventional (Von Neumann, 1993) architectures. To overcome the memory wall problem, the decentralized manycore architecture emerges in recent years for performing neural network workloads, which presents massive processing parallelism, memory locality, and multi-chip scalability (Painkras et al., 2013; Akopyan et al., 2015; Han et al., 2016; Jouppi et al., 2017; Parashar et al., 2017; Shin et al., 2017; Davies et al., 2018; Chen et al., 2019; Pei et al., 2019; Shao et al., 2019; Deng et al., 2020; Zimmer et al., 2020). Each functional core

contains independent computation and memory resources with close distance, and cores communicate through a flexible routing fabric (Wu et al., 2020). Due to the limited hardware resources in each core, a large neural network model has to be partitioned and mapped onto cores during deployment. The mapping process experiences two stages: logical mapping and physical mapping.

In the logical mapping stage, the requirements for computation and memory resources are important consideration factors for allocating cores. The parameters and the associated computations are divided into small slices through tensor dimension partition and each slice is allocated into a single core with limited hardware resources (Shao et al., 2019; Deng et al., 2020; Wu et al., 2020). For a convolutional layer, most previous work adopts the 2D partition to split the input channel (C_{in}) and the output channel (C_{out}) dimensions. However, partitioning the input channel dimension would generate partial sums (*psum*), which might degrade the model accuracy. To avoid the accuracy loss, the bit-width of *psum* has to be enlarged, which unfortunately results in longer communication latency and larger memory overhead.

The logical mapping only partitions a neural network and allocates the partitioned sub-networks to cores logically. This stage does not care the physical locations of cores on real hardware. The physical mapping places each logical core to a physical location in the chip, which greatly affects the communication latency and might cause the deadlock problem (Wu et al., 2020). The core placement optimization for minimized latency is actually an NP-hard problem (Myung et al., 2021) and the search space grows rapidly as the number of cores increases. Existing algorithms for physical mapping on a 2D mesh topology are usually heuristic.

In this work, we find that there exists a limit in mapping a neural network model onto the decentralized multi-core architecture widely used by neuromorphic hardware. To approach this limit for fully utilizing hardware resources, we propose the closed-loop mapping strategy. Specifically, in logical mapping, we propose an asynchronous 4D partition between input activations (IA) and weights (W) from four dimensions for reducing execution latency; in physical mapping, we propose a Hamilton Loop Algorithm (HLA) for deadlock-free core placement with reduced communication latency. With our optimization, the running speed and computing efficiency can be improved by 7.6 and 8.8×, respectively via the integration of the logical mapping and the physical mapping, compared with the synchronous partition. Moreover, the running speed and computing efficiency can approach the performance limit of hardware, which is validated on the TianjicX chip (Deng et al., 2018).

2. Preliminaries and related works

2.1. Graph representation

As aforementioned, mapping a neural network model onto a decentralized multi-core architecture has two stages: the logical mapping and the physical mapping, as illustrated in Figure 1.

The logical cores can be represented by a graph $G(V,\ E)$, thus the physical mapping can be viewed to place $G(V,\ E)$ on a circuit graph $T(U,\ S)$. V and U denote the sets of nodes, i.e., logical cores and physical cores, respectively; E and S denote the sets of edges, i.e., connections between logical cores and physical cores, respectively. Specifically, the physical mapping can be described as follows:

$$V \to U,\ s.t. = \begin{cases} |V| \le |U|, \\ \forall v_i \in V, map(v_i) \in U, \\ \forall v_i \ne v_j \in V, map(v_i) \ne map(v_j). \end{cases} \tag{1}$$

where v_i and v_j denote the i-th and j-th nodes (i.e., core) in V, respectively; $|V|$ and $|U|$ represent the numbers of logical and physical cores, respectively. Above constraints imply one-to-one mapping from logical cores to physical cores. Furthermore, we denote the weighted edges (#packets) between v_i and v_j as c_{ij} and denote the Manhattan distance between $map(v_i)$ and $map(v_j)$ as M_{ij}, i.e., $M_{ij} = |x_i - x_j| + |y_i - y_j|$ where $(x_i,\ y_i)$ and $(x_j,\ y_j)$ are the coordinates of the two physical cores on the 2D physical plane. Let's define $E|h|$ as the energy of transmitting a routing packet through a single hop distance, and define L_{ij} as the communication latency with a routing packet between $map(v_i)$ and $map(v_j)$, respectively. With the above definitions, the total communication cost C_{cost} (Myung et al., 2021), the average communication latency L (Amin et al., 2020), and the average power consumption P (Pei et al., 2019; Ma et al., 2020) can be calculated by

$$C_{cost} = \sum\nolimits_{\forall v_i, v_j \in V} c_{ij} \times M_{ij}, \tag{2}$$

$$L = avg(\frac{1}{N_i} \sum\nolimits_j c_{ij} \times L_{ij}), \tag{3}$$

$$P = \frac{C_{cost} \times E\left|h\right|}{T}. \tag{4}$$

where the N_i is the number of routing packets received by the physical core $map(v_i)$. The working period T can be approximately viewed as a fixed variable.

2.2. Logical mapping

At present, most researchers adopt 2D partition in logical mapping by splitting both input and output channel dimensions. The partition of the input channel dimension would compromise accuracy due to the accumulation of *psum*s, while the partition of the output channel dimension would cause the requirement for data reshaping in the next layer. Besides the 2D partition, some works such as Simba (Shao et al., 2019; Zimmer et al., 2020) and Tianjic (Pei et al., 2019; Deng et al., 2020) can support 4D partition further on the feature map width and height. However, current mapping strategies face some challenges as shown in Figure 2, including the data overlap between the feature map partition, the *psum* accumulation in the input channel partition, and the data reshaping in the output channel partition.

The additional storage overheads in a core can be generated by the row-wise overlap S_{row_add}, the column-wise overlap S_{col_add}, the *psum* S_{p_add}, and the reshaped data S_{re_add}, which also result in additional computation latency. The additional storage and computation latency can be obtained by

$$S_{add} = S_{row\_add} + S_{col\_add} + S_{p\_add} + S_{re\_add}, \tag{5}$$

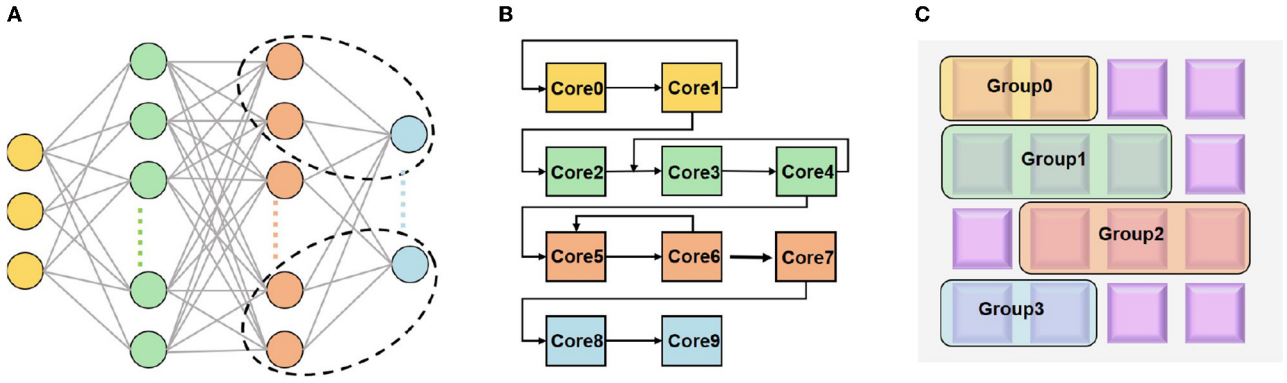


FIGURE 1
Illustration of mapping **(A)** a neural network onto neuromorphic hardware with two stages: **(B)** logical mapping; **(C)** physical mapping.

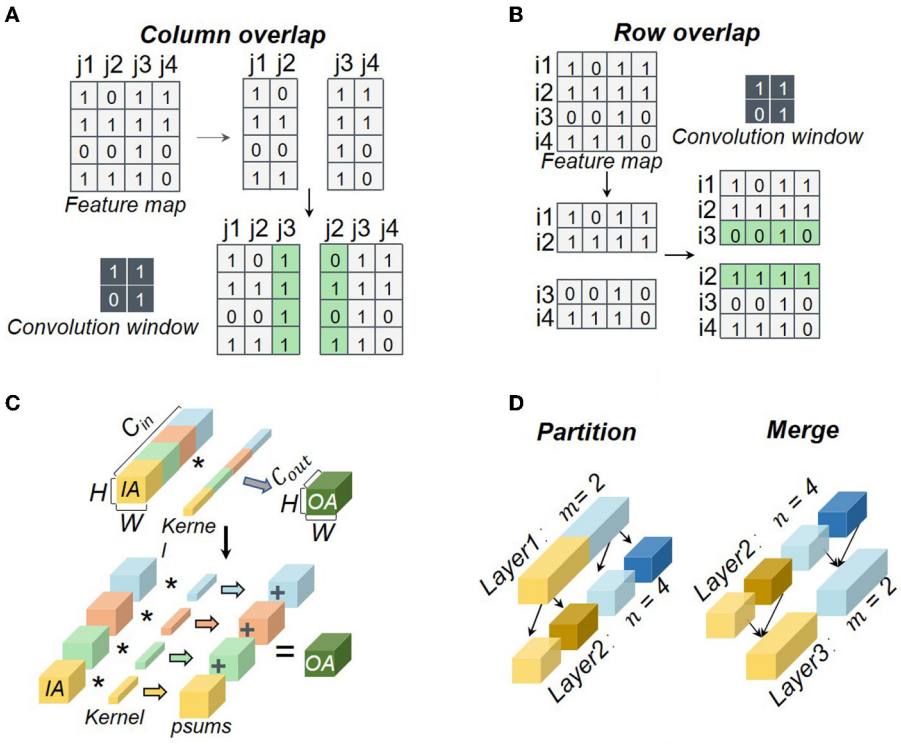


FIGURE 2
Typical challenges in neural network partition: **(A)** column-wise overlap; **(B)** row-wise overlap; **(C)** *psum* accumulation; **(D)** data reshaping. The symbol * means convolution.

$$t_{add\_1} = f_1(S_{row\_add}) + f_2(S_{col\_add}) + f_3(S_{p\_add}) + f_4(S_{re\_add}). \quad (6)$$

where S_{add} denotes the additional storage overhead in a core. $f_i(\cdot)$ represents the function for processing the additional data. Note that we have $f_i(0) = 0$. t_{add_1} is the additional computation latency of a core neglecting the additional latency caused by the physical mapping. Because the partition depth of input channels is equal to that of each weight filter, all of these partition methods are viewed as the synchronous partition in this work.

2.3. Physical mapping

The optimal physical mapping is acknowledged to be an NP-hard problem (Myung et al., 2021). The 2D mesh topology is widely adopted by neuromorphic hardware owing to its high throughput and scalability (Painkras et al., 2013; Akopyan et al., 2015; Davies et al., 2018; Pei et al., 2019; Shao et al., 2019; Deng et al., 2020; Zimmer et al., 2020). And the deadlock occurs in the 2D mesh usually. When the requested number of packets is more than that of the packet buffer size, the cores wait each other infinitely, thus deadlock occurs. To avoid deadlock

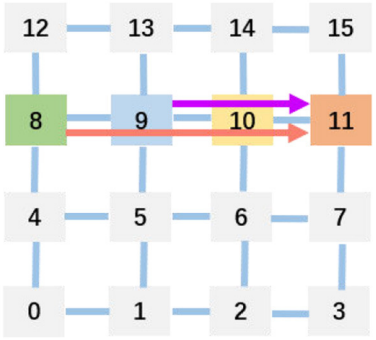


FIGURE 3
Example for helping understand communication latency caused by physical mapping.

and optimize the communication latency and energy in physical mapping, reinforcement learning (Ma et al., 2019; Barrett et al., 2020; Feng et al., 2020; Cappart et al., 2021; Mazyavkina et al., 2021) is used by some researchers (Wu et al., 2020; Myung et al., 2021). Moreover, Figure 3 explains the communication latency in a multi-core architecture after physical mapping (Amin et al., 2020). Because the 8-th and the 9-th cores send data to the 11-th core concurrently, the 10th core is crossed twice by them due to the physical route. Thus, the latency is generated by the 10th core.

There are many heuristic solutions to physical mapping, such as the genetic algorithm (GA) (Lei and Kumar, 2003; Zhou et al., 2006) and the simulated annealing (SA) (Ma et al., 2020) algorithm. Some teams (Davies et al., 2018; Shao et al., 2019; Zimmer et al., 2020) also use the greedy algorithm to optimize the communication latency and energy. We use t_{add_2} to denote the additional computation latency of a core when considering the physical mapping. It can be obtained by

$$\begin{aligned} t_{add\_2} = g_1(S_{row\_add}) + g_2(S_{col\_add}) \\ + g_3(S_{p\_add}) + g_4(S_{re\_add}). \end{aligned} \tag{7}$$

where $g_i(\cdot)$ represents the function for processing above additional data under the condition of physical mapping. Similarly, we have $g_i(0) = 0$.

3. Mapping limit

3.1. Logical mapping limit

For logical mapping, we introduce a theoretical description. We denote the sets of weights (W), input activations (IA), and output activations (OA) of the i-th and j-th core as W_i, W_j, IA_i, IA_j, OA_i, and OA_j, and further denote the storage volume of W and IA in each core as S_W and S_{IA}, respectively. Then, the logical mapping can be described as

$$\forall i,\ \cup_i IA_i = IA,\ \cup_i W_i = W, \tag{8}$$

$$\forall i,\ OA_i = IA_i * W_i,\ OA = IA * W, \tag{9}$$

$$\forall i \neq j,\ \cup_i OA_i = OA,\ OA_i \cap OA_j = \emptyset, \tag{10}$$

$$S_{IA} + S_W \leq S_{mem}, \tag{11}$$

where S_{mem} represents the total memory volume of a core. The non-overlap of OA indicates each output activation is calculated only once. Because OA will be transmitted to the IA memory of the cores for the next layer, S_{mem} does not take S_{OA} into account.

The additional storage overhead for a core generated in partition can be calculated by:

$$S_{col\_add} = \frac{H_{in}C_{in}}{Jm}(K_w - s) \cdot \mu(I - 2), \tag{12}$$

$$S_{row\_add} = \frac{W_{in}C_{in}}{Im}(K_h - s) \cdot \mu(J - 2), \tag{13}$$

$$S_{re\_add} = \frac{H_{out}W_{out}C_{out}}{IJm} \cdot \mu(n - 2), \tag{14}$$

$$S_{p\_add} = \frac{H_{out}W_{out}C_{out}}{IJm} \cdot \frac{b_p}{b} \cdot \mu(m - 2). \tag{15}$$

where $\mu(\cdot)$ represents the unit step function, s represents the stride of the filter, H_{in}, W_{in}, H_{out}, and W_{out} represent the height and width sizes of IA and OA, respectively. k_w and k_h are the width and height sizes of each weight kernel. And b_p and b represent the bit-width of *psum* and IA, respectively. The logical mapping limit means that the logical mapping does not produce any additional storage overhead, which can be described as

$$\forall i \neq j,\ IA_i \cap IA_j = \emptyset,\ W_i \cap W_j = \emptyset \tag{16}$$

$$S_{add} = 0. \tag{17}$$

When we approach the logical mapping limit, the values of S_{row_add}, S_{col_add}, S_{p_add}, S_{re_add}, t_{add_1}, and t_{add_2} should be zero.

3.2. Physical mapping limit

After the logical mapping stage, the logical cores would be mapped onto the physical cores in a real chip. With the aforementioned graph representation, we optimize the average communication latency (L) and power consumption (P) without deadlock. The physical mapping limit here implies all logical cores are physically placed very close, especially being neighbors with Manhattan distance equal to one, which can be described as

$$\forall v_i, v_j,\ M_{ij} = 1. \tag{18}$$

Then, the communication cost can be reduced to

$$C_{cost} = \sum\nolimits_{\forall v_i, v_j \in V} c_{ij} \times M_{ij} = \sum\nolimits_{\forall v_i, v_j \in V} c_{ij}. \tag{19}$$

Because W and IA must be put in cores, the minimum of C_{cost} is the sum of W and IA. Now, the communication cost can be given as follows:

$$\begin{aligned} C_{cost} &= \sum\nolimits_{\forall v_i, v_j \in V} c_{ij} \\ &= \sum\nolimits_{\forall v_i \in V} (IA_i + W_i) = IA + W. \end{aligned} \tag{20}$$

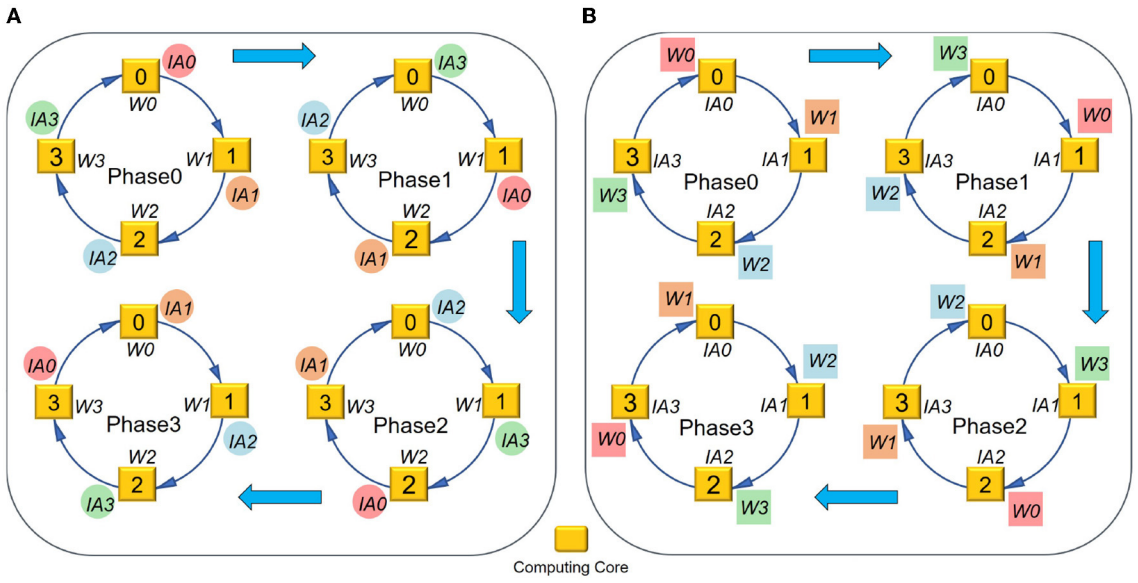


FIGURE 4
Closed-loop mapping based on **(A)** IA or **(B)** W.

The average communication latency and power consumption can be the communication latency and power consumption by transmitting a routing packet between two neighboring cores due to $M_{ij} = 1$.

In short, integrating the logical mapping limit and the physical mapping limit, the overall mapping limit follows

$$\forall i \neq j,\ IA_i \cap IA_j = \emptyset,\ W_i \cap W_j = \emptyset, \tag{21}$$

$$S_{add} = 0, \tag{22}$$

$$\forall v_i, v_j,\ M_{ij} = 1. \tag{23}$$

To approach the mapping limit, a closed-loop mapping strategy is proposed in the next section.

4. Approaches

4.1. Closed-loop mapping strategy

To approach the logical mapping limit, we propose a closed-loop mapping strategy with two forms. As illustrated in Figure 4, one form is based on IA, and the other is based on W. Taking four cores and the IA-based form as an example (see Figure 4A), the computing process can be described as follows. In the first phase, each core performs the convolution operation between IA_i and W_i. At the end of the first phase, each core keeps its W_i stationary and sends its IA_i to the downstream core. In the next phase, each core performs the convolution operation between W_i and its newly received IA. This loop would be closed when all cores have performed a complete convolution operation between its local W_i and all IAs. In this example, the loop needs four phases to close, and then we can get all OAs distributed in the four cores. The computing process can be summarized as

$$OA_{(i-t+N)\%N} = \sum_{t=0}^{N-1} IA_{(i-t+N)\%N} * W_i. \tag{24}$$

where N denotes the number of cores used for the layer and t is the index of phases. It can be seen that the above mapping strategy does not consume any additional memory overhead, satisfying the logical mapping limit given in Equations (21)–(22). For the W-based closed-loop mapping, the overall flow is similar. The only difference is that each core keeps IA stationary and exchanges W between cores.

In order to implement the closed-loop mapping on hardware, a 4D partition with synchronous and asynchronous methods is proposed for logical mapping, which is more flexible and general than the existing 2D synchronous partition. Here "4D" refers to C_{in}, C_{out}, and two dimensions of each feature map.

First, we try the 4D synchronous partition, as illustrated in Figure 5. Note that I, J, m, and n represent the number of partition groups in feature map height, feature map width, C_{in}, and C_{out} dimensions, respectively. k_w and k_h are the width and height sizes of each weight kernel. In the synchronous partition, W should be broadcasted along the feature map dimensions $I \times J$ times, and IA should be broadcasted along the output channel dimension n times. Therefore, the redundancy of storage caused by this partition is

$$S_{sync} = \frac{n(H_{in}W_{in}C_{in}) + IJ(k_w k_h C_{in} C_{out})}{H_{in}WinC_{in} + k_h k_w C_{in} C_{out}} - 1. \tag{25}$$

Moreover, the number of allocating cores is

$$N = IJmn. \tag{26}$$

The resulting storage overheads for IA and W in each core should be

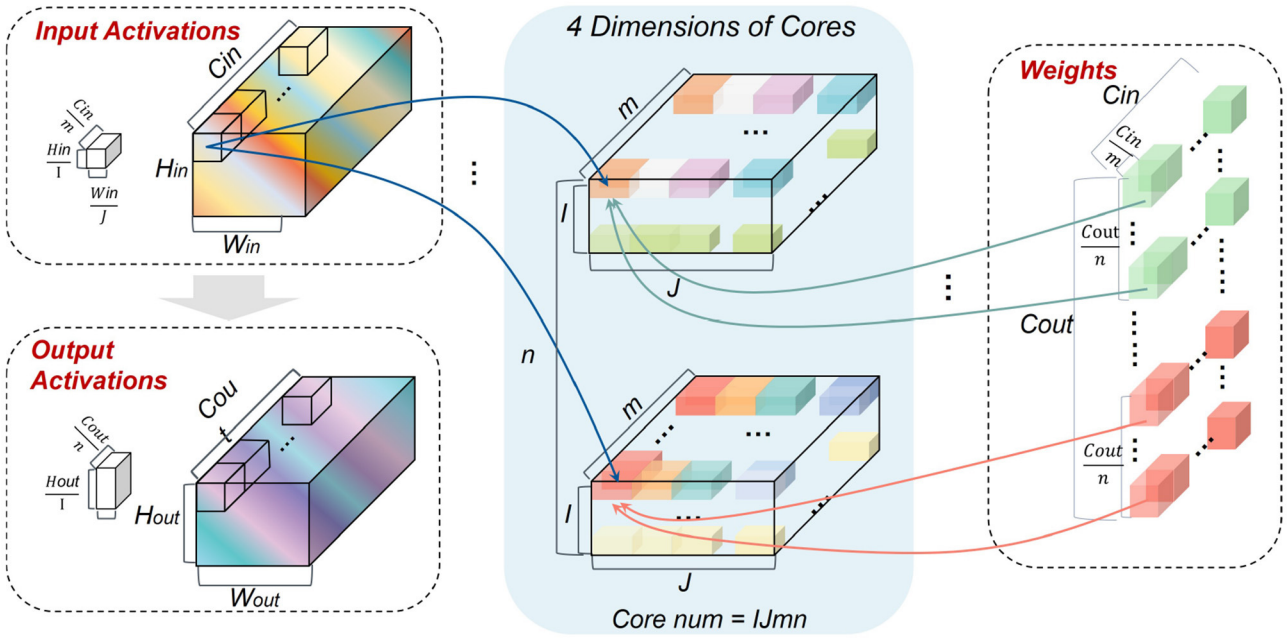


FIGURE 5
Illustration of the 4D synchronous mapping.

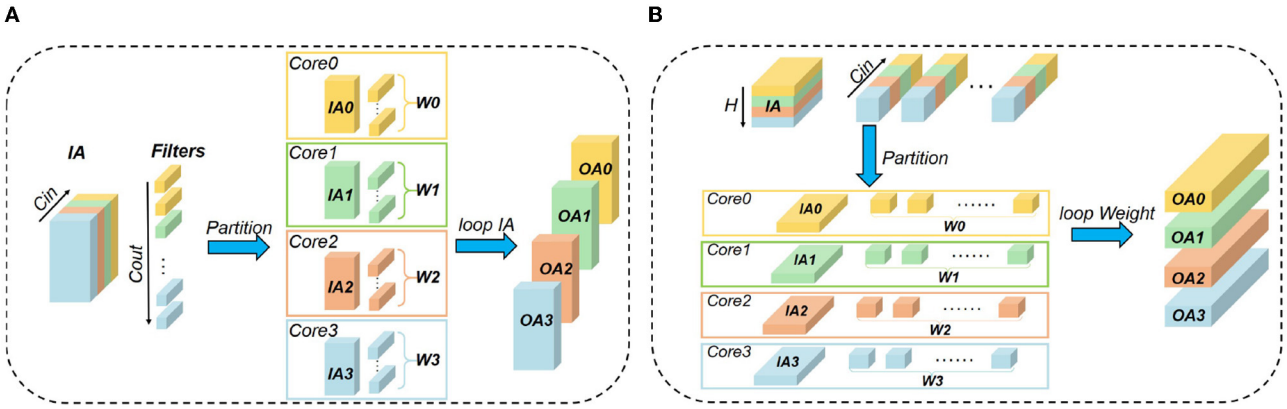


FIGURE 6
Illustration of the 4D asynchronous closed-loop mapping: **(A)** exchanging IA as Figure 4A; **(B)** exchanging W as Figure 4B.

$$S_{IA} = \frac{H_{in}W_{in}C_{in}}{IJm},\ S_W = \frac{k_w k_h C_{in} C_{out}}{mn}. \tag{27}$$

In short, the additional storage overhead on hardware given the 4D synchronous partition can be

$$S_{hw\_add} = (IJ-1)k_w k_h C_{in} C_{out} + (n-1)H_{in}W_{in}C_{in} + IJmnS_{add}. \tag{28}$$

Apparently, Equation (21) can only be satisfied under the condition $IJn = 1$, but Equation (22) cannot be satisfied under this case because *psums* exist. Therefore, the synchronous 4D partition fails to approach the logical mapping limit.

In order to approach the logical mapping limit described in Equations (21)–(22), we further propose an asynchronous partition method based on the closed-loop mapping strategy. Corresponding to the IA-based closed-loop mapping, the asynchronous partition method selects C_{in} of IA and C_{out} of W to partition. Taking $N = 4$ as an example, it can be seen from Figure 6A that both C_{in} of IA and C_{out} of W are partitioned into $m = n = N$ groups. Then, the resulting IA and W in each core can satisfy Equation (21) without duplication. The reshaping overhead does not exist because the shape of OA is consistent with that of IA. Because *psums* can be accumulated locally, the *psum* communication also does not exist. Therefore, all additional storage overheads are zero and Equation

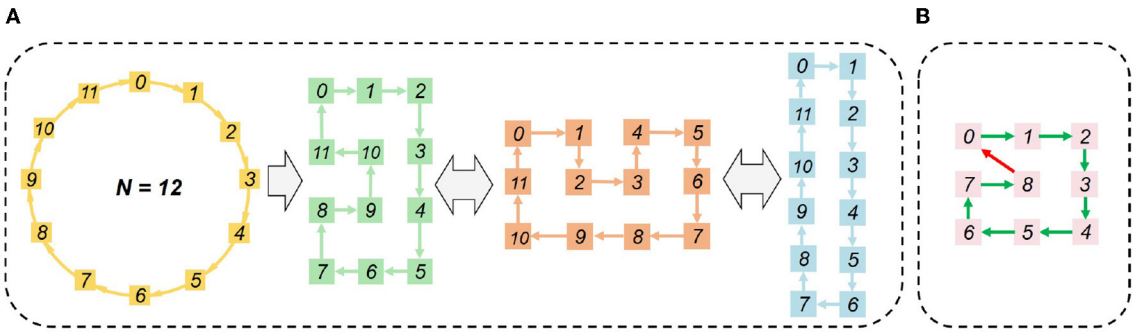


FIGURE 7
HLA-based physical mapping: **(A)** even number of cores; **(B)** odd number of cores.

(22) is satisfied. For the W-based closed-loop mapping in Figure 6B, the overall idea is similar to the IA-based case while H_{in} of IA and C_{in} of W are selected to partition. For the asynchronous closed-loop mapping, the storage overheads for IA and W in each core are

$$S_{IA} = \frac{H_{in}W_{in}C_{in}}{N},\ S_W = \frac{k_w k_h C_{in} C_{out}}{N}. \tag{29}$$

With the above knowledge, we make an explanation for the words "synchronous" and "asynchronous." In this work, "synchronous" means both the partitioning dimensions of IA and W involve C_{in}. In contrast, "asynchronous" means the partitioning dimensions of IA and W are different, for example in Figure 6A partitioning IA along the C_{in} dimension while partitioning W along the C_{out} dimension, and in Figure 6B partitioning IA along the H_{in} dimension while partitioning W along the C_{in} dimension. In the asynchronous closed-loop mapping, one of IA and W in each core has a complete C_{in} dimension, and the other is gradually acquired by exchanging data between cores without any redundant data copy.

4.2. Hamilton loop algorithm for physical mapping

To satisfy Equation (23) of the physical mapping limit, the Hamilton Loop Algorithm (HLA) is proposed for the closed-loop mapping strategy with asynchronous partition. Taking 12 cores as an example, it can be seen from Figure 7A that the Manhattan distance of every two logically neighboring cores equals 1, i.e., satisfying $M_{ij} = 1$ as given in Equation (23). The physical mapping form can be flexibly arranged according to the array form of the available physical cores, e.g., 4 × 3, 3 × 4, and 6 × 2. Notice that the number of cores cannot be odd, as illustrated in Figure 7B. In those cases, Equation (23) cannot be satisfied unless there is a diagonal communication path. Usually, only one Hamiltonian loop is needed. A fast algorithm is proposed to find a Hamiltonian loop, whose pseudo-codes are given in Algorithm 1.

```
for i in range(m):    //row
 //x direction communication distance
 dx[i][0] = 1 if  i == 0  else  0
 //y direction communication distance
 dy[i][0] = 0 if  i == 0  else  -1
 if n == 2:
 dx[i][n - 1] = -1 if i == m-1 else 0
 dy[i][n - 1] = -1 if i == m-1 else 0
 else:
   dx[i][n - 1] = 0 if i % 2 ==0 else -1
   dy[i][n - 1] = 1 if i % 2 ==0 else 0
 for j in range(1,n-1):    //column
    if j = =1 and i != m - 1:
   dx[i][j] = 1 if i % 2 ==0 else 0
   dy[i][j] = 0 if i % 2 ==0 else 1
 else:
  dx[i][j] = 1 if i % 2 ==0 else 1
 dy[i][j] = 0
```

Algorithm 1. Fast algorithm to find a Hamiltonian loop.

5. Experimental results

The mapping methods are implemented on a 28nm neuromorphic chip, TianjicX (Ma et al., 2022), which adopts a decentralized manycore architecture with 160 functional cores. Each core has 128 multipliers and accumulators (MACs) for parallel execution operations in neural networks. To maintain accuracy as high as possible, the precision for accumulating *psum*s is 32-bit. TianjicX supports the aforementioned 4D partition methods. The testing system includes a host computer, an Intel Arria 10 FPGA, four TianjicX chips, and an oscilloscope, as presented in Figure 8. The parameters and inputs of neural networks can be downloaded onto the chip by the configuration software on the host computer. The oscilloscope (RIGOL MSO8104) is used to measure the running time. Notice that the results of logical mapping are produced by the TianjicX simulator, while the results involving physical mapping are measured from the real chip.

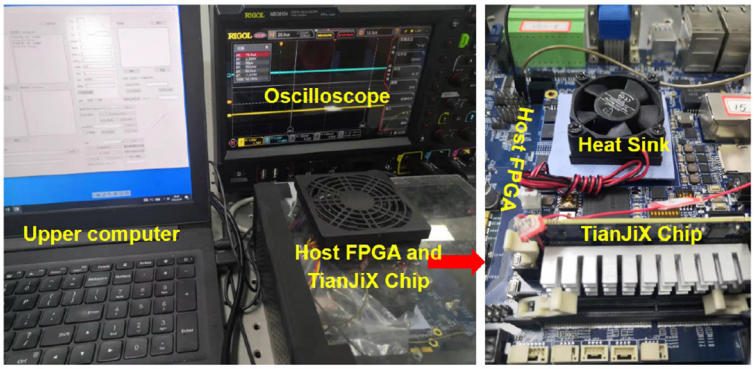


FIGURE 8
Testing system based on the TianjicX neuromorphic chip.

5.1. Analysis of logical mapping

We focus our application measurements on the ResNet50 (He et al., 2016), which is often used to benchmark by many hardwares (Jiao et al., 2020; Zimmer et al., 2020; Jouppi et al., 2021). However, as we do not have an automatic mapping tool at the current stage, we select a portion of the ResNet50 convolutional network for experimental analyses. In essence, the methodology can be extended to the whole convolutional networks in principle. To optimize the running time of each dimension, the synchronous partition method is selected as a baseline for investigation. The benchmarking layers are the 5-th and 6-th layers of ResNet50. The dimension settings of synchronous mapping are listed in Table 1. J, I, m and n represent the numbers of partition groups in the width, height, input channel, and output channel dimensions, respectively. First, from Model 1 to Model 6, $I \times J$ is set to a constant, 28, to explore the influence of partitioning I, J on the running clocks. Second, from Model 7 to Model 12, $J \times m = const$ and $I \times m = const$ are set to compare the influence priority of J, I, and m in dimension partition. Third, from Model 13 to Model 15, the influence priority is further compared among I, J, m, and n. Finally, we analyze the impact of changing partition dimensions on the running latency and computing efficiency.

TABLE 1 Dimension settings of synchronous partition.

No. of cores	J	I	m	n	Model
28	28	1	1	1	Model 1
28	14	2	1	1	Model 2
28	7	4	1	1	Model 3
28	4	7	1	1	Model 4
28	2	14	1	1	Model 5
28	1	28	1	1	Model 6
56	28	1	1	2	Model 7
56	14	1	2	2	Model 8
56	7	1	4	2	Model 9
56	1	28	1	2	Model 10
56	1	14	2	2	Model 11
56	1	7	4	2	Model 12
56	28	1	2	1	Model 13
28	14	1	1	2	Model 14
28	1	28	1	1	Model 15

The experimental results of partitioning different dimensions are provided in Figure 9. From Figure 9A, it can be seen that the close the values between J and I, the shorter running time can be achieved. Meanwhile, we observe that the latency results of Model 1–6 present small variance, which implies that the partition of feature map dimensions has a negligible impact on the execution latency. From Figure 9B, it can be seen that the running time would be increased when we partition C_{in}, which introduces additional accumulation of *psums* and extra inter-core communication. As Figure 9C shows, although Model 7 introduces reshaping latency as n increases, it still reduces the total running clocks by 8000 owing to the decrease of m. It indicates that the partition of C_{in} has a larger impact on the running time than the partition of C_{out}. Similarly, by comparing Model 14 and Model 15, we find the partition of C_{out} has larger impact than the partition of feature map dimensions. Overall, the accumulation and communication of *psums* caused by partitioning C_{in} has the greatest impact on the execution latency, while reshaping caused by partitioning C_{out} has a greater impact than overlapping caused by partitioning feature map dimensions. With the above knowledge, it is possible to optimize execution latency by elaborating partition method.

As aforementioned, the asynchronous partition based on the closed-loop mapping strategy can approach the mapping limit. To demonstrate its superior performance, we compare the running latency and computing efficiency of both synchronous partition and asynchronous partition. We use two types of layers: one is the 15-th layer of ResNet50 with 3×3 weight kernels, and the other is the 16-th layer of ResNet50 with 1×1 weight kernels. The model settings for the two benchmarking layers are respectively listed in Tables 2, 3.

The experimental results are depicted in Figures 9D, E. Due to the limited number of primitive instructions in TianjicX, the maximum number of nodes in a closed loop cannot be larger

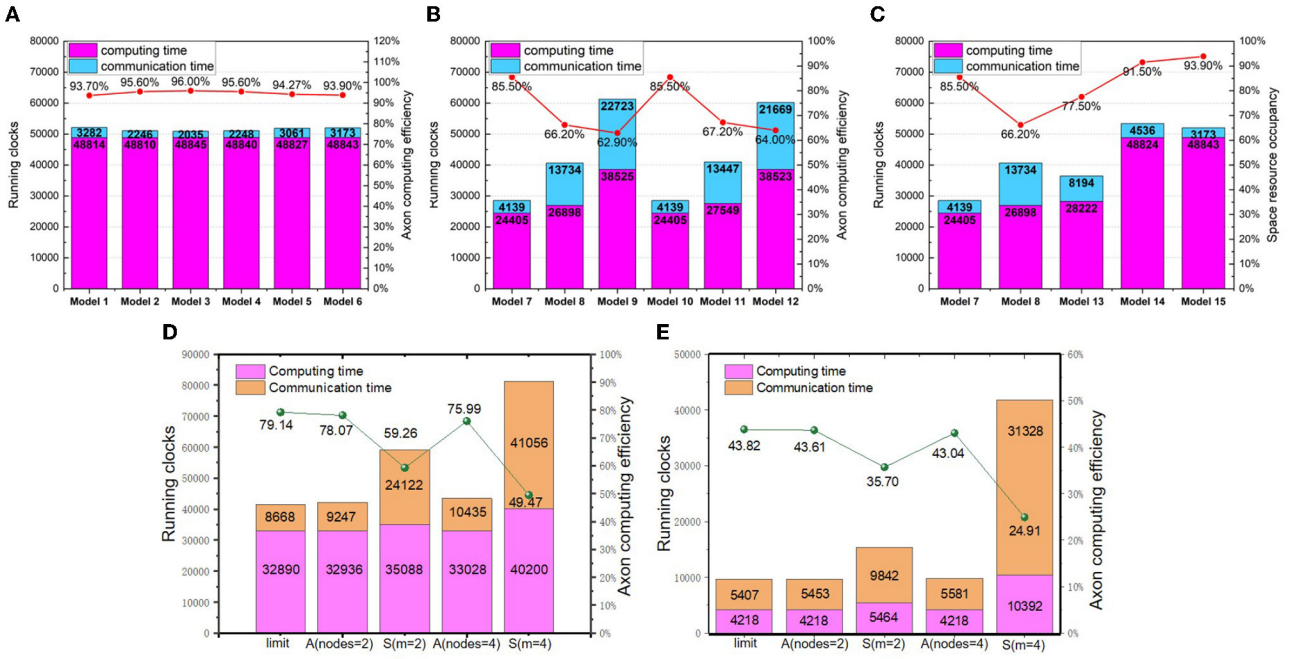


FIGURE 9
The logical mapping of running clocks and computing efficiency **(A)** synchronous partition of changing *I* and *J*; **(B)** synchronous partition of changing *m*, (*I*, *J*); **(C)** synchronous partition of changing *m*, *n*, and (*I*, *J*) **(D)** asynchronous partition with the 15-th layer of ResNet50; **(E)** asynchronous partition with the 16-th layer of ResNet50.

TABLE 2 Dimension setting of synchronous and asynchronous partition for the 15-th layer of ResNet50.

No. of cores	J	I	m	*n*	Model
28	7	1	1	4	limit
28	7	1	2	2	S($m = 2$)
28	7	2	1	4	A(#*nodes* = 2)
28	7	1	4	1	S($m = 4$)
28	7	4	1	4	A(#*nodes* = 4)

S, synchronous mapping; A, asynchronous mapping; No. of nodes, number of nodes in a closed loop.

TABLE 3 Dimension setting of synchronous and asynchronous partition for the 16-th layer of ResNet50.

No. of cores	J	I	m	*n*	Model
112	7	1	1	16	limit
112	7	1	2	8	S($m = 2$)
112	7	2	1	16	A(#*nodes* = 2)
112	7	1	4	4	S($m = 4$)
112	7	4	1	16	A(#*nodes* = 4)

S, synchronous mapping; A, asynchronous mapping; No. of nodes, number of nodes in a closed loop.

than four. As Figure 9E presents, the running latency under asynchronous partition based on the closed-loop mapping is faster than that of synchronous mapping. For example, the running latency of the 16-th layer can be improved by 4.12× under four nodes in a loop. Without the communication of *psums*, the communication latency of asynchronous mapping can also be greatly reduced.

5.2. Analysis of HLA physical mapping

To test the latency of HLA, we select all-to-all communication to conduct experiments. The 15-th layer of ResNet50 with 98KB parameters is the target workload. The all-to-all communication topology is illustrated in Figure 10A. The communicating latency is tested on TianjicX by enabling 4, 8, 16, or 32 cores. The energy consumption is estimated through simulation. Each case is tested with multiple physical mapping methods, including HLA, sequential neighboring placement with and without multicast (Myung et al., 2021), and several prior placement methods, including sequential placement (BS) (Wu et al., 2020), random search (RS) (Wu et al., 2020), simulated annealing (SA), and the RL-based approach (Wu et al., 2020). Due to the deadlock issue, we do not give the results of the zigzag physical mapping (Ma et al., 2020).

The communication latency results can be found in Figures 10B, C. As predicted, the communication latency of HLA is the shortest among all tested physical mapping methods, which is quite close to the mapping limit. The communication latency of HLA can be reduced by 4.22× compared to the neighboring placement with broadcast, and reduced by 84.1, 80.1, 74.1, and 67.9% compared to BS, RS, SA, and RL, respectively. Due to the launching delay of chip primitives, the communication latency increases as the number of used cores grows. The communicating

latency of HLA approaches that of the mapping limit if there is no launching overhead. When using the HLA physical mapping, all cores are parallel to communicate without deadlock.

Assuming the number of cores is N and the data of a core for communication is V, the total energy consumption of the mapping limit, HLA and the neighboring placement with broadcast and without broadcast can be calculated as follows:

$$\begin{aligned} E_{HLA} = E_{limit} &= N(VE|h|\sum_{i=0}^{N-1} 1) \\ &= VE|h|\frac{N(N-1)}{T}, \end{aligned} \tag{30}$$

$$\begin{aligned} E_{N\_B} &= VE|h|\sum_{i=0}^{N-1} i + N(VE|h|\sum_{i=0}^{N-1} 1) \\ &= VE|h|\frac{(N-1)(3N-2)}{2T}, \end{aligned} \tag{31}$$

$$\begin{aligned} E_{N\_W\_B} &= VE|h|\sum_{i=0}^{N}(\sum_{t=0}^{i} t + \sum_{j=0}^{N-i} j) \\ &= VE|h|\frac{2N^3 - 3N^2 + N}{6T}, \end{aligned} \tag{32}$$

where E_{HLA}, E_{limit}, E_{N_B}, and $E_{N\_W\_B}$ represent the energy consumption of under HLA, the mapping limit, the neighboring placement with broadcast, and the neighboring placement without broadcast, respectively. The energy results can be found in Figure 10D. The energy consumption of all methods increases as the number of the allocated cores grows. Obviously, the energy consumption under the neighboring placement is much higher than that under the mapping limit, while the energy consumption under HLA is equal to that under the mapping limit.

In short, the HLA physical mapping based on the closed-loop mapping strategy shows significant superiority on reducing communication latency and energy consumption compared with other methods. More importantly, the HLA physical mapping can approach the mapping limit.

5.3. Integration of logical and physical mapping

To demonstrate the performance of asynchronous logical mapping and HLA physical mapping based on the closed-loop mapping strategy, we deploy neural layers on TianjicX. The experimental results are provided in Figure 11. Again, Figures 11A, B evidence the superior latency of our closed-loop mapping strategy compared to the conventional synchronous mapping with C_{in} partition adopted by Simba (Shao et al., 2019; Zimmer et al., 2020), Tianjic (Pei et al., 2019; Deng et al., 2020), and other neural network accelerators (Han et al., 2016; Jouppi et al., 2017; Parashar et al., 2017; Shin et al., 2017; Chen et al., 2019). The

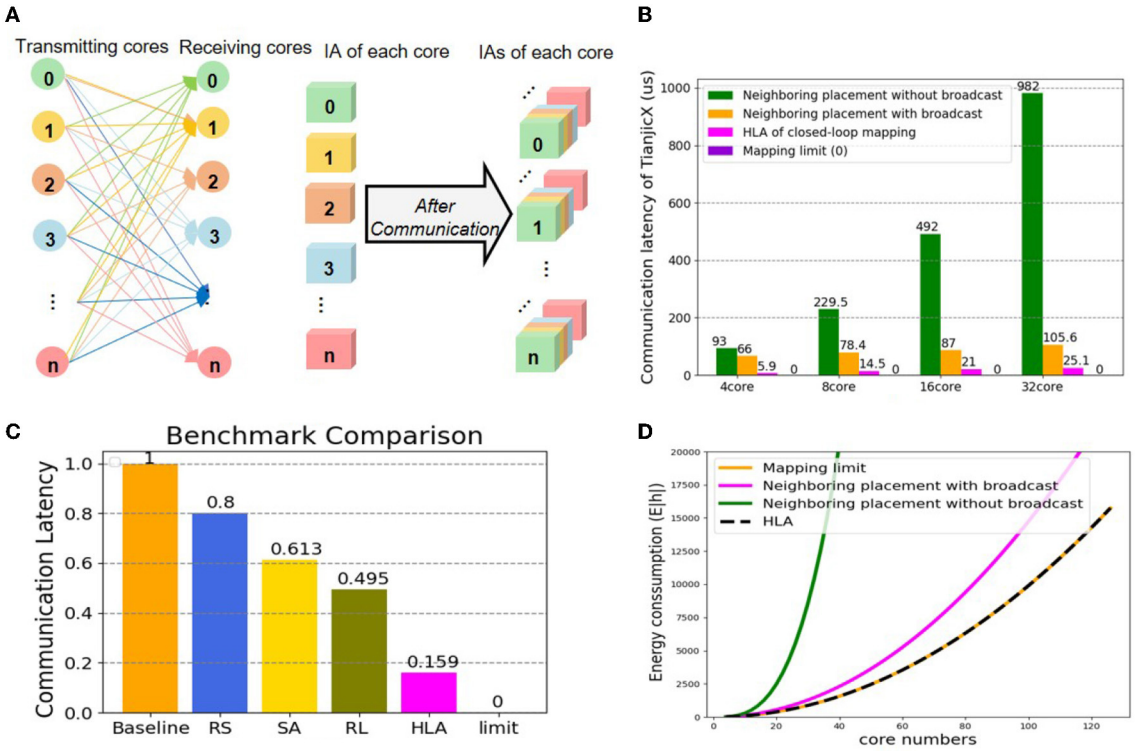


FIGURE 10
The physical mapping of latency and consumption **(A)** all-to-all communication between cores; **(B)** comparing communication latency between the neighboring placement, the mapping limit, and HLA; **(C)** comparing communication latency between prior methods, the mapping limit, and HLA; **(D)** comparing energy consumption between the neighboring placement, the mapping limit, and HLA.

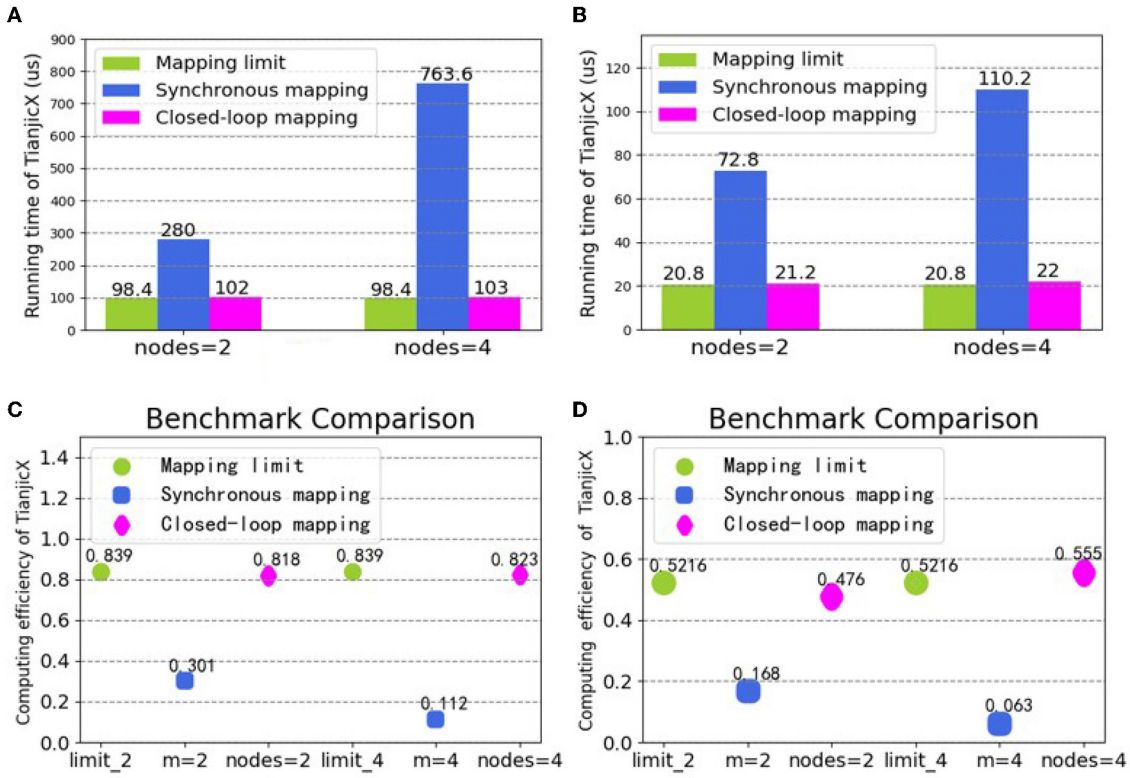


FIGURE 11
Running time and computing efficiency by integrating the logical and physical mapping under the closed-loop mapping strategy: **(A)** running time for the 15-th layer of ResNet50; **(B)** running time for the 16-th layer of ResNet50; **(C)** computing efficiency for the 15-th layer of ResNet50; **(D)** computing efficiency for the 16-th layer of ResNet50.

better computing efficiency of the closed-loop mapping strategy is also evidenced by Figures 11C, D. Specifically, with four nodes in a closed loop, the running time can be reduced by 7.6× for the 15-th layer of ResNet 50, and the computing efficiency can be improved by 8.8× for the 16-th layer. The proposed closed-loop mapping strategy implemented by integrating the asynchronous partition and the HLA placement can approach the mapping limit.

6. Conclusion and discussion

In this work, we propose the mapping limit concept for neuromorphic hardware based on the decentralized manycore architecture, which points out the resource saving upper limit during model deployment. To approach the mapping limit, we further propose the closed-loop mapping strategy that includes the asynchronous 4D partition for logical mapping and the HLA placement for physical mapping. Our experiments demonstrate the superiority of the proposed mapping methods. For example, compared to conventional synchronous C_{in} partition, our mapping methods improve the running time and computing efficiency by 7.6× and 8.8×, respectively, which can approach the mapping limit.

Generally, the mapping schemes for multi-core system can be divided into two processes: the first is the logical mapping process and the second is the physical mapping process. Furthermore, the logical mapping can be divided into two sets of models, which are synchronization and asynchronization. Most of the previous researches adopt the synchronization model based on the 2D mapping system (Shao et al., 2019; Ma et al., 2020; Wu et al., 2020; Myung et al., 2021), which only partitions the in-channel and out-channel of the neural network. And these researches focus on the physical mapping based on the 2D mapping system, while the 4D mapping system is a general model that has wider applications. Based on our 4D mapping system, we propose the mapping limit concept for the multi-core system. In the 4D mapping system, both the synchronization model and asynchronization model are demonstrated through intensive experiments. To achieve the mapping limit, we adopt the asynchronization mode to integrate the logical process and the physical process by the closed-loop mapping strategy.

Since the GPU is not a distributed architecture, the optimized result may be slightly rather than significantly improved in terms of energy consumption and computational speed. With the emergence of the decentralized architecture, the multi-core system is expected to be widely adopted due to its high-parallelism and memory locality (Painkras et al., 2013; Akopyan et al., 2015; Han et al., 2016; Parashar et al., 2017; Shin et al., 2017; Davies et al., 2018; Chen et al., 2019; Pei et al., 2019; Shao et al., 2019; Deng et al., 2020; Zimmer et al., 2020). Therefore, we are convinced that our proposed methods will provide a systematic solution to map neural networks onto multi-core systems, and provide guidance

for further development of auto-mapping tools. Moreover, with the proposed mapping limit and the closed-loop mapping strategy, it is possible to build a general and efficient mapping framework for multi-core system in the future.

Data availability statement

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

Author contributions

SW proposed the idea, designed and did the experiments, and wrote the manuscript. SW and QY conducted the algorithm modeling work, contributed to the analysis, and interpretation of results. SW, QY, and TX conducted the design and implementation of the hardware testing platform. CM led the discussion and revised it. JP directed the project and provided overall guidance. All authors contributed to the article and approved the submitted version.

Funding

This work was partially supported by Science and Technology Innovation 2030—New Generation of Artificial Intelligence, China Project (No. 2020AAA0109100).

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

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Amin, W., Hussain, F., Anjum, S., Khan, S., Baloch, N. K., Nain, Z., et al. (2020). Performance evaluation of application mapping approaches for network-on-chip designs. *IEEE Access* 8, 63607–63631. doi: 10.1109/ACCESS.2020.2982675

Barrett, T., Clements, W., Foerster, J., and Lvovsky, A. (2020). "Exploratory combinatorial optimization with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34* (New York, NY), 3243–3250. doi: 10.1609/aaai.v34i04.5723

Cappart, Q., Moisan, T., Rousseau, L.-M., Prémont-Schwarz, I., and Cire, A. A. (2021). "Combining reinforcement learning and constraint programming for combinatorial optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35* (Palo Alto, CA), 3677–3687. doi: 10.1609/aaai.v35i5.16484

Chen, Y.-H., Yang, T.-J., Emer, J., and Sze, V. (2019). Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Select. Top. Circuits Syst.* 9, 292–308. doi: 10.1109/JETCAS.2019.2910232

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., Liang, L., Wang, G., Chang, L., Hu, X., Ma, X., et al. (2018). Semimap: a semi-folded convolution mapping for speed-overhead balance on crossbars. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* 39, 117–130. doi: 10.1109/TCAD.2018.2883959

Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020). Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid-State Circuits* 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709

Feng, K., Wang, Q., Li, X., and Wen, C.-K. (2020). Deep reinforcement learning based intelligent reflecting surface optimization for miso communication systems. *IEEE Wireless Commun. Lett.* 9, 745–749. doi: 10.1109/LWC.2020.2969167

Gholami, A., Yao, Z., Kim, S., and Mahoney, M. W. (2021). *AI and Memory Wall[j]*. RiseLab Medium Post.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., et al. (2016). EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Comput. Architect. News* 44, 243–254. doi: 10.1145/3007787.3001163

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[0mm][8mm] and Pattern Recognition* (Las Vegas), 770–778. doi: 10.1109/CVPR.2016.90

Jiao, Y., Han, L., Jin, R., Su, Y.-J., Ho, C., Yin, L., et al. (2020). "7.2 a 12nm programmable convolution-efficient neural-processing-unit chip achieving 825tops," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Francisco, CA), 136–140. doi: 10.1109/ISSCC19947.2020.9062984

Jouppi, N. P., Yoon, D. H., Ashcraft, M., Gottscho, M., Jablin, T. B., Kurian, G., et al. (2021). "Ten lessons from three generations shaped Google's tpuv4i: industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (Valencia), 1–14. doi: 10.1109/ISCA52012.2021.00010

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON), 1–12. doi: 10.1145/3140659.3080246

Lei, T., and Kumar, S. (2003). "A two-step genetic algorithm for mapping task graphs to a network on chip architecture," in *Euromicro Symposium on Digital System Design, 2003* (Belek-Antalya), 180–187.

Ma, C., Zhao, Q., Li, G., Deng, L., and Wang, G. (2020). A deadlock-free physical mapping method on the many-core neural network chip. *Neurocomputing* 401, 327–337. doi: 10.1016/j.neucom.2020.03.078

Ma, Q., Ge, S., He, D., Thaker, D., and Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*.

Ma, S., Pei, J., Zhang, W., Wang, G., Feng, D., Yu, F., et al. (2022). Neuromorphic computing chip with spatiotemporal elasticity for multi-intelligent-tasking robots. *Sci. Robot.* 7:eabk2948. doi: 10.1126/scirobotics.abk2948

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: a survey. *Comput. Oper. Res.* 134:105400. doi: 10.1016/j.cor.2021.105400

Myung, W., Lee, D., Song, C., Wang, G., and Ma, C. (2021). Policy gradient-based core placement optimization for multichip many-core systems. *IEEE Trans. Neural Netw. Learn. Syst.* 1–15. doi: 10.1109/TNNLS.2021.3117878

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

Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., et al. (2017). SCNN: an accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Comput. Architect. News* 45, 27–40. doi: 10.1145/3140659.3080254

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., et al. (2019). "SIMBA: scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (New Jersey, NJ), 14–27. doi: 10.1145/3352460.3358302

Shin, D., Lee, J., Lee, J., and Yoo, H.-J. (2017). "DNPU: an 8.1 tops/w reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA), 240–241. doi: 10.1109/ISSCC.2017.7870350

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Advances in Neural Information Processing Systems, 30* (Long Beach, CA).

Von Neumann, J. (1993). First draft of a report on the EDVAC. *IEEE Ann. Hist. Comput.* 15, 27–75. doi: 10.1109/85.238389

Wu, N., Deng, L., Li, G., and Xie, Y. (2020). Core placement optimization for multi-chip many-core neural network systems with reinforcement learning. *ACM Trans. Design Autom. Electron. Syst.* 26, 1–27. doi: 10.1145/3418498

Zhou, W., Zhang, Y., and Mao, Z. (2006). "An application specific NOC mapping for optimized delay," in *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006* (Tunis), 184–188.

Zimmer, B., Venkatesan, R., Shao, Y. S., Clemons, J., Fojtik, M., Jiang, N., et al. (2020). A 0.32-128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE J. Solid State Circuits* 55, 920–932. doi: 10.1109/JSSC.2019.2960488

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 14 June 2023
DOI 10.3389/fnins.2023.1167134

OPEN ACCESS

EDITED BY
Huajin Tang,
Zhejiang University, China

REVIEWED BY
Mikhail A. Mishchenko,
Lobachevsky State University of Nizhny Novgorod, Russia
Shuangming Yang,
Tianjin University, China

*CORRESPONDENCE
Hong Qu
✉ hongqu@uestc.edu.cn

RECEIVED 16 February 2023
ACCEPTED 26 May 2023
PUBLISHED 14 June 2023

CITATION
Chen Y, Liu H, Shi K, Zhang M and Qu H (2023) Spiking neural network with working memory can integrate and rectify spatiotemporal features. *Front. Neurosci.* 17:1167134. doi: 10.3389/fnins.2023.1167134

Spiking neural network with working memory can integrate and rectify spatiotemporal features

Yi Chen, Hanwen Liu, Kexin Shi, Malu Zhang and Hong Qu*

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

In the real world, information is often correlated with each other in the time domain. Whether it can effectively make a decision according to the global information is the key indicator of information processing ability. Due to the discrete characteristics of spike trains and unique temporal dynamics, spiking neural networks (SNNs) show great potential in applications in ultra-low-power platforms and various temporal-related real-life tasks. However, the current SNNs can only focus on the information a short time before the current moment, its sensitivity in the time domain is limited. This problem affects the processing ability of SNN in different kinds of data, including static data and time-variant data, and reduces the application scenarios and scalability of SNN. In this work, we analyze the impact of such information loss and then integrate SNN with working memory inspired by recent neuroscience research. Specifically, we propose Spiking Neural Networks with Working Memory (SNNWM) to handle input spike trains segment by segment. On the one hand, this model can effectively increase SNN's ability to obtain global information. On the other hand, it can effectively reduce the information redundancy between adjacent time steps. Then, we provide simple methods to implement the proposed network architecture from the perspectives of biological plausibility and neuromorphic hardware friendly. Finally, we test the proposed method on static and sequential data sets, and the experimental results show that the proposed model can better process the whole spike train, and achieve state-of-the-art results in short time steps. This work investigates the contribution of introducing biologically inspired mechanisms, e.g., working memory, and multiple delayed synapses to SNNs, and provides a new perspective to design future SNNs.

KEYWORDS

spiking neural network, working memory, convolutional neural network, CIFAR10, multi-dendrite

1. Introduction

Artificial Neural Networks (ANNs) learned from biological neural networks achieved huge success in these years. Spiking Neural Networks (SNNs) as a step forward to biological neural networks caught up with their ANNs counterparts and even outperform in computer vision (Meng et al., 2022; Zhu et al., 2022), sound recognition (Pan et al., 2020, 2021), and so on (Lobo et al., 2020; Li et al., 2022). Theoretically, SNNs, which are more similar to biological neural networks, should have more advantages in dealing with real tasks. On the

contrary, there is still a certain gap between SNN and ANN in terms of the scope of application and performance in general. The reason is that ANN's synchronicity in processing simulated information allows it to fully consider every detail. In contrast, SNN's asynchronous processing of discrete information makes it better in power consumption performance, but it cannot comprehensively consider the complete information. In fact, SNNs' advantages in complex temporal dependence have not been fully discovered. A classic spiking neuron can only accumulate the most recent spike train it has received and fails to integrate comprehensive spatiotemporal features, as shown in Figure 1. For the static image with rate-based coding, the one-way aggregation of SNN itself makes it unable to judge based on valid information. The effect of this problem is even more pronounced with latency coding, where neurons see only a small part of the picture. The same is true for dynamic sequential data. For example, if a video of a long jump contains two consecutive actions, namely a run-up, and a jump, the traditional SNN structure may make a judgment during the run-up and ignore the subsequent jump.

Previously, researchers have tried to increase the temporal receiver domain of SNNs in various ways to improve the ability of SNNs to process spatiotemporal data. According to the different ideas of their methods, they can be divided into two categories: training more parameters of neurons and changing the structure of neural networks.

In addition to the weights that can be trained as in ANN, spiking neurons have many different parameters that determine the dynamic characteristics of neurons. In Luo et al. (2022), by training synaptic delay, the neuron obtains the ability to rearrange and integrate the spike train, which further increases the ability of SNN to process timing data based on the original learning algorithm. In Fang et al. (2021b), by training the time constant of the neuron, the spike response function of the neuron is changed, and the spiking neuron can obtain the time receiving field length and attenuation coefficient more suitable for the current task through learning. In Rathi and Roy (2023), the threshold is changed into a trainable parameter. However, the above algorithm is only optimized at the neuron level, and a single neuron can still only obtain limited information and cannot obtain a global perspective.

In order to expand spiking neurons' temporal receptive fields, researchers have made various works according to specific tasks by referring to various ANN structures that already exist. In Zhang and Li (2021), the author added circular connections to SNN, but the weight of phantom connections was manually set. Zhang and Li (2019), the author also changed the network into a loop structure and proposed an effective training method. El-Assal et al. (2022), the author takes the 3D convolution kernel as the input weight. However, this method will extract a large amount of redundant information between adjacent time periods, so compared with the 2D convolution method commonly used in SNN, the improvement is limited. In Yao et al. (2021), the author introduced the attention structure into SNN and proposed the SNN network based on temporal attention. This kind of network is a hybrid network of ANN and SNN. In the forward inference stage, in addition to the original SNN operation, the network also needs to pass a fully connected network with a multi-layer sigmoid function as the activation function, which increases the computation amount.

In the biological brain, cortical neurons process information on multiple timescales, and areas important for working memory contain neurons capable of integrating information over a long timescale (Kim and Sejnowski, 2021). In terms of vision, working memory is already involved in the early part of the whole visual pathway. The distinct visual stimuli (oriented gratings and moving dots) are flexibly recorded into the same working memory format in visual and parietal cortices when that representation is useful for memory-guided behavior (Kwak and Curtis, 2022). Therefore, working memory is very important for the extraction of temporal information. Introducing working memory into SNN can effectively improve the processing ability of SNN on spatiotemporal data. Although the specific structure of working memory in the brain has not been determined, we can still combine the properties of SNN to propose a working memory block suitable for SNN.

Based on this, we integrate multiple delayed synapses in spiking neural networks and propose a simple but effective structure Spiking Neural Network with Working Memory (SNNWM). Compared with traditional SNN, SNNWM adds multiple groups of dendrites with different delays. These dendrites effectively increase SNN's receptive field in the time domain, enabling SNN to gradually acquire global vision with the deepening of network layers. After that, we provide a simple method to implement the proposed network architecture in both software and hardware. Among them, we analyze the differences between SNNWM and traditional SNN in hardware implementation and draw the conclusion that SNNWM can increase a small number of storage resource access operations without increasing the extra consumption in computation. Finally, we test our method on two different data: static image, and dynamic event sequence. Experimental results show that working memory increases SNN's power dealing with spike trains and reaches the state of the art with low latency.

In summary, our main contributions are as follows:

1) We propose the spiking neural network with working memory by introducing multiple delayed synapses and offer a simple method to reduce the number of parameter increases.
2) For the model proposed in this paper, we give the implementation methods of software and hardware and further demonstrate that the proposed model will not generate excessive resource consumption when implemented by hardware.
3) We further validated the effectiveness of adding working memory to SNN by testing the effectiveness of the proposed model on static and dynamic data, respectively.

2. Materials and methods

In this section, we first introduce the spiking neuron model and later propose our spiking neural network with working memory based on this neuron model. After that, we propose a practical implementation of SNNWM for neuromorphic hardware as well as FPGAs (Field Programmable Gate Arrays). Subsequently, to further enhance the temporal aggregation ability of the model and simplify the computational burden, we propose a temporal fusion layer. Finally, we introduce the training algorithm used in this paper.

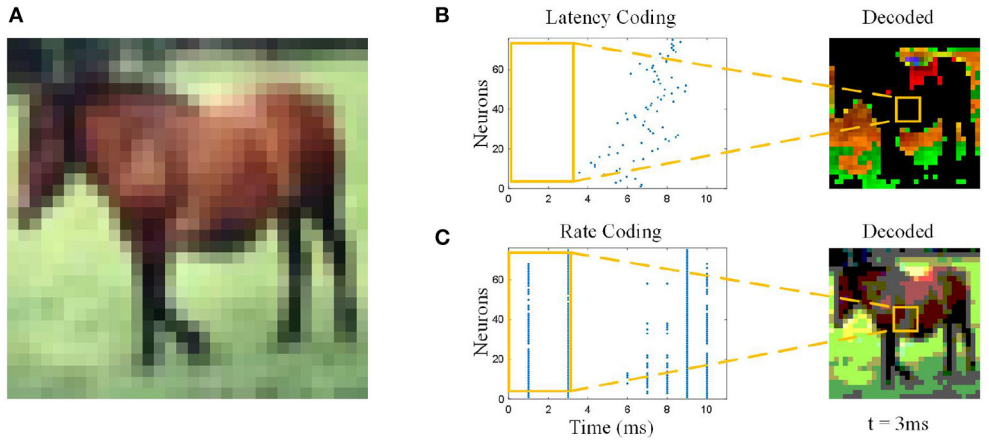


FIGURE 1
Information loss with different coding methods. **(A)** Original image. The pixel value of the image will be directly input into ANNs, so the ANNs see the full image. **(B)** Latency coding. At 3 ms, the neurons "see" only part of the background grass in the picture, not the horse, and the color was distorted. **(C)** Rate coding. At 3 ms, the neurons "see" the shape of the horse, but the color was distorted.

2.1. Spiking neuron models

The Leaky Integrate and Fire (LIF) model and Integrate and Fire (I&F) model are the two most commonly used spiking neuron model at present, which is more optimized for neuromorphic hardware design due to their lower complexity and iterative representation. In general, the I&F model can be regarded as the LIF model with the leaky term set to 1. Therefore, for simplicity and generality, we adopt the discrete representation LIF model in Wu et al. (2018). In LIF model, the membrane potential V_j of neuron j at time t is updated as follow:

$$V_j(t) = e^{-\frac{1}{\tau}} V_j(t-1) + \sum_{i=1}^{N_i} w_{ij} K(s_i(t)), \tag{1}$$

where w_{ij} is the synaptic weight between neuron i and j, $K(s)$ is the spike response function and here we chose $K(s) = s$ for simplicity, $s_i(t) \in \{0, 1\}$ is the spike train from presynaptic neuron i and $s(t) = 1$ means neuron i fires a spike at time t. When the membrane potential exceeds the threshold θ from below, the spiking neuron j fires a spike $s_j(t)$ at this time t, and its membrane potential is set to resting potential V_{rest}. This procedure can be described as:

$$s_j(t) = H(V_j(t) - \theta), \tag{2}$$

$$V_j(t) = V_j(t)(1 - s_j(t)) + V_{rest} s_j(t), \tag{3}$$

where H is the Heaviside step function:

$$H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & else \end{cases}. \tag{4}$$

It can be found by Equation (1) that SNN can process simple time-series information naturally, and its temporal reception field is closely related to constant τ, namely leaky term. Specifically, the previous spikes can affect the membrane potential at that moment, dotted arrows from layer l to layer $l + 1$ in Figure 2A, and then indirectly affect the membrane potential at the current moment by leaky term, the solid arrows on the bottom in Figure 2A. Moreover, this indirect effect may be eliminated by firing a spike that causes the membrane potential to reset. Spiking neurons themselves have limited temporal information processing ability, so it is necessary to make some changes in the network structure to improve the ability of SNN.

2.2. Spiking neural network with working memory

Working memory is defined as a processing resource of limited capacity, involved in the preservation of information while simultaneously processing the same or other information. There are a variety of theories about the formation mechanism and storage structure of working memory. Still, here we only focus on its main function, which is to store a piece of related information for other modules to use. Here, we expect the spiking neuron to change its membrane potential mainly based on the spikes over a period of time, rather than on the spikes at the present moment. This means that we need to modify the spike train over a period of time so that it can reach the neuron at the same time.

In the biological brain, there are multiple synapses between neurons, and these diverse synapses increase the brain's ability to process complex spatial-temporal signals. In ANNs, there is only one synaptic connection between two neurons to simplify the model and facilitate calculation. Even if there are multiple synapses, because of the way ANN works synchronously, multiple synapses can be equivalent to one synaptic connection. On the contrary, in SNN, the spiking neurons' temporal dynamics enable multiple delayed synapses effectively increasing the ability of SNN to process complex spatial-temporal data.

Inspired by multiple delayed synapses in Bohte et al. (2000), we integrate multiple groups of dendrites with different delays based on the original LIF model and proposed a multi-dendrite spiking

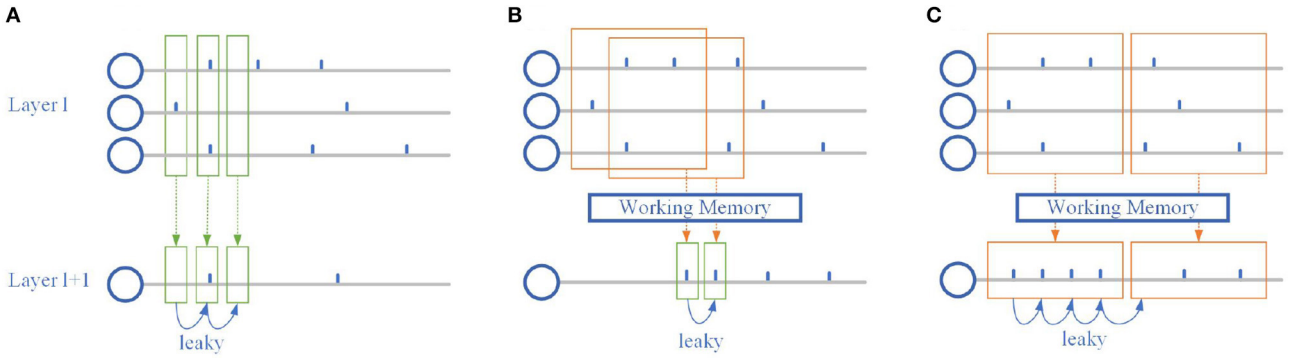


FIGURE 2
Temporal dependence of the spiking neuron. **(A)** The classic spiking neuron model. The output spike train is directly affected by the current input and indirectly by the previous input. **(B)** Spiking neuron with overlapping working memory. The output spike train is directly affected by both the current and the previous input. **(C)** Spiking neuron with individual working memory. The output spike train in the same working memory window is rectified.

neural network with delay. Equation (1) becomes:

$$V_j(t) = e^{-\frac{1}{\tau}} V_j(t-1) + \sum_{k=1}^{N_k} \sum_{i=1}^{N_i} w_{ijk} s_i(t-d_k), \tag{5}$$

where w_{ijk} and d_k is the weight and transmission delay of dendrite group k, respectively. Multiple groups of dendrites with different delays effectively increase the time domain exploration of the neuron's input spikes at the same synapse. As shown in Figures 2B, C, Multiple dendrites help spiking neurons explore many different combinations of spike trains simultaneously, without having to wait for all inputs to proceed to the next layer of computation as in ANN-SNN hybrid networks that introduce the attention.

This multi-dendrite structure would increase the number of SNNs' parameters and further reduce usage in resource-restricted edging platforms. Inspired by spatial factorization in Inception-V2 (Szegedy et al., 2016), we split the weight matrix w_{ijk} of size $N_i \times N_j \times N_k$ in Equation (5) into two matrices of size $N_i \times N_j \times 1$ and $1 \times N_k$, respectively. In this way, the number of network parameters is reduced from the original $N_i \times N_j \times N_k$ to $N_i \times N_j + N_k$ and only increased by N_k compared with classical SNN. In this way, Equation (5) becomes:

$$V_j(t) = e^{-\frac{1}{\tau}} V_j(t-1) + \sum_{k=1}^{N_k} w_k \sum_{i=1}^{N_i} w_{ij} s_i(t-d_k), \tag{6}$$

To further enhance the global processing capability of SNN, we introduce an additional memory mechanism to rectify spike train through working memory to increase synchronization, as shown in Figure 2C. The rectified output spike train can effectively increase the stability of the neural network and improve the ability to handle static data, and Equation (6) becomes:

$$V_j(t) = e^{-\frac{1}{\tau}} V_j(t-1) + \sum_{k=1}^{N_k} w_k \sum_{i=1}^{N_i} w_{ij} s_i(t-d_k-m), \tag{7}$$

$$m = \lfloor t/mem\_len \rfloor, \tag{8}$$

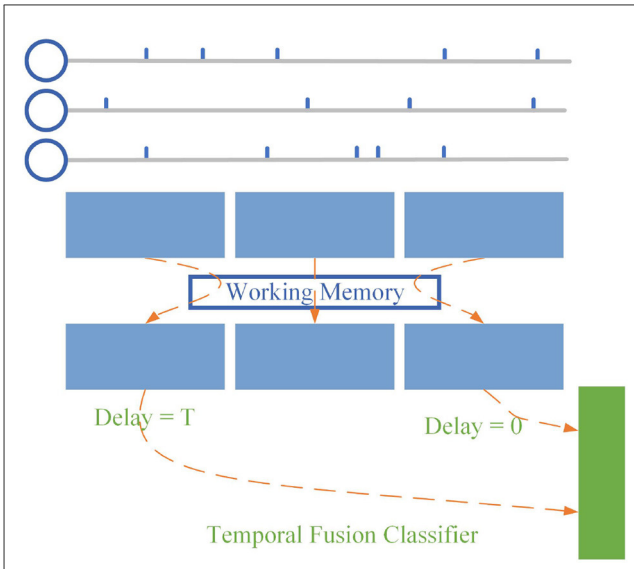


FIGURE 3
Temporal fusion layer. Instead of using the dynamic processes inside the SNN, the temporal fusion layer combines pulses at all times to give the SNN a global view.

where *mem_len* is the length of working memory. The working memory that operates in this way takes into account all the spikes within *mem_len* and continuously feeds them into the spiking neuron for *mem_len*. This method can effectively alleviate the information loss caused by uneven spike distribution in the time domain for data that do not need fast time-varying information.

2.3. Temporal fusion layer

At present, most SNNs take the spike frequency or average membrane potential of the last layer as output when processing tasks. We designed the final layer based on the proposed SNNWM. For spike sequences with fixed input length T, we set the dendrite groups' transmission delay of neurons at the last layer as

$\{0, 1, 2..., T\}$, as shown in Figure 3 and only the mode potential of the Tth time step is used as the output. According to Equation (4), the decoding scheme can be expressed as:

$$o_j = \sum_{k=1}^{T} w_k \sum_{i=1}^{N_i} w_{ij} s_i (T - d_k). \tag{9}$$

In this way, on the one hand, the neurons in the last layer obtain spike information at all times at the time step T, thus enhancing the performance of SNN; on the other hand, the computation at the previous $T - 1$ time step is reduced, effectively reducing the computation cost.

2.4. Implementation in software and hardware

Our goal is to make the most of the SNN's low-power, high-dynamic processing capabilities, without having to be the same as biological neurons. Therefore, in the process of practical application, the $\{0, 1\}$ sequence is often used to encode the spike sequence. A 0 or 1 in each bit represents whether there is a spike event, and each position represents a small period. For example, a spike train 010110 can represent a spike train with a simulation duration of 6 ms and each period of 1 ms.

To further simplify the model, we use an arithmetic sequence as the delay in SNNWM. That is, the synaptic delay between the two neurons is $\{0, 1, 2, 3...L\}$, where L represents the length of working memory. And the synaptic delay between the two adjacent layers of neurons was the same. In this way, SNNWM with overlapping working memories can be achieved by a simple one-dimensional convolution operation with an additional convolution kernel $W1$, or w_k in Equation (6), in the time dimension.

The implementation method of SNNWM without overlap is shown in the pseudo-code of Algorithm 1. Firstly, the input spike train S_{in} is divided into N segments of length L in the time dimension. These segments can be processed in parallel with each other before calculating changes in membrane potential V. The change in membrane potential I_{seg} from the input is obtained by multiplying the spike segments with the synaptic weights of sizes $N_i \times N_j$ and N_k, respectively. The change in membrane potential from the input is obtained by multiplying the spike segment with the synaptic weight $W1$ of size $N_i \times N_j \times 1$ and the synaptic weight $W2$ of size $1 \times N_k$.

The method of SNNWM's hardware implementation, specifically FPGA, is shown in Figure 4. The traditional SNN will input the spike train at each moment (the sequence in the green box) into the LIF unit, and extract the weight matrix and the membrane potential at the last moment from BRAM. After completing the calculation of the membrane potential at the current moment, the obtained spike train at the current moment will be output, and then the new neuron membrane potential will be stored in V_BRAM.

The total input spike data size remains the same, and the output spike train data size remains the same. It is worth noting that since the introduction of working memory will simultaneously participate in the calculation of spikes within each working memory window size, the fetch operation of V_BRAM only occurs at the beginning of this window time, and the save operation only occurs at the end of this window time. Compared with the previous SNN, the number of V_BRAM accesses is saved.

Input: Input spike train: S_{in}, Synaptic weights: $W1, W2$, Working memory length: L, Constant: τ, Threshold: V_{th}, Resting potential: V_{rest}
Output: Output spike train: S_{out}

Initialization: $S_{out} \leftarrow []$
Split S_{in} into N segments of length L;
$S_{seg} = split(S_{in})$
foreach $n \leftarrow 0$ *to* N **do**
  $I_{seg}[n] = S_{seg}[n] \times W1 \times W2$
end
for $n \leftarrow 0$ *to* N **do**
  for $t \leftarrow 0$ *to* L **do**
    if *n is 0 AND t is 0* **then**
      $V = I_{seg}[n]$
    else
      $V = V \cdot e^{-\frac{1}{\tau}} + I_{seg}[n]$
    end
    if $V \geq V_{th}$ **then**
      $V = V_{rest}$;
      $spike = 1$;
    else
      $V = V$;
      $spike = 0$;
    end
    Concat *spike* into output spike train S_{out};
    $S_{out} = concat(S_{out}, spike)$
  end
end

Algorithm 1. Pseudo code of SNNWM.

2.5. Training

For the optimization of network parameters, we choose the STBP (Spatio-Temporal BackPropagation) (Wu et al., 2018) in the surrogate gradient learning method. At present, there are many opinions about the selection of approximate functions. Wu et al. (2018) believes that the parameters of the function are more important than the selection of the function, while Fang et al. (2021a) believes that the approximate function with different shapes can bring better results. In this work, we chose the surrogate function:

$$\frac{\partial H(x)}{\partial x} = e^{-2x^2} \tag{10}$$

for gradient learning. Compared with the triangular or rectangular surrogate function, the exponential surrogate function can ensure that some gradient information can be transmitted even when the membrane potential is far from the threshold, rather than no gradient at all. At the same time, we found in the experiment

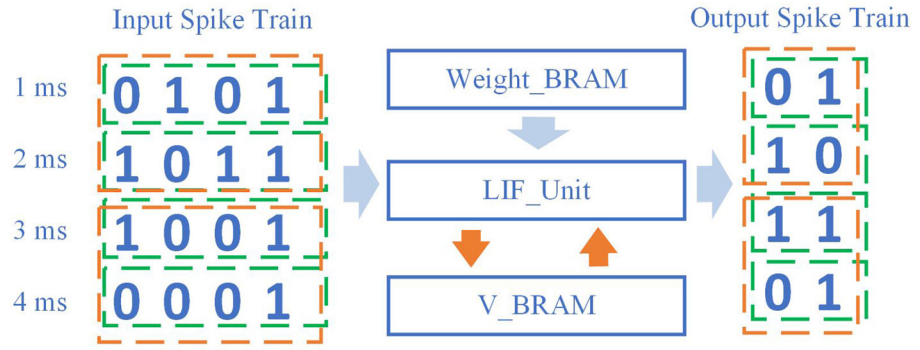


FIGURE 4
Implementation of working memory in hardware. Compared with classical methods, this method does not consume too much in global data transmission and processing. The computation cost by the input spike train and the change of neuron state remains unchanged.

that using hard Tan as an approximation function would slightly increase the training time and have no effect on the final result.

3. Results

To verify the ability of our proposed model to extract temporal and spatial features, we designed a variety of different experiments on two different data types: images and event streams. We compared the proposed method with other different methods with the same or similar network structure and scale. These networks include conventional network structures or are optimized according to the characteristics of SNNs, introducing particular neurons or layer structures. The training methods adopted by them include transform-based methods and BP-based methods. The details of the implemented methodology and the experimental setup are presented below. After that, we carried out experiments for different types of data sets and comprehensively tested the influence of preprocessing methods and network working memory size on the model's performance.

3.1. Implementation details

In the following experiments, we implemented the proposed model on two NVIDIA RTX 3090s using the Pytorch training framework. For CIFAR10 (Krizhevsky et al., 2017) and CIFAR100 (Krizhevsky et al., 2017), we utilize the SGD optimizer with the momentum of $1e^{-4}$ to accelerate the training process, and for DVS128 Gesture (Amir et al., 2017) and CIFAR10-DVS (Li et al., 2017), we utilize the Adam optimizer. The hyperparameters used for training are shown in Table 1 for different data sets. Compared with some other SNN-related works, we adopt the conventional training parameter setting here, and all experimental results are the average values obtained after 5 repetitions with different random seeds.

To ensure the fairness of the comparison, we choose different network structures for different tasks to test, as shown in Table 2. In this table, C represents the convolutional layer, MP represents the max pooling layer, AP represents the average pooling layer, GAP represents the global average pooling layer, and pure numbers represent the fully connected layer. The number before all symbols represents the number of output channels or neurons, and the number after symbols represents the size of the kernel. Among them, for the task of image class, spatial information is more important than temporal information, and we adopt the residual structure of Fang et al. (2021a). In the training process of the SNN network, compared with the layer-by-layer stacked VGG structure, the gradient information of the residual structure can be transmitted through shortcuts, which can effectively alleviate the gradient error caused by the surrogate gradient function. For the event stream data, time characteristics and spatial characteristics are equally important, we use the VGG structure network to avoid the time characteristics in the event stream being disrupted.

3.2. Static data

As previously analyzed, SNNs are limited in their ability to process even static data if they cannot make valid judgments based on the entire spike train. Here, we select the benchmark of two static image classification tasks: CIFAR10 and CIFAR100 to verify that the proposed model is simple and effective. For the still image data, we did not adopt additional data augmentation methods for pulse sequences according to the time-dependent characteristics of SNN and only used data augmentation methods for the original image data. Specifically, we use Autoaugment (Cubuk et al., 2019) as an augmentation to improve the accuracy of image classification models. Compared to the classical crop adopted in most previous works, With random horizontal flipping and normalization, using auto augmentation improves the final classification result on CIFAR10 by about 0.4%.

3.2.1. Comparison with prior works

For image-type data, the main factor affecting the classification results is the spatial feature extraction ability of the model. Therefore, most of the work in this area focuses on ensuring the accuracy of the information represented by the spike train during the forward propagation or the accuracy of the gradient

TABLE 1 Hyper parameters.

Hyper parameter	CIFAR10	CIFAR100	DVS128 Gesture	CIFAR10-DVS
Training epoch	200	300	300	300
Batch size	128	128	16	32
Learning rate	$1e^{-1}$	$1e^{-1}$	$1e^{-3}$	$1e^{-3}$
Time steps	6	6	16	10

TABLE 2 Network structures.

Dataset	Architecture	Detail
CIFAR10	SEW ResNet18	64C3-MP2-64SEWblock*2-128SEWblock*2 - 256SEWblock*2-512SEWblock*2-GAP-10
CIFAR100	SEW ResNet18	64C3-MP2-64SEWblock*2-128SEWblock*2 - 256SEWblock*2-512SEWblock*2-GAP-100
DVS128 gesture	VGG-small	128C3-MP2-128C3-MP2-128C3-MP2- 128C3-MP2-512-11
DVSCIFAR10	VGG	16C3-AP2-32C3-AP2-64C3-64C3-AP2-128C3- 128C3-AP2-128C3-128C3-AP2-256-10

TABLE 3 Classification accuracy on static data.

Dataset	Proposals	Architecture	Timesteps	Accuracy(%)
CIFAR10	Rathi and Roy, 2023	VGG16	10	93.44
	Rathi and Roy, 2023	ResNet20	10	92.54
	Wu J. et al., 2021	CifarNet	8	90.98
	Fang et al., 2021a	VGG	8	93.50
	Zheng et al., 2021	ResNet19	6	93.16
	Deng et al., 2022	ResNet19	256	94.50
	This work	SEW-ResNet18	6	95.41
CIFAR100	Rathi et al., 2020	VGG11	125	67.87
	Rathi and Roy, 2023	ResNet20	5	64.09
	Deng et al., 2022	ResNet19	6	74.72
	This work	SEW-ResNet18	6	78.77

information in the backward propagation. As shown in Table 3, compared with other methods, our method only needs 6 timesteps to achieve the classification accuracy of 95.41% on CIFAR10 and 78.77% on CIFAR100.

3.2.2. Effects of different encoding methods

The commonly used static data coding methods include direct coding, rate coding, and latency coding. To be specific: (1) Direct coding is to input the pre-processed data directly into SNN as the current at every moment. (2) The commonly used method in rate coding is to encode the preprocessed data into the pulse train of Poisson distribution, where the expectation of Poisson distribution is related to the data value. (3) Latency coding is to map the pre-processed data size directly to the specific pulse firing moment, in which case the neuron fires only once. As mentioned before in this paper, different coding schemes have a significant impact on the classification results of the model in the classical SNN model, here, we test the effect of different coding methods on the CIFAR10 dataset as shown in Figure 5.

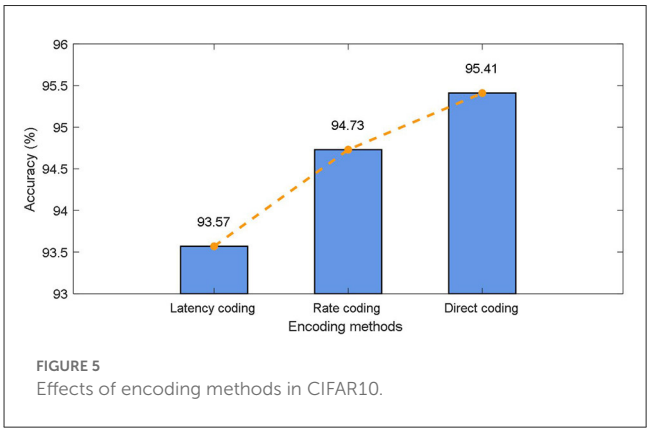


FIGURE 5
Effects of encoding methods in CIFAR10.

It can be found that, on the one hand, the direct encoding approach achieves the best results. The direct encoding scheme is more biologically interpretable. The photoreceptor neurons used to convert light signals into spike trains already have preliminary feature extraction capabilities in biological retinas. On the other

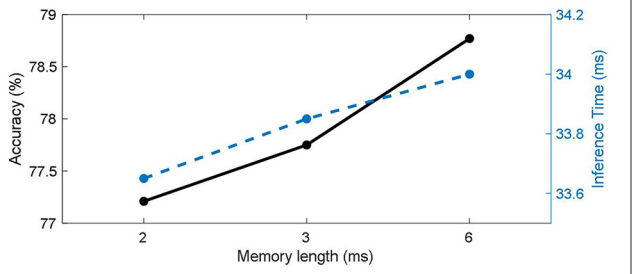


FIGURE 6
Effects of working memory length in CIFAR100.

hand, the proposed model achieves better classification results with different coding schemes. In particular, when latency coding is used, there is less drop in accuracy relative to direct encoding. This is attributed to the fact that the working memory in the proposed method integrates the input information at multiple time steps, allowing the model to obtain a larger field of view.

3.2.3. Effects of working memory length

The length of the working memory will affect the number of parameters that the model has, as well as its ability to integrate temporal information. We tested different working memory lengths separately on CIFAR100 with direct coding method, and the results are shown in Figure 6. It can be seen that the classification accuracy (solid black line) increases with memory length. For static data, longer working memory can give the model more time integration ability, which is also beneficial to the stability of the model. Therefore, the best classification results were obtained for networks with memory lengths up to 6 timesteps of the simulation duration. On the other hand, the reasoning time for a single batch (blue dashed line) increases with memory length.

3.3. Sequence data

Furthermore, we verify the effect of the proposed model on sequential data where time correlation is more important. Here, we choose two datasets, DVS128 Gesture and CIFAR10-DVS, for testing. In the CIFAR10-DVS dataset, the information of the original picture in the CIFAR10 dataset is obtained through motion, and the short-term information is more critical.

3.3.1. DVS data encoding

A variety of coding schemes exist for event streams, including time-surface (Lagorce et al., 2017), timestamp (Huang, 2021), and so on (Sabater et al., 2022; Wang et al., 2022). Here we use the more common approach in SNNs. Specifically, we use the encoding method in SpikingJelly (Fang et al., 2020) to convert the event stream data into multiple consecutive pictures. After that, the direct input coding method was used, that is, the image pixels were normalized and directly fed into the SNN as the input current.

The data augmentation method's impact on the results of event stream data is important. In this paper, to reduce the training cost and save time, we convert the event stream data into image data and then use the data augmentation method. In this article, we will use a data augmentation approach similar to that commonly used for still images. Specifically, for the DVS128 Gesture dataset, we used a random crop. For the CIFAR10-DVS data set, random crop and random horizontal flipping were used.

3.3.2. Comparison with prior works

For sequential event stream data, local and global temporal information is equally important. Local temporal information can be captured by using the time-varying property of spiking neurons, but the long-term information may disappear due to discrete neuron firing. Therefore, most works enhance the ability of SNN to process event streams by changing the neuron model or adding modules that can obtain long-term information, such as recurrent structures or attention modules.

It can be seen from Table 4 that our proposed method achieves a classification accuracy of 98.26% on the DVS128 Gesture data set and 80.1% on the CIFAR10-DVS data set.

3.3.3. Effects of working memory length

Event stream data is different from static images in that in addition to spatial features, temporal features are also always required, so the model's ability to extract temporal features has a significant impact on the final classification results. The working memory size affects the model's ability to extract temporal features. We tested it on the DVS128 Gesture dataset and the results are shown in Figure 7. It can be seen that with the increase in working memory size, the classification accuracy (solid black line) of the model shows a trend of decreasing now and then increasing. This is partly because the size of the working memory matches the length of key features of the data itself. When the working memory size is small, the model mainly classifies by spatial features. As working memory size grows, models tend to make judgments based on certain key short-term features, ignoring global information. As the working memory grows, the model acquires a global view and can classify actions based on the sequence. This is also in line with the example given at the beginning of the article. On the other hand, the inference time for a single batch (blue dashed line) increases with memory length.

4. Discussion

In this paper, we first analyze the key problems that affect SNN network processing spatiotemporal data, namely, SNN can only aggregate information in one direction, and the temporal receiver field is limited. To solve this problem, we introduce working memory into SNN, propose a new network structure by combining multi-delay synapses, and give an effective method to reduce the number of parameters. Then, we provide the implementation method of the proposed model in software and hardware. Finally, we test the performance of the proposed model on several static and dynamic datasets. Experimental results show

TABLE 4 Classification accuracy on sequence data.

Dataset	Proposals	Architecture	Timesteps	Accuracy (%)
DVS128 gesture	Zheng et al., 2021	CifarNet	40	96.87
	Wu Z. et al., 2021	VGG	60	97.56
	Fang et al., 2021b	ResNet19	20	97.57
	Fang et al., 2021a	ResNet19	16	97.92
	This work	VGG-small	16	98.26
CIFAR10-DVS	Wu Z. et al., 2021	VGG	10	70.4
	Fang et al., 2021b	VGG	20	74.8
	Yao et al., 2021	VGG	10	72.0
	Fang et al., 2021a	Wide-7B-Net	16	74.4
	This work	VGG	10	80.1

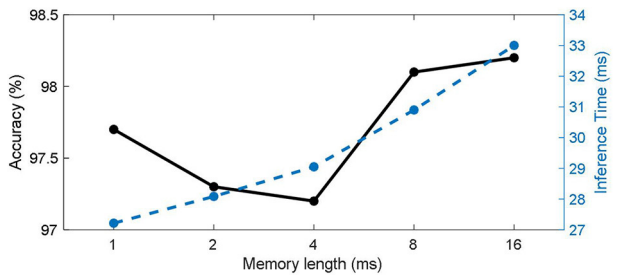


FIGURE 7
Effects of working memory length in DVS128 Gesture.

that SNN with working memory can effectively aggregate and rectify spatiotemporal features, thus improving the ability of SNN to process spatiotemporal data. Next, we discuss the effects of different encoding schemes on SNNWM, the effects of working memory length on SNNWM's inference speed and storage, and potential improvements for future SNNWM.

First, for static data, we analyzed the effects of three different coding schemes on SNNWM and found that direct coding had the best effect, while delayed coding had the worst effect, which was consistent with the results of most SNN-related studies. For direct coding, SNNWM achieved the highest classification accuracy because the same full precision value is stably entered at every moment. For rate coding and latency coding, since the input is a binary spike train and the simulation length determines the precision of the data received by SNNWM, so there will be certain performance degradation. The performance gap between the two is mainly due to the fact that rate coding has more input spikes than latency coding and the randomness in coding. For example, the Poisson distribution-based method used in this paper is equivalent to an augmentation of the training data and reduces overfitting.

Secondly, the length of Memory has a certain influence on the SNNWM's inference speed and storage. The introduction of working memory adds additional data slice and matrix multiplication operations and the storage grows as memory length increases. In particular, the additional memory consumption caused by working memory is approximately the same as adding an additional matrix multiplication of the size related to memory length to the original spiking neuron. Therefore, there is a trade-off between performance and resource consumption. At the same time, it can be found in the experiment on sequence data that the influence of memory length on model performance is not linear, so it is important to choose the appropriate memory length according to specific tasks.

Finally, SNNWM, as a neuron-level model refinement, can be used in various SNN network structures, such as the latest transformer-based model. At the same time, the learning algorithm used in this paper is based on BPTT (Back-Propagation Through Time) method, and the gradient needs to be transmitted step by step in time. Since SNNWM can be regarded as processing the spike train segment by segment, it is possible to try the gradient feedback at a segment level to reduce the training time and cost. Besides, this work is only a preliminary attempt at the visual classification, which can be extended to other types of data, such as speech, natural language processing, automatic driving, etc.

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

YC proposed the idea, performed the experiments, and wrote the manuscript. All authors contributed to the experiment's design, result interpretation, and writing. All authors contributed to the article and approved the submitted version.

Funding

This work was partially supported by the Science and Technology Support Program of Sichuan Province under Grant

2022YFG0313 and the National Science Foundation of China under Grant 62106038 and 61976043.

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

References

Amir, A., Taba, B., Berg, D. J., Melano, T., McKinstry, J. L., 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* (Honolulu, HI: IEEE Computer Society), 7388–7397. doi: 10.1109/CVPR.2017.781

Bohte, S. M., Kok, J. N., and La Poutré, J. A. (2000). "Spikeprop: backpropagation for networks of spiking neurons," in *ESANN* (Bruges), 419–424.

Cubuk, E. D., Zph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). "Auto augment: Learning augmentation strategies from data," in *IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA: Computer Vision Foundation; IEEE), 113–123. doi: 10.1109/CVPR.2019.00020

Deng, S., Li, Y., Zhang, S., and Gu, S. (2022). "Temporal efficient training of spiking neural network via gradient re-weighting," in *The Tenth International Conference on Learning Representations* (OpenReview.net).

el Assal, M., Tirilly, P., and Bilasco, I. M. (2022). "2D versus 3D convolutional spiking neural networks trained with unsupervised STDP for human action recognition," in *International Joint Conference on Neural Networks* (Padua: IEEE), 1–8. doi: 10.1109/IJCNN55064.2022.9892063

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *Spikingjelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed February 02, 2023).

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. (2021a). "Deep residual learning in spiking neural networks," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021*, eds M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan. p. 21056-21069. Available online at: https://proceedings.neurips.cc/paper/2021/hash/afe434653a898da20044041262b3ac74-Abstract.html

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021b). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *2021 IEEE/CVF International Conference on Computer Vision* (Montreal, QC: IEEE), 2641–2651. doi: 10.1109/ICCV48922.2021.00266

Huang, C. (2021). "Event-based timestamp image encoding network for human action recognition and anticipation," in *International Joint Conference on Neural Networks* (Shenzhen: IEEE), 1–9. doi: 10.1109/IJCNN52387.2021.9534386

Kim, R., and Sejnowski, T. J. (2021). Strong inhibitory signaling underlies stable temporal dynamics and working memory in spiking neural networks. *Nat. Neurosci.* 24, 129–139. doi: 10.1038/s41593-020-00753-w

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). *ImageNet Classification with Deep Convolutional Neural Networks*. New York, NY: Association for Computing Machinery. doi: 10.1145/3065386

Kwak, Y., and Curtis, C. E. (2022). Unveiling the abstract format of mnemonic representations. *Neuron* 110, 1822.e5–1828.e5. doi: 10.1016/j.neuron.2022.03.016

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). HOTS: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). CIFAR10-DVS: an event-stream dataset for object classification. *Front. Neurosci.* 11, 309. doi: 10.3389/fnins.2017.00309

Li, J., Li, D., Jiang, R., Xiao, R., Tang, H., and Tan, K. C. (2022). Vision-action semantic associative learning based on spiking neural networks for cognitive robot. *IEEE Comput. Intell. Mag.* 17, 27–38. doi: 10.1109/MCI.2022.3199623

Lobo, J. L., Del Ser, J., Bifet, A., and Kasabov, N. (2020). Spiking neural networks and online learning: an overview and perspectives. *Neural Netw.* 121, 88–100. doi: 10.1016/j.neunet.2019.09.004

Luo, X., Qu, H., Wang, Y., Yi, Z., Zhang, J., and Zhang, M. (2022). Supervised learning in multilayer spiking neural networks with spike temporal error backpropagation. *IEEE Trans. Neur. Netw. Learn. Syst.* 1–13. doi: 10.1109/TNNLS.2022.3164930

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., and Luo, Z. Q. (2022). "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA: IEEE), 12434–12443. doi: 10.1109/CVPR52688.2022.01212

Pan, Z., Chua, Y., Wu, J., Zhang, M., Li, H., and Ambikairajah, E. (2020). An efficient and perceptually motivated auditory neural encoding and decoding algorithm for spiking neural networks. *Front. Neurosci.* 13, 1420. doi: 10.3389/fnins.2019.01420

Pan, Z., Zhang, M., Wu, J., Wang, J., and Li, H. (2021). Multi-tone phase coding of interaural time difference for sound source localization with spiking neural networks. *IEEE/ACM Trans. Audio Speech Lang. Process.* 29, 2656–2670. doi: 10.1109/TASLP.2021.3100684

Rathi, N., and Roy, K. (2023). DIET-SNN: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Trans. Neur. Netw. Learn. Syst.* 34, 3174–3182. doi: 10.1109/TNNLS.2021.3111897

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). "Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation," in *8th International Conference on Learning Representations* (OpenReview.net).

Sabater, A., Montesano, L., and Murillo, A. C. (2022). "Event Transformer. A sparse-aware solution for efficient event data processing," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (New Orleans, LA: IEEE), 2676–2685. doi: 10.1109/CVPRW56347.2022.00301

Szegedy, C., Vanhucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV: IEEE Computer Society), 2818–2826. doi: 10.1109/CVPR.2016.308

Wang, Y., Zhang, X., Shen, Y., Du, B., Zhao, G., Cui, L., et al. (2022). Event-stream representation for human gaits identification using deep neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 3436–3449. doi: 10.1109/TPAMI.2021.3054886

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2021). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Trans. Neur. Netw. Learn. Syst.* 34, 446–460. doi: 10.1109/TNNLS.2021.3095724

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Wu, Z., Zhang, H., Lin, Y., Li, G., Wang, M., and Tang, Y. (2021). LIAF-Net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing. *IEEE Trans. Neur. Netw. Learn. Syst.* 33, 6249–6262. doi: 10.1109/TNNLS.2021.3073016

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z. -X. et al. (2021). "Temporal-wise attention spiking neural networks for event streams classification," in *2021 IEEE/CVF International Conference on Computer Vision* (Montreal, QC: IEEE), 10201–10210. doi: 10.1109/ICCV48922.2021.01006

Zhang, W., and Li, P. (2019). "Spike-train level backpropagation for training deep recurrent spiking neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, eds H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc, E. B. Fox, and R. Garnett (Vancouver, BC: NeurIPS), 7800–7811. Available online at: https://proceedings.neurips.cc/paper/2019/hash/f42a37d114a480b6b57b60ea9a14a9d2-Abstract.html

Zhang, W., and Li, P. (2021). "Spiking neural networks with laterally-inhibited self-recurrent units," in *International Joint Conference on Neural Networks* (Shenzhen: IEEE), 1–8. doi: 10.1109/IJCNN52387.2021.9533726

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). "Going deeper with directly-trained larger spiking neural networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021* (AAAI Press), 11062–11070.

Zhu, L., Wang, X., Chang, Y., Li, J., Huang, T., and Tian, Y. (2022). "Event-based video reconstruction via potential-assisted spiking neural network," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA: IEEE), 3584–3594. doi: 10.1109/CVPR52688.2022.00358

frontiers | Frontiers in Neuroscience

TYPE Review
PUBLISHED 16 June 2023
DOI 10.3389/fnins.2023.1209795

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Jibin Wu,
Hong Kong Polytechnic University,
Hong Kong SAR, China
Zhaofei Yu,
Peking University, China

*CORRESPONDENCE
Zhe Ma
✉ mazhe_thu@163.com

RECEIVED 21 April 2023
ACCEPTED 01 June 2023
PUBLISHED 16 June 2023

CITATION
Guo Y, Huang X and Ma Z (2023) Direct learning-based deep spiking neural networks: a review. *Front. Neurosci.* 17:1209795. doi: 10.3389/fnins.2023.1209795

Direct learning-based deep spiking neural networks: a review

Yufei Guo[1,2], Xuhui Huang[1,2] and Zhe Ma[1,2]*

[1]Intelligent Science & Technology Academy of CASIC, Beijing, China, [2]Scientific Research Laboratory of Aerospace Intelligent Systems and Technology, Beijing, China

The spiking neural network (SNN), as a promising brain-inspired computational model with binary spike information transmission mechanism, rich spatially-temporal dynamics, and event-driven characteristics, has received extensive attention. However, its intricately discontinuous spike mechanism brings difficulty to the optimization of the deep SNN. Since the surrogate gradient method can greatly mitigate the optimization difficulty and shows great potential in directly training deep SNNs, a variety of direct learning-based deep SNN works have been proposed and achieved satisfying progress in recent years. In this paper, we present a comprehensive survey of these direct learning-based deep SNN works, mainly categorized into accuracy improvement methods, efficiency improvement methods, and temporal dynamics utilization methods. In addition, we also divide these categorizations into finer granularities further to better organize and introduce them. Finally, the challenges and trends that may be faced in future research are prospected.

KEYWORDS

spiking neural network, brain-inspired computation, direct learning, deep neural network, energy efficiency, spatial-temporal processing

1. Introduction

The Spiking Neural Network (SNN) has been recognized as one of the brain-inspired neural networks due to its bio-mimicry of the brain neurons. It transmits information by firing binary spikes and can process the information in a spatial-temporal manner (Wu et al., 2019a; Wu Y. et al., 2019; Zhang et al., 2020a,b; Fang et al., 2021b). This event-driven and spatial-temporal manner makes the SNN very efficient and good at handling temporal signals, thus receiving a lot of research attention, especially recently.

Despite the energy efficiency and spatial-temporal processing advantages, it is a challenge to train deep SNNs due to the firing process of the SNN is undifferentiable, thus making it impossible to train SNNs via gradient-based optimization methods. At first, many works leverage the spike-timing-dependent plasticity (STDP) approach (Lobov et al., 2020), which is inspired by biology, to update the SNN weights. However, STDP cannot help train large-scale networks yet, thus limiting the practical applications of the SNN. There are two widely used effective pathways to obtain deep SNNs up to now. First, the ANN-SNN conversion approach (Han and Roy, 2020; Li et al., 2021a; Bu et al., 2022, 2023; Li and Zeng, 2022; Liu et al., 2022; Wang Y. et al., 2022) converts a well-trained ANN to an SNN by replacing the activation function from ReLU with spiking activation. It provides a fast way to obtain an SNN. However, it is limited in the rate-coding scheme and ignores the rich temporal dynamic behaviors of SNNs. Second, the surrogate gradient (SG)-based direct learning approach (Wu Y. et al., 2018; Fang et al., 2021a; Li et al., 2021b; Guo et al., 2022a) tries to find an alternative differentiable surrogate function to replace the undifferentiable firing activity when doing back-propagation of the spiking neurons. Since SG can handle temporal data and provide decent performance with few time-steps on the large-scale dataset, it has received more attention recently.

Considering the sufficient advantages and rapid development of the direct learning-based deep SNN, a comprehensive and systematic survey on this kind of work is essential. Previously related surveys (Ponulak and Kasinski, 2011; Roy et al., 2019; Tavanaei et al., 2019; Wang et al., 2020; Yamazaki et al., 2022; Zhang D. et al., 2022) have begun to classify existing works mainly based on the key components of SNNs: biological neurons, encoding methods, SNN structures, SNN learning mechanisms, software and hardware frameworks, datasets, and applications. Though such classification is intuitive to general readers, it is difficult for them to grasp the challenges and the landmark work involved. While in this survey, we provide a new perspective to summarize these related works, i.e., starting from analyzing the characteristics and difficulties of the SNN, and then classify them into (i) accuracy improvement methods, (ii) efficiency improvement methods, and (iii) temporal dynamics utilization methods, based on the solutions for corresponding problems or the utilization of SNNs' advantages.

Further, these categories are divided into finer granularities: (i) accuracy improvement methods are subdivided as improving representative capabilities and relieving training difficulties; (ii) efficiency improvement methods are subdivided as network compression techniques and sparse SNNs; (iii) temporal dynamics utilization methods are subdivided as sequential learning and cooperating with neuromorphic cameras. In addition to the classification by using strengths or overcoming weaknesses of SNNs, these recent methods can also be divided into the neuron level, network structure level, and training technique level, according to where these methods actually work. The classifications and main techniques of these methods are listed in Tables 1, 2. Finally, some promising future research directions are provided.

The organization of the remaining part is given as follows, Section 2 introduces the preliminary for spiking neural networks. The characteristics and difficulties of the SNN are also analyzed in Section 2. Section 3 presents the recent advances falling into different categories. Section 4 points out future research trends and concludes the review.

2. Preliminary

Since the neuron models are not the focus of the paper, here, we briefly introduce the commonly used discretized Leaky Integrate-and-Fire (LIF) spiking neurons to show the basic characteristic and difficulties in SNNs, which can be formulated by

$$U_l^t = \tau U_l^{t-1} + \mathbf{W}_l O_{l-1}^t, \qquad U_l^t < V_{\text{th}}, \tag{1}$$

where U_l^t is the membrane potential at t-th time-step for l-th layer, O_{l-1}^t is the spike output from the previous layer, $\mathbf{W}_l$ is the weight matrix at l-th layer, V_{th} is the firing threshold, and τ is a time leak constant for the membrane potential, which is in $(0, 1)$. When τ is 1, the above equation will degenerate to the Integrate-and-Fire (IF) spiking neuron.

Characteristic 1. *Rich spatially-temporal dynamics. Seen from Equation (1), different from ANNs SNNs enjoy the unique spatial-temporal dynamic in the spiking neuron model.*

Then, when the membrane potential exceeds the firing threshold, it will fire a spike and then fall to resting potential, given by

$$O_l^t = \begin{cases} 1, & \text{if } U_l^t \geq V_{\text{th}} \\ 0, & \text{otherwise} \end{cases}. \tag{2}$$

Characteristic 2. *Efficiency. Since the output is a binary tensor, the multiplications of activations and weights can be replaced by additions, thus enjoying high energy efficiency. Furthermore, when there is no spike output generated, the neuron will keep silent. This event-driven mechanism can further save energy when implemented in neuromorphic hardware.*

Characteristic 3. *Limited representative ability. Obviously, transmitting information by quantizing the real-valued membrane potentials into binary output spikes will introduce the quantization error in SNNs, thus causing information loss (Guo et al., 2022b; Wang et al., 2023). Furthermore, the binary spike feature map from a timestep cannot carry enough information like the real-valued one in ANNs (Guo et al., 2022d). These two problems limit the representative ability of SNN to some extent.*

Characteristic 4. *Non-differentiability. Another thorny problem in SNNs is the non-differentiability of the firing function.*

To demonstrate this problem, we formulate the gradient at the layer l by the chain rule, given by

$$\frac{\partial L}{\partial \mathbf{W}_l} = \sum_t \left(\frac{\partial L}{\partial O_l^t} \frac{\partial O_l^t}{\partial U_l^t} + \frac{\partial L}{\partial U_l^{t+1}} \frac{\partial U_l^{t+1}}{\partial U_l^t} \right) \frac{\partial U_l^t}{\partial \mathbf{W}_l}, \tag{3}$$

where $\frac{\partial O_l^t}{\partial U_l^t}$ is the gradient of firing function at t-th time-step for l-th layer and is 0 almost everywhere, while infinity at V_{th}. As a consequence, the gradient descent $(\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \frac{\partial L}{\partial \mathbf{W}_l})$ either freezes or updates to infinity.

Most existing direct learning-based SNN works focus on solving difficulties or utilizing the advantages of SNNs. Boosting the representative ability and mitigating the non-differentiability can both improve SNN's accuracy. From this perspective, we organize the recent advances in the SNN field as accuracy improvement methods, efficiency improvement methods, and temporal dynamics utilization methods.

3. Recent advances

In recent years, a variety of direct learning-based deep spiking neural networks have been proposed. Most of these methods fall into solving or utilizing the intrinsic disadvantages or advantages of SNNs. Based on this, in the section, we classify these methods into accuracy improvement methods, efficiency improvement methods, and temporal dynamics utilization methods. In addition, these classifications are also organized in different aspects with a comprehensive analysis. Tables 1, 2 summarizes the surveyed SNN methods in different categories.

Note that the direct learning methods can be divided into time-based methods and activation-based methods based on whether the gradient represents spike timing (time-based) or spike scale (activation-based; Zhu Y. et al., 2022). In time-based methods, the gradients represent the direction where the timing of a spike should be moved, i.e., be moved leftward or rightward on the time axis. The SpikeProp (Bohte et al., 2002) and its variants

TABLE 1 Overview of direct learning-based deep spiking neural networks: part I.

Type		Method	Key technology	On the level*		
				NL	NSL	TTL
Accuracy improvement	Improving representative capabilities	LSNN (Bellec et al., 2018)	Adaptive threshold	✓		
		LTMD (Wang S. et al., 2022)	Adaptive threshold	✓		
		BDETT (Ding et al., 2022)	Dynamic threshold	✓		
		PLIF (Fang et al., 2021b)	Learnable leak constant	✓		
		Plastic synaptic delays (Yu et al., 2022a)	Learnable leak constant	✓		
		Diet-SNN (Rathi and Roy, 2020)	Learnable leak constant& threshold	✓		
		DS-ResNet (Feng et al., 2022)	Multi-firing & Act before Add-ResNet	✓	✓	
		SNN-MLP (Li W. et al., 2022)	Group LIF	✓		
		GLIF Yao et al., 2022	Unified gated LIF	✓		
		Augmented spikes (Yu et al., 2022b)	Augmented spikes	✓		
		InfLoR-SNN (Shen et al., 2023)	Leaky integrate and fire or burst	✓		
		MT-SNN (Wang et al., 2023)	Multiple threshold approach	✓		
		SEW-ResNet (Fang et al., 2021a)	Act before ADD form-based ResNet		✓	
		MS-ResNet (Hu et al., 2021)	Pre-activation form-based ResNet		✓	
		AutoSNN (Na et al., 2022)	Neural architecture search		✓	
		SNASNet (Kim et al., 2022a)	Neural architecture search		✓	
		TA-SNN (Yao et al., 2021)	Attention mechanism		✓	
		TCJA-SNN (Zhu et al., 2022)	Attention mechanism		✓	
		Real spike (Guo et al., 2022d)	Training-inference decoupled structure		✓	
		IM-loss (Guo et al., 2022a)	Information maximization loss			✓
		RecDis-SNN (Guo et al., 2022c)	Membrane potential distribution loss			✓
		Distilling spikes (Kushawaha et al., 2021)	Knowledge distillation		✓	✓
		Local tandem learning (Yang et al., 2022)	Tandem learning			✓
		sparse-KD (Xu et al., 2023a)	Knowledge distillation			✓
		KDSNN (Xu et al., 2023b)	Knowledge distillation			✓
		SNN distillation (Takuya et al., 2021)	Knowledge distillation			✓
	Relieving training difficulties	SuperSpike (Zenke and Ganguli, 2018)	Fixed surrogate gradient			✓
		LISNN (Cheng et al., 2020)	Fixed surrogate gradient			✓
		IM-Loss (Guo et al., 2022a)	Dynamic surrogate gradient			✓
		Gradual surrogate gradient (Guo et al., 2022a)	Dynamic surrogate gradient			✓
		Differentiable spike (Li et al., 2021b)	Learnable surrogate gradient			✓
		SpikeDHS (Leng et al., 2022)	Differentiable surrogate gradient search			✓
		DSR (Meng et al., 2022)	Differentiation on spike representation			✓
		STDBP (Zhang M. et al., 2022)	Rectified postsynaptic potential function		✓	
		SEW-ResNet (Fang et al., 2021a)	Act before ADD form-based ResNet		✓	
		MS-ResNet (Hu et al., 2021)	Pre-activation form-based ResNet		✓	
		NeuNorm (Wu Y. et al., 2019)	Constructing auxiliary feature maps			✓

(Continued)

TABLE 1 (Continued)

Type		Method	Key technology	On the level*		
				NL	NSL	TTL
		tdBN (Zheng et al., 2021)	Threshold-dependent batch normalization			✓
		BNTT (Kim and Panda, 2021)	Temporal batch normalization through time			✓
		PSP-BN (Ikegawa et al., 2022)	Postsynaptic potential normalization		✓	
		TEBN (Kim and Panda, 2021)	Temporal effective batch normalization			✓
		RecDis-SNN (Guo et al., 2022c)	Membrane potential distribution loss			✓
		TET (Deng et al., 2022)	Temporal regularization loss			✓
		Tandem learning (Wu et al., 2021a)	Tandem learning			✓
		Progressive tandem learning (Wu et al., 2021b)	Progressive tandem learning			✓
		Joint A-SNN (Guo et al., 2023)	Joint training of ANN and SNN			✓

*NL, neuron Level; NSL, network structure level; TTL, training technique level.

(Booij and tat Nguyen, 2005; Xu et al., 2013; Hong et al., 2019) all belong to this kind of method and they adopt the negative inverse of the time derivative of membrane potential function to approximate the derivative of spike timing to membrane potential. Since most of the time-based methods would restrict each neuron to fire at most once, in Zhou et al. (2021), the spike time is directly taken as the state of a neuron. Thus the relation of neurons can be modeled by the spike time and the SNN can be trained similarly to an ANN. Though the time-based methods enjoy less computation cost than the activation-based methods and many works (Zhang and Li, 2020; Zhu Y. et al., 2022) have greatly improved the accuracy of the field, it is still difficult to train deep time-based SNN models and apply them to large-scale datasets, e.g., ImageNet. Considering the limits of the time-based methods and the topic of summarizing the recent deep SNNs here, we mainly focus on activation-based methods in the paper.

3.1. Accuracy improvement methods

As aforementioned, the limited information capacity and the non-differentiability of firing activity of the SNN cause its accuracy loss for wide tasks. Therefore, to mitigate the accuracy loss in the SNN, a great number of methods devoted to improving the representative capabilities and relief training difficulties of SNNs have been proposed and achieved successful improvements in the past few years.

3.1.1. Improving representative capabilities

Two problems result in the representative ability decreasing of the SNN, the process of firing activity will induce information loss, which has been proved in Guo et al. (2022b) and binary spike maps suffer the limited information capacity, which has been proved in Guo et al. (2022d). These problems can be mitigated on the neuron level, network structure level, and training technique level.

3.1.1.1. On the neuron level

A common way to boost the representative capability of the SNN is to make some hyper-parameters in the spiking neuron learnable. In LSNN (Bellec et al., 2018) and LTMD (Wang S. et al., 2022), the adaptive threshold spike neuron was proposed to enhance the computing and learning capabilities of SNNs. Further, a novel bio-inspired dynamic energy-temporal threshold, which can be adjusted dynamically according to input data for SNNs was introduced in the BDETT (Ding et al., 2022). Some works adopted the learnable membrane time constant in spiking neurons (Zimmer et al., 2019; Yin et al., 2020; Fang et al., 2021b; Luo et al., 2022; Yu et al., 2022a). Combining these two manners, Diet-SNN (Rathi and Roy, 2020) simultaneously adopted the learnable membrane leak and firing threshold.

There are also some works focusing on embedding more factors in the spiking neuron to improve its diversity. A multi-level firing (MLF) unit, which contains multiple LIF neurons with different level thresholds thus could generate more quantization spikes with different thresholds was proposed in DS-ResNet (Feng et al., 2022). A full-precision LIF to communicate between patches in Multi-Layer Perceptron (MLP), including horizontal LIF and vertical LIF in different directions was proposed in SNN-MLP (Li W. et al., 2022). SNN-MLP used group LIF to extract better local features. In GLIF (Yao et al., 2022), to enlarge the representation space of spiking neurons, a unified gated leaky integrate-and-fire Neuron was proposed to fuse different bio-features in different neuronal behaviors via embedding gating factors. In augmented spikes (Yu et al., 2022b), a special spiking neuron model was proposed to process augmented spikes, where additional information can be carried from spike strength and latency. This neuron model extends the computation with an additional dimension and thus could be of great significance for the representative ability of the SNN. In LIFB (Shen et al., 2023), a new spiking neuron model called the Leaky Integrate and Fire or Burst was proposed. The neuron model exhibits three modes including resting, regular spike, and burst spike, which significantly enriches the representative capability. Similar to LIFB, MT-SNN (Wang et al., 2023) proposed a multiple

TABLE 2 Overview of direct learning-based deep spiking neural networks: part II.

Type		Method	Key technology	On the level*		
				NL	NSL	TTL
Efficiency improvement	Network compression techniques	Spatio-temporal pruning (Chowdhury et al., 2021)	Spatio-temporal pruning			✓
		SD-SNN (Han et al., 2022)	Pruning-regeneration method			✓
		Grad R (Chen et al., 2021)	Pruning-regeneration method			✓
		Temporal pruning (Chowdhury et al., 2022)	Temporal pruning			✓
		Autosnn (Na et al., 2022)	Neural architecture searching		✓	
		SNASNet (Kim et al., 2022a)	Neural architecture searching		✓	
		Lottery ticket hypothesis (Kim et al., 2022b)	Lottery ticket hypothesis		✓	
		Distilling spikes (Kushawaha et al., 2021)	Knowledge distillation		✓	✓
		Local tandem learning (Yang et al., 2022)	Tandem learning			✓
		sparse-KD (Xu et al., 2023a)	Knowledge distillation			✓
		KDSNN (Xu et al., 2023b)	Knowledge distillation			✓
		SNN distillation (Takuya et al., 2021)	Knowledge distillation			✓
	Sparse SNNs	ASNN (Zambrano and Bohte, 2016)	A lot of adaptive spiking neurons	✓		
		Correlation-based regularization (Han and Lee, 2022)	Correlation-based regularizer			✓
		Superspike (Zenke and Ganguli, 2018)	Heterosynaptic regularization term			✓
		RecDis-SNN (Guo et al., 2022c)	Membrane potential distribution			✓
		Low-activity SNN (Pellegrini et al., 2021)	Regularization term		✓	✓
Temporal dynamics utilization	Sequential learning	Sequence approximation (She et al., 2021)	Dual-search-space optimization			✓
		Sequential learning (Ponghiran and Roy, 2022)	Improved recurrence dynamics	✓		
		SNN_HAR (Li Y. et al., 2022)	Spatio-temporal extraction		✓	
		Robust SNN (Nomura et al., 2022)	Temporal penalty settings		✓	
		Tandem learning-based SNN model (Wu et al., 2020)	Tandem learning			✓
		SG-based SNN model (Bittar and Garner, 2022b)	Surrogate gradient method			✓
		Combination-based SNN (Bittar and Garner, 2022a)	Combination of many techniques	✓	✓	
		Low-activity SNN (Pellegrini et al., 2021)	Regularization term			✓
		SNNCNN (Sadovsky et al., 2023)	Combination of CNNs and SNNs		✓	✓
		RSNNs (Yin et al., 2021)	activity-regularizing SG		✓	✓
	Cooperating with neuromorphic cameras	daptive-spikenet (Kosta and Roy, 2022)	Learnable neuronal dynamics	✓		
		StereoSpike (Rançon et al., 2021)	Modified U-Net-like architecture		✓	✓
		SuperFast (Gao et al., 2022)	Event-enhanced frame interpolation		✓	
		E-SAI (Yu L. et al., 2022)	Synthetic aperture imaging method		✓	
		EVSNN (Zhu L. et al., 2022)	Potential-assisted SNN	✓	✓	
		Spiking-Fer (Barchid et al., 2023)	Deep CSNN		✓	

(Continued)

TABLE 2 (Continued)

Type		Method	Key technology	On the level*		
				NL	NSL	TTL
		Automotive detection (Cordone et al., 2022)	PLIF & SG & Event encoding	✓		✓
		STNet (Zhang J. et al., 2022)	Spiking transformer network		✓	
		LaneSNNs (Viale et al., 2022)	offline supervised learning rule			✓
		HALSIE (Biswas et al., 2022)	Hybrid approach		✓	
		SpikeMS (Parameshwara et al., 2021)	Spatio-temporal loss			✓
		Event-based pose tracking (Zou et al., 2023)	Spiking spatiotemporal transformer		✓	

*NL, neuron Level; NSL, network structure level; TTL, training technique level.

threshold approach to firing different spike modes to alleviate the quantization error, such that it could reach a high accuracy at fewer steps.

Different from these works, InfLoR-SNN (Guo et al., 2022b) proposed a membrane potential rectifier (MPR), which can adjust the membrane potential to a new value closer to quantization spikes than itself before firing activity. MPR directly handles the quantization error problem in SNNs, thus improving the representative ability.

3.1.1.2. On the network structure level

To increase the SNN diversity, some works advocate for improving the SNN architecture. In SEW-ResNet (Fang et al., 2021a) and DS-ResNet (Feng et al., 2022), the widely used standard ResNet backbone is replaced by activation before addition form-based ResNet. In this way, the blocks in the network will fire positive integer spikes. Its representation capability will no doubt be increased, however, the advantages of event-driven and multiplication-addition transform in SNNs will be lost in the meantime. To solve the aforementioned problem, MS-ResNet (Hu et al., 2021) adopted the pre-activation form-based ResNet. In this way, the spike-based convolution can be retained. The difference between these methods is shown in Figure 1. However, these SNN architectures are all manually designed. For designing well-performed SNN models automatically, AutoSNN (Na et al., 2022) and SNASNet (Kim et al., 2022a) combined the Neural Architecture Search (NAS) approach to find better SNN architectures. And TA-SNN (Yao et al., 2021) and TCJA-SNN (Zhu et al., 2022) leveraged the learnable attention mechanism to improve the SNN performance.

Different from changing the network topology, Real Spike (Guo et al., 2022d) provides a training-inference decoupled structure. This method enhances the representation capacity of the SNN by learning real-valued spikes during training. While in the inference phase, the rich representation capacity will be transferred from spike neurons to the convolutions by a re-parameterization technique, and meanwhile, the real-valued spikes will be transformed into binary spikes, thus maintaining the event-driven and multiplication-addition transform advantages of SNNs.

Besides, increasing the timestep of SNN will undoubtedly improve the SNN accuracy too, which has been proved in many works (Wu Y. et al., 2018, 2019; Fang et al., 2021a). To some extent, increasing the timestep is equivalent to increasing neuron output bits through the temporal dimension, which will increase the representation capability of feature map (Feng et al., 2022). However, using more timesteps achieves better performance at the cost of increasing inference time.

3.1.1.3. On the training technique level

Some works attempted to improve the representative capability of the SNN on the training technique level, which can be categorized as regularization and distillation. Regularization is a technique that introduces another loss term to explicitly regularize the membrane potential or spike distribution to retain more useful information in the network that could indirectly help train the network as follows,

$$\mathcal{L}_{Total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{DL} \tag{4}$$

where $\mathcal{L}_{CE}$ is the common cross-entropy loss, $\mathcal{L}_{DL}$ is the distribution loss for learning the proper membrane potential or spike, and λ is a coefficient to balance the effect of the two types of losses. IM-Loss (Guo et al., 2022a) argues that improving the activation information entropy can reduce the quantization error, and proposed an information maximization loss function that can maximize the activation information entropy. In RecDis-SNN (Guo et al., 2022c), a loss for membrane potential distribution to explicitly penalize three undesired shifts was proposed. Though the work is not designed for reducing quantization error specifically, it still results in a bimodal membrane potential distribution, which has been proven can mitigate the quantization error problem.

The distillation methodology aims to help train a small student model by transferring knowledge of a rather large trained teacher model based on the consensus that the representative ability of a teacher model is better than that of the student model. Recently, some interesting works that introduce the distillation method in the SNN domain were proposed. In Kushawaha et al. (2021), a big teacher SNN model is used to guide the small SNN counterpart learning. While in Yang et al. (2022), Takuya et al. (2021), and Xu et al. (2023a,b) an ANN-teacher is used to guide SNN-student learning. In specific, Local Tandem Learning (Yang et al., 2022) uses the intermediate feature representations of the ANN to supervise the learning of SNN. While in sparse-KD (Xu et al., 2023a), the logit output of the ANN was adopted to guide the learning of the SNN. Furthermore, KDSNN (Xu et al., 2023b) and SNN

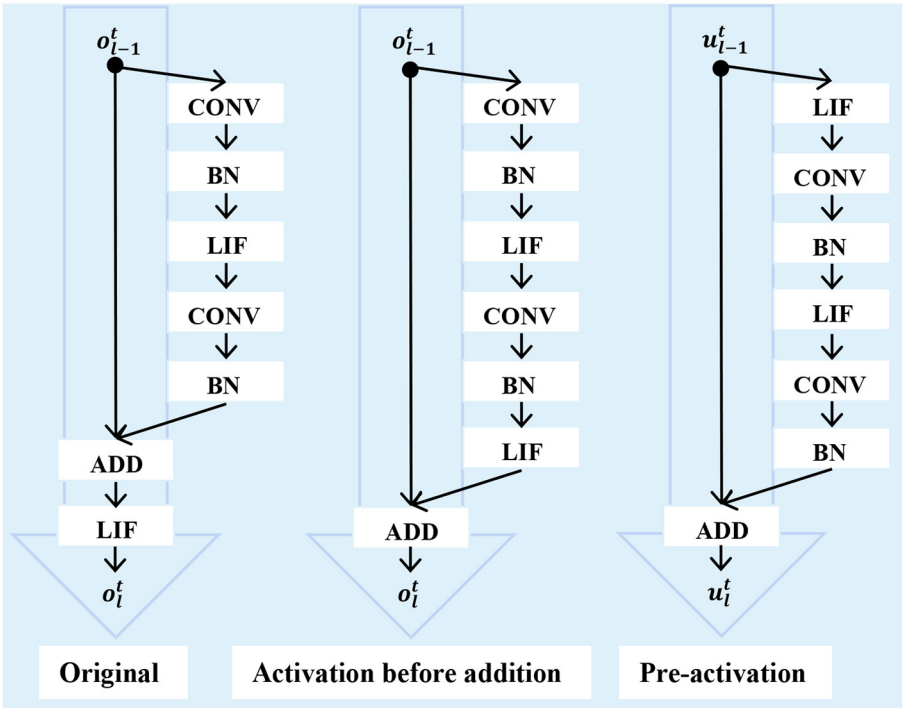


FIGURE 1
Different SNN ResNet architectures.

distillation (Takuya et al., 2021) used both feature-based and logit-based information to distill the SNN.

3.1.2. Relieving training difficulties

The non-differentiability of the firing function impedes the deep SNN direct training. To handle this problem, recently, using the surrogate gradient (SG) function for spiking neurons has received much attention. SG method utilizes a differentiable surrogate function to replace the non-differentiable firing activity to calculate the gradient in the back-propagation (Neftci et al., 2019; Wu Y. et al., 2019; Rathi and Roy, 2020; Fang et al., 2021a). Though the SG method can alleviate the non-differentiability problem, there exists an obvious gradient mismatch between the gradient of the firing function and the surrogate gradient. And the problem easily leads to under-optimized SNNs with severe performance degradation. Intuitively, an elaborately designed surrogate gradient can help to relieve the gradient mismatch in the backward propagation. As a consequence, some works are focusing on designing better surrogate gradients. In addition, the gradient explosion/vanishing problem in SNNs is severer over ANNs, due to the adoption of `tanh`-like function for most SG methods. There are also some works focusing on handling the gradient explosion/vanishing problem. Note that, these methods in this section can also be classified as the improvement on the neuron level, network structure level, and training technique level, which can be seen in the Table 1. Nevertheless, to better introduce these works, we still organize them as designing the better surrogate gradient and relieving the gradient explosion/vanishing problem.

3.1.2.1. Designing the better surrogate gradient

Most earlier works adopt fixed SG-based methods to handle the non-differentiability problem. For example, the derivative of a truncated quadratic function, the derivatives of a sigmoid, and a rectangular function were respectively adopted in Bohte (2011), Zenke and Ganguli (2018), and Cheng et al. (2020). However, such a strategy would limit the learning capacity of the network. To this end, a dynamic SG method was proposed in Guo et al. (2022a) and Chen et al. (2022), where the SG could change along with epochs as follows,

$$\varphi(x) = \frac{1}{2}\tanh(K(i)(x - V_{\text{th}})) + \frac{1}{2} \tag{5}$$

where $\varphi(x)$ is the backward approximation function for the firing activity and $K(i)$ is a dynamic coefficient that changes along with the training epoch as follows,

$$K(i) = \frac{(10^{\frac{i}{N}} - 10^0)K_{max} + (10^1 - 10^{\frac{i}{N}})K_{min}}{9} \tag{6}$$

where K_{min} and K_{max} are the lower bound and the upper bound of K, and i is the index of epoch starting from 0 to $N - 1$. The $\varphi(x)$ and its gradient can be seen in Figure 2. Driven by $K(i)$, it will gradually evolve to the firing function, thus ensuring sufficient weight updates at the beginning and accurate gradients at the end of the training. Nevertheless, the above SG methods are still designed manually. To find the optimal solution, in Li et al. (2021b), the Differentiable Spike method that can adaptively evolve during training to find the optimal shape and smoothness for gradient estimation based on the finite difference technique was proposed. Then, in Leng et al. (2022), combined with the NAS technique, a

differentiable SG search (DGS) method to find the optimized SGs for SNN was proposed. Different from designing a better SG for firing function, DSR (Meng et al., 2022) derived that the spiking dynamics with spiking neural models can be represented as some sub-differentiable mapping and trained the SNNs by the gradients of the mapping, thus avoiding the non-differentiability problem in SNN training.

3.1.2.2. Relieving the gradient explosion/vanishing problem

The gradient explosion or vanishing problem is still severe in SG-only methods. There are three kinds of methods to solve this problem: using improved neurons or architectures, improved batch normalizations, and regularization. In Zhang M. et al. (2022), a simple yet efficient rectified linear postsynaptic potential function (ReL-PSP) for spiking neurons, which benefits for handling the gradient explosion problem, was proposed. On the network architecture level, SEW-ResNet (Fang et al., 2021a) showed that standard spiking ResNet is inapplicable to overcome identity mapping and vanishing/explosion gradient problems and advised using ResNet with activation before addition form. Recently, the pre-activation form-based ResNet was explored in MS-ResNet (Hu et al., 2021). This network topology can simultaneously handle the gradient explosion/vanishing problem and retain the advantages of the SNN.

The normalization approaches are widely used in ANNs to train well-performed models, and these approaches are also introduced in the SNN field to handle the vanishing/explosion gradient problems. For example, NeuNorm (Wu Y. et al., 2019) normalized the data along the channel dimension like BN in ANNs through constructing auxiliary feature maps. Threshold-dependent batch normalization (tdBN; Zheng et al., 2021) considers the SNN normalization from a temporal perspective and extends the scope of BN to the additional temporal dimension. Furthermore, some works (Kim and Panda, 2021; Duan et al., 2022; Ikegawa et al., 2022) argued that the distributions of different timesteps vary wildly, thus bringing a negative impact when using shared parameters. Subsequently, the temporal Batch Normalization Through Time (BNTT), postsynaptic potential normalization (PSP-BN), and temporal effective batch normalization (TEBN) that can regulate the spike flows by utilizing separate sets of BN parameters on different timesteps were proposed. Though adopting temporal BN parameters on different timesteps can obtain more well-performed SNN models, this kind of BN technique can not fold the BN parameters into the weights and will increase the computations and running time in the inference stage, which should also be noticed.

Using the regularization loss can also mitigate the gradient explosion/vanishing problem. In RecDis-SNN (Guo et al., 2022c), a new perspective to further classify the gradient explosion/vanishing difficulty of SNNs into three undesired shifts of the membrane potential distribution was presented. To avoid these undesired shifts, a membrane potential regularization loss was proposed in RecDis-SNN, this loss introduces no additional operations in the SNN inference phase. In TET (Deng et al., 2022), an extra temporal regularization loss to compensate for the loss of momentum in the gradient descent with SG methods was proposed. With this loss, TET can converge into flatter minima with better generalizability.

Since ANNs are fully differentiable to be trained with gradient descent, there is also some work utilizing ANN to guide the SNN's optimization (Wu et al., 2021a,b; Guo et al., 2023). In Wu et al. (2021a) a tandem learning framework was proposed, that consists of an SNN and an ANN that share the same weight. In this framework, the spike count as the discrete neural representation in the SNN would be presented to the coupled ANN activation function in the forward phase. And in the backward phase, the error back-propagation is performed on the ANN to update the shared weight for both the SNN and the ANN. Furthermore, in Wu et al. (2021b), a progressive tandem learning framework was proposed, that introduces a layer-wise learning method to fine-tune the shared network weights. Considering the difference between the ANN and SNN, Joint A-SNN (Guo et al., 2023) developed a partial weight-sharing regime for the joint training of weight-shared ANN and SNN, that applies the Singular Value Decomposition (SVD) to the weights parameters and keep the same eigenvectors while the separated eigenvalues for the ANN and SNN.

3.2. Efficiency improvement methods

An important reason why have SNNs received extensive attention recently is that they are seen as more energy efficient than ANNs due to their event-driven computation mechanism and the replacement of energy-consuming weight multiplication with addition. To further explore the efficiency advantages of SNNs so that they can be applied to energy-constrained devices is also a hot topic in the SNN field. This kind of method can be mainly categorized into network compression techniques and sparse SNNs.

3.2.1. Network compression techniques

Network compression techniques have been widely used in ANNs. There are also some works applying these techniques in SNNs. In the literature, approaches for compressing deep SNNs can be classified into three categories: parameter pruning, NAS, and knowledge distillation.

3.2.1.1. Parameter pruning

Parameter pruning mainly focuses on eliminating the redundant parameters in the model by removing the uncritical ones. SNNs, unlike their non-spiking counterparts, consist of a temporal dimension. Along with considering temporal information, a spatial and temporal pruning of SNNs is proposed in Chowdhury et al. (2021). Generally speaking, pruning will cause accuracy degradation to some extent. To avoid this, SD-SNN (Han et al., 2022) and Grad R (Chen et al., 2021) proposed the pruning-regeneration method for removing the redundancy in SNNs from the brain development plasticity mechanism. With synaptic regeneration, these works can effectively prevent and repair over-pruning. Recently, an interesting temporal pruning, which is specific for SNNs, was proposed in Chowdhury et al. (2022). This method starts with an SNN of T timesteps and reduces T every iteration of training, which results in a continuum of accurate and efficient SNNs from T timesteps, down to 1 timestep.

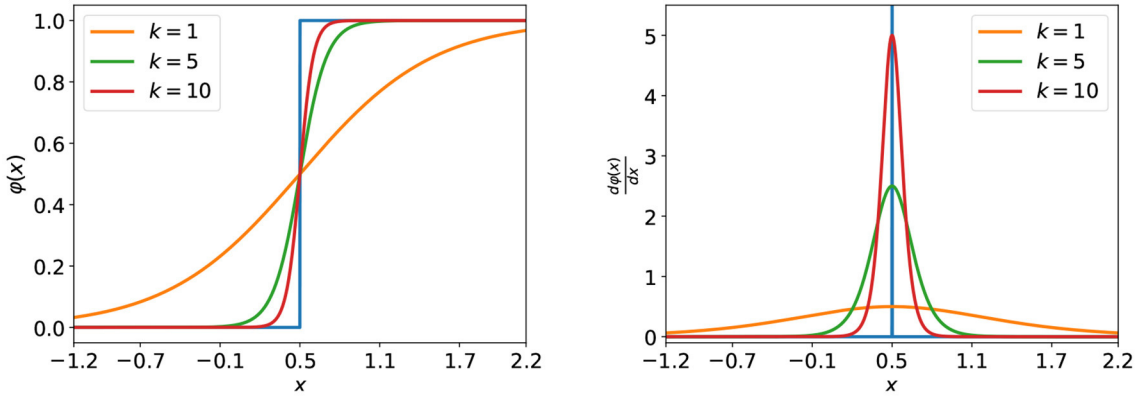


FIGURE 2
The approximation function **(left)** under different values of the coefficient, k and its corresponding gradient **(right)**. The blue curves represent the firing function **(left)** and its true gradient **(right)**.

3.2.1.2. Neural architecture searching

Obviously, a compact network carefully designed can reduce the storage and computation complexity of SNNs. However, due to the limitations of humans' inherent knowledge, it is difficult for people to jump out of their original thinking paradigm and design an optimal compact model. Therefore, there are some works using NAS techniques to let the algorithm automatically design the compact neural architecture (Kim et al., 2022a; Na et al., 2022). Furthermore, in Kim et al. (2022b), the lottery ticket hypothesis was investigated which shows that dense SNN networks contain smaller SNN subnetworks, i.e., winning tickets, which can achieve comparable performance to the dense ones, and the smaller compact one is picked as to be used network.

3.2.1.3. Knowledge distillation

The knowledge distillation methods aim at obtaining a compact model from a large model. In Kushawaha et al. (2021), a larger teacher SNN model is used to distill a smaller SNN model. And in Yang et al. (2022), Takuya et al. (2021), and Xu et al. (2023a,b), the same architecture ANN-teacher is used to distill SNN-student.

3.2.2. Sparse SNNs

Different from ANNs, SNNs transmit information by spike events, and the computation occurs only when the neuron receives spike events. Benefitting from this event-driven computation mechanism, SNNs can greatly save energy and run efficiently when implemented on neuromorphic hardware. Hence, limiting the firing rate of spiking neurons to achieve a sparse SNN is also a widely used way to improve the efficiency of the SNN. These kinds of methods can limit the firing rate of the SNN on both the neuron level and training technique level.

3.2.2.1. On the neuron level

In ASNN (Zambrano and Bohte, 2016), an adaptive SNN based on a group of adaptive spiking neurons was proposed. These adaptive spiking neurons can optimize their firing rate using asynchronous pulsed Sigma-Delta coding efficiently.

3.2.2.2. On the training technique level

In Han and Lee (2022), a correlation-based regularizer, which is incorporated into a loss function, was proposed to minimize the redundancies between the features at each layer for structural sparsity. Obviously, this method is beneficial for energy-efficient. Superspike (Zenke and Ganguli, 2018) added a heterosynaptic regularization term to the learning rule of the hidden layer weights to avoid pathologically high firing rates. RecDis-SNN (Guo et al., 2022c) incorporated a membrane potential loss into the SNN to regulate the membrane potential distribution to an appropriate range to avoid high firing rates. In Pellegrini et al. (2021), to enforce sparse spiking activity, a l_1 or l_2 regularization on the total number of spikes emitted by each layer was applied.

3.3. Temporal dynamics utilization methods

Different from ANNs, SNNs enjoy rich temporal dynamics characteristics, which makes them more suitable for some particular temporal tasks and some vision sensors with high resolution in time, e.g., neuromorphic cameras, which can capture temporally rich information asynchronously inspired by the information process form of eyes. Given such characteristics, a great number of methods falling in sequential learning and cooperating with neuromorphic cameras have been proposed for SNNs.

3.3.1. Sequential learning

As aforementioned in Section 2, SNNs maintain a dynamic state in the neuron memory. In Ponghiran and Roy (2022), the usefulness of the inherent recurrence dynamics of the SNN for sequential learning was demonstrated, that it can retain important information. Thus, SNNs show better performance on sequential learning compared to ANNs with similar scales in many works. In She et al. (2021), a function approximation theoretical basis was developed that any spike-sequence-to-spike-sequence mapping functions can be approximated by an SNN with one neuron per

layer using skip-layer connections. And then, based on the basis, a suitable SNN model for the classification of spatio-temporal data was designed. In Li Y. et al. (2022), SNNs were leveraged to study the Human Activity Recognition (HAR) task. Since SNNs allow spatio-temporal extraction of features and enjoy low-power computation with binary spikes, they can reduce up to 94% energy consumption while achieving better accuracy compared with homogeneous ANN counterparts. In Nomura et al. (2022), an interesting phenomenon was found that SNNs trained with the appropriate temporal penalty settings are more robust against adversarial images than ANNs.

As the common sequential signal, many preliminary works on speech recognition systems based on spiking neural networks have been explored (Tavanaei and Maida, 2017a,b; Wu et al., 2018a,b, 2019b, 2020; Zhang et al., 2019; Hao et al., 2020). In Wu et al. (2020), a deep spiking neural network was trained by the tandem learning method to handle the large vocabulary automatic speech recognition task. The experimental results demonstrated that the deep SNN trained could compete with its ANN counterpart while requiring as low as 0.68 times total synaptic operations to their ANN counterparts. There are also some works training deep SNN directly with SG methods for the speech task. In Ponghiran and Roy (2022), inspired by the LSTM, a custom version of SNNs was defined that combines a forget gate with multi-bit outputs instead of binary spikes, yielding better accuracy than that of LSTMs, but with 2× fewer parameters. In Bittar and Garner (2022b), the spiking neural networks trained like recurrent neural networks only using the standard surrogate gradient method can achieve promising results on speech recognition tasks, which shows the advantage of SNNs to handle this kind of task. In Bittar and Garner (2022a), a combination of adaptation, recurrence, and surrogate gradient techniques for spiking neural networks was proposed. And with these improvements, light spiking architectures that are not only able to compete with ANN solutions but also retain a high degree of compatibility with them were yielded. In Pellegrini et al. (2021), the dilated convolution spiking layers and a new regularization term to penalize the averaged number of spikes were used to train low-activity supervised convolutional spiking neural networks. The results showed that the SNN models can reach an error rate very close to standard DNNs while very energy efficient for speech tasks. In Sadovsky et al. (2023), a new technique for speech recognition that combines convolutional neural networks with spiking neural networks was presented to create an SNNCNN model. The results showed that the combination of CNNs and SNNs outperforms both MLPs and ANNs, providing a new route to further improvements in the field. In Yin et al. (2021), an activity-regularizing surrogate gradient method combined with recurrent networks of tunable and adaptive spiking neurons for SNNs was proposed, and the method performed well on the speech recognition task.

3.3.2. Cooperating with neuromorphic cameras

Neuromorphic camera, which is also called event-based cameras, have recently shown great potential for high-speed motion estimation owing to their ability to capture temporally rich information asynchronously. SNNs, with their spatio-temporal and event-driven processing mechanisms, are very suitable for handling such asynchronous data. Many excellent works combine SNNs and neuromorphic cameras to solve real-world large-scale problems. In Hagenaars et al. (2021) and Kosta and Roy (2022), an event-based optical flow estimation method was presented. In StereoSpike (Rançon et al., 2021) a depth estimation method was provided. SuperFast (Gao et al., 2022) leveraged an SNN and an event camera to present an event-enhanced high-speed video frame interpolation method. SuperFast can generate a very high frame rate (up to 5,000 FPS) video from the input low frame rate (25 FPS) video. Furthermore, Based on a hybrid network composed of SNNs and ANNs, E-SAI (Yu L. et al., 2022) provided a novel synthetic aperture imaging method, which can see through dense occlusions and extreme lighting conditions from event data. And in EVSNN (Zhu L. et al., 2022) a novel Event-based Video reconstruction framework was proposed. To fully use the information from different modalities, HALSIE (Biswas et al., 2022) proposed a hybrid approach for semantic segmentation comprised of dual encoders with an SNN branch to provide rich temporal cues from asynchronous events, and an ANN branch for extracting spatial information from regular frame data by simultaneously leveraging image and event modalities.

There are also some works that apply this technique in autonomous driving. In Cordone et al. (2022), fast and efficient automotive object detection with spiking neural networks on automotive event data was proposed. In Zhang J. et al. (2022), a spiking transformer network, STNet, which can dynamically extract and fuse information from both temporal and spatial domains was proposed for single object tracking using event data. Besides, since event cameras enjoy extremely low latency and high dynamic range, they can also be used to handle the harsh environment, i.e., extreme lighting conditions or dense occlusions. LaneSNNs (Viale et al., 2022) presented an SNN-based approach for detecting the lanes marked on the streets using the event-based camera input. The experimental results show a very low power consumption of about 1 W, which can significantly increase the lifetime and autonomy of battery-driven systems.

Based on the event-based cameras and SNNs, some works attempted to assist the behavioral recognition research. For examples, Spiking-Fer (Barchid et al., 2023) proposed a new end-to-end deep convolutional SNN method to predict facial expression. SpikeMS (Parameshwara et al., 2021) proposed a deep encoder-decoder SNN architecture and a novel spatio-temporal loss for motion segmentation using the event-based DVS camera as input. In Zou et al. (2023), a dedicated end-to-end sparse deep SNN consisting of the Spike-Element-Wise (SEW) ResNet and a novel Spiking Spatiotemporal Transformer was proposed for event-based pose tracking. This method achieves a significant computation reduction of 80% in FLOPS, demonstrating the superior advantage of SNN in this kind of task.

4. Future trends and conclusions

The spiking neural networks, born in mimicking the information process of brain neurons, enjoy many specific

characteristics and show great potential in many tasks, but meanwhile suffer from many weaknesses. As a consequence, a number of direct learning-based deep SNN solutions for handling these disadvantages or utilizing the advantages of SNNs have been proposed recently. As we summarized in this survey, these methods can be roughly categorized into (i) accuracy improvement methods, (ii) efficiency improvement methods, and (iii) temporal dynamics utilization methods. Though successful milestones and progress have been achieved through these works, there are still many challenges in the field.

On the accuracy improvement aspect, the SNN still faces serious performance loss, especially for the large network and datasets. The main reasons might include:

- *Lack of measurement of information capacity:* it is still unclear how to precisely calculate the information capacity of the spike maps and what kind of neuron types or network topology is suitable for preserving information while the information passing through the network, even after firing function. We believe SNN neurons and architectures should not be referenced from brains or ANNs completely. Specific designs in regard to the characteristic of SNNs for preserving information should be explored. For instance, to increase the spiking neuron representative ability, the binary spike {0, 1}, which is used to mimic the activation or silence in the brain, can be replaced by ternary spike {-1, 0, 1}, thus the information capacity of the spiking neuron will be boosted, but the event-driven and multiplication-free operation advantages of the binary spike can be preserved still. And as aforementioned, the widely used standard ResNet backbone in ANNs is not suitable for SNNs. And the PreAct ResNet backbone performs better since the membrane potential in neurons before the firing function will be added to the next block, thus the complete information will be transmitted simultaneously. While for the standard ResNet backbone, only quantized information is transmitted. To further preserve the information, adding the shortcut layer by layer in the PreAct ResNet backbone is better in our experiment, which is much different from the architectures in ANNs and is a promising exploration direction.
- *Inherent optimization difficulties:* It is still a difficult problem to optimize the SNN in a discrete space, even though many novel gradient estimators or approximate functions have been proposed, there are still some huge obstacles in the field. Such as the gradient explosion/vanishing problem, with the increasing timestep, the problem along with the gradient errors will become severer and make the network hard to converge. Thus, how to completely eliminate the impact of this problem to directly train an SNN with large timesteps is still under exploration. We believe more theoretical studies and practical tricks will emerge to answer this question in the future.

It is also worth noting that accuracy is not the only criterion of SNNs, the versatility is another key criterion, that measures whether a method can be used in practice. Some methods proposed in prior works are very versatile, such as learnable spike factors proposed in Real Spike (Guo et al., 2022d), membrane potential rectifier proposed in InfLoR-SNN (Guo et al., 2022b), temporal regularization loss proposed in TET (Deng et al., 2022), etc. These methods enjoy simple implementation and low coupling, thus having become common widely used practices to improve the accuracy of SNNs. Some methods improve the accuracy of SNNs by designing complex spiking neurons or specific architectures. Such improvements usually show a stronger ability to increase performance. However, as we have pointed out before, some of them suffer complicated computation and even lose the energy-efficiency advantage, which violates the original intention of SNNs. Therefore, purely pursuing high accuracy without considering versatility has limited significance in practice. The balance between accuracy and versatility is also an essential criterion for SNN research that should be considered in the following works.

On the efficiency improvement aspect, some prior works ignore the important fact, that the event-driven paradigm and friendly to the neuromorphic hardware make SNNs much different from ANNs. When implemented on the neuromorphic hardware, the computation in the SNN occurs only if the spiking neuron receives spike events. Hence, the direct reason for improving the efficiency of the SNN is reducing the the number of the firing spikes, not reducing network size. Some methods intending to improve the efficiency of SNNs by pruning inactive neurons as doing in ANNs can not make sense. We even think that under the condition the SNN network size does not exceed the capacity of the neuromorphic hardware, enlarging the network size but limiting the number of the firing spikes at the same time may be a potential route to improve the accuracy and efficiency simultaneously. In this way, different weights of the SNN may respond to different data, thus being equivalent to improving the representative capabilities of the SNN. However, a more systematic study needs to be done in the future.

On the temporal dynamics utilization aspect, a great number of interesting methods have been proposed and shown wide success. We think it is a very potential direction in the SNN field. Some explainable machine learning-related study indicates that different network types follow different patterns and enjoy different advantages. In this sense, it might be more meaningful to dive into the temporal dynamics of the SNN deeply, but not to pursue higher accuracy as ANNs. Meanwhile, considering the respective advantages, to use ANNs and SNNs together needs to be studied further.

Last but not least, more special applications for SNNs also should be explored still. Though SNNs have been used widely in many fields, including the neuromorphic camera, HAR task, speech recognition, autonomous driving, etc., as aforementioned and the object detection (Kim et al., 2020; Zhou et al., 2020), object tracking (Luo et al., 2020), image segmentation (Patel et al., 2021), robotic (Stagsted et al., 2020; Dupeyroux et al., 2021), etc., where some remarkable studies have applied SNNs on recently, compared to ANNs, their real-world applications are still very limited. Considering the unique advantage, efficiency of SNNs, we think there is a great opportunity for applying SNNs in the Green Artificial Intelligence (GAI), which has become an important subfield of Artificial Intelligence and has notable practical value.

We believe many studies focusing on using SNNs for GAI will emerge soon.

Author contributions

YG and XH wrote the paper with ZM being active contributors toward editing and revising the paper as well as supervising the project. All authors contributed to the article and approved the submitted version.

Funding

This work was supported by grants from the National Natural Science Foundation of China under contract Nos. 12202412 and 12202413.

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

Barchid, S., Allaert, B., Aissaoui, A., Mennesson, J., and Djéraba, C. (2023). Spiking-FER: spiking neural network for facial expression recognition with event cameras. *arXiv preprint arXiv:2304.10211*.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems, Vol. 31*. (Montréal, QC: Palais des Congrés de Montréal).

Biswas, S. D., Kosta, A., Liyanagedera, C., Apolinario, M., and Roy, K. (2022). Halsie-hybrid approach to learning segmentation by simultaneously exploiting image and event modalities. *arXiv preprint arXiv:2211.10754*.

Bittar, A., and Garner, P. (2022a). A surrogate gradient spiking baseline for speech command recognition. *Front. Neurosci.* 16:865897. doi: 10.3389/fnins.2022.865897

Bittar, A., and Garner, P. N. (2022b). Surrogate gradient spiking neural networks as encoders for large vocabulary continuous speech recognition. *arXiv preprint arXiv:2212.01187*.

Bohte, S. M. (2011). "Error-backpropagation in networks of fractionally predictive spiking neurons," in *International Conference on Artificial Neural Networks* (Espoo: Springer), 60–68. doi: 10.1007/978-3-642-21735-7_8

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Booij, O., and tat Nguyen, H. (2005). A gradient descent rule for spiking neurons emitting multiple spikes. *Inform. Process. Lett.* 95, 552–558. doi: 10.1016/j.ipl.2005.05.023

Bu, T., Ding, J., Yu, Z., and Huang, T. (2022). "Optimized potential initialization for low-latency spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36* (Columbia), 11–20. doi: 10.1609/aaai.v36i1.19874

Bu, T., Fang, W., Ding, J., Dai, P., Yu, Z., and Huang, T. (2023). Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. *arXiv preprint arXiv:2303.04347*.

Chen, Y., Yu, Z., Fang, W., Huang, T., and Tian, Y. (2021). Pruning of deep spiking neural networks through gradient rewiring. *arXiv preprint arXiv:2105.04916*. doi: 10.24963/ijcai.2021/236

Chen, Y., Zhang, S., Ren, S., and Qu, H. (2022). "Gradual surrogate gradient learning in deep spiking neural networks," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8927–8931. doi: 10.1109/ICASSP43922.2022.9746774

Cheng, X., Hao, Y., Xu, J., and Xu, B. (2020). "LISNN: improving spiking neural networks with lateral interactions for robust object recognition," in *IJCAI*, 1519–1525. doi: 10.24963/ijcai.2020/211

Chowdhury, S. S., Garg, I., and Roy, K. (2021). "Spatio-temporal pruning and quantization for low-latency spiking neural networks," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–9. doi: 10.1109/IJCNN52387.2021.9534111

Chowdhury, S. S., Rathi, N., and Roy, K. (2022). "Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning," in *Computer Vision-ECCV 2022*, eds S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner (Cham: Springer Nature Switzerland), 709–726. doi: 10.1007/978-3-031-20083-0_42

Cordone, L., Miramond, B., and Thierion, P. (2022). Object detection with spiking neural networks on automotive event data. *arXiv preprint arXiv:2205.04339*. doi: 10.1109/IJCNN55064.2022.9892618

Deng, S., Li, Y., Zhang, S., and Gu, S. (2022). Temporal efficient training of spiking neural network via gradient re-weighting. *arXiv preprint arXiv:2202.11946*.

Ding, J., Dong, B., Heide, F., Ding, Y., Zhou, Y., Yin, B., et al. (2022). "Biologically inspired dynamic thresholds for spiking neural networks," in *Advances in Neural Information Processing Systems*, eds A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho (New Orleans, LA).

Duan, C., Ding, J., Chen, S., Yu, Z., and Huang, T. (2022). "Temporal effective batch normalization in spiking neural networks,;; in *Advances in Neural Information Processing Systems*, eds A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho.

Dupeyroux, J., Hagenaars, J. J., Paredes-Vallés, F., and de Croon, G. C. (2021). "Neuromorphic control for optic-flow-based landing of MAVs using the loihi processor," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 96–102. doi: 10.1109/ICRA48506.2021.9560937

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. (2021a). Deep residual learning in spiking neural networks. *Adv. Neural Inform. Process. Syst.* 34, 21056–21069. doi: 10.48550/arXiv.2102.04159

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021b). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2661–2671. doi: 10.1109/ICCV48922.2021.00266

Feng, L., Liu, Q., Tang, H., Ma, D., and Pan, G. (2022). "Multi-level firing with spiking DS-ResNet: enabling better and deeper directly-trained spiking neural networks," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, ed L. D. Raedt (Vienna), 2471–2477. ijcai.org. doi: 10.24963/ijcai.2022/343

Gao, Y., Li, S., Li, Y., Guo, Y., and Dai, Q. (2022). Superfast: 200x video frame interpolation via event camera. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 7764–7780. doi: 10.1109/TPAMI.2022.3224051

Guo, Y., Chen, Y., Zhang, L., Liu, X., Wang, Y., Huang, X., et al. (2022a). "IM-loss: information maximization loss for spiking neural networks," in *Advances in Neural Information Processing Systems*, eds A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho (New Orleans, LA).

Guo, Y., Chen, Y., Zhang, L., Wang, Y., Liu, X., Tong, X., et al. (2022b). "Reducing information loss for spiking neural networks," in *Computer Vision-ECCV 2022*, eds S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner (Cham: Springer Nature Switzerland), 36–52. doi: 10.1007/978-3-031-20083-0_3

Guo, Y., Peng, W., Chen, Y., Zhang, L., Liu, X., Huang, X., et al. (2023). Joint a-SNN: joint training of artificial and spiking neural networks via self-distillation and weight factorization. *Pattern Recogn.* 2023:109639. doi: 10.1016/j.patcog.2023.109639

Guo, Y., Tong, X., Chen, Y., Zhang, L., Liu, X., Ma, Z., et al. (2022c). "RECDIS-SNN: rectifying membrane potential distribution for directly training spiking neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 326–335. doi: 10.1109/CVPR52688.2022.00042

Guo, Y., Zhang, L., Chen, Y., Tong, X., Liu, X., Wang, Y., et al. (2022d). "Real spike: learning real-valued spikes for spiking neural networks," in *Computer*

Vision-ECCV 2022: 17th European Conference (Tel Aviv: Springer), 52–68. doi: 10.1007/978-3-031-19775-8_4

Hagenaars, J., Paredes-Vallés, F., and De Croon, G. (2021). Self-supervised learning of event-based optical flow with spiking neural networks. *Adv. Neural Inform. Process. Syst.* 34, 7167–7179. doi: 10.48550/arXiv.2106.01862

Han, B., and Roy, K. (2020). "Deep spiking neural network: energy efficiency through time based coding," in *European Conference on Computer Vision* (Glasgow: Springer), 388–404. doi: 10.1007/978-3-030-58607-2_23

Han, B., Zhao, F., Zeng, Y., and Pan, W. (2022). Adaptive sparse structure development with pruning and regeneration for spiking neural networks. *arXiv preprint arXiv:2211.12219*. doi: 10.48550/arXiv.2211.12219

Han, C. S., and Lee, K. M. (2022). "Correlation-based regularization for fast and energy-efficient spiking neural networks," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, SAC '22* (New York, NY: Association for Computing Machinery), 1048–1055. doi: 10.1145/3477314.3507085

Hao, Y., Huang, X., Dong, M., and Xu, B. (2020). A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule. *Neural Netw.* 121, 387–395. doi: 10.1016/j.neunet.2019.09.007

Hong, C., Wei, X., Wang, J., Deng, B., Yu, H., and Che, Y. (2019). Training spiking neural networks for cognitive tasks: a versatile framework compatible with various temporal codes. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 1285–1296. doi: 10.1109/TNNLS.2019.2919662

Hu, Y., Wu, Y., Deng, L., and Li, G. (2021). Advancing residual learning towards powerful deep spiking neural networks. *arXiv preprint arXiv:2112.08954*.

Ikegawa, S.-I., Saiin, R., Sawada, Y., and Natori, N. (2022). Rethinking the role of normalization and residual blocks for spiking neural networks. *Sensors* 22:2876. doi: 10.3390/s22082876

Kim, S., Park, S., Na, B., and Yoon, S. (2020). "Spiking-yolo: spiking neural network for energy-efficient object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34*, 11270–11277. doi: 10.1609/aaai.v34i07.6787

Kim, Y., Li, Y., Park, H., Venkatesha, Y., and Panda, P. (2022a). "Neural architecture search for spiking neural networks," in *Computer Vision-ECCV 2022: 17th European Conference* (Tel Aviv: Springer), 36–56. doi: 10.1007/978-3-031-20053-3_3

Kim, Y., Li, Y., Park, H., Venkatesha, Y., Yin, R., and Panda, P. (2022b). "Exploring lottery ticket hypothesis in spiking neural networks," in *European Conference on Computer Vision* (Springer), 102–120. doi: 10.1007/978-3-031-19775-8_7

Kim, Y., and Panda, P. (2021). Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Front. Neurosci.* 2021:1638. doi: 10.3389/fnins.2021.773954

Kosta, A. K., and Roy, K. (2022). Adaptive-spikeNet: event-based optical flow estimation using spiking neural networks with learnable neuronal dynamics. *arXiv preprint arXiv:2209.11741*. doi: 10.48550/arXiv.2209.11741

Kushawaha, R. K., Kumar, S., Banerjee, B., and Velmurugan, R. (2021). "Distilling spikes: Knowledge distillation in spiking neural networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 4536–4543. doi: 10.1109/ICPR48806.2021.9412147

Leng, L., Che, K., Zhang, K., Zhang, J., Meng, Q., Cheng, J., et al. (2022). "Differentiable hierarchical and surrogate gradient search for spiking neural networks," in *Advances in Neural Information Processing Systems*, eds A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho (New Orleans, LA).

Li, W., Chen, H., Guo, J., Zhang, Z., and Wang, Y. (2022). "Brain-inspired multilayer perceptron with spiking neurons," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 783–793. doi: 10.1109/CVPR52688.2022.00086

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. (2021a). "A free lunch from ANN: towards efficient, accurate spiking neural networks calibration," in *International Conference on Machine Learning*, 6316–6325.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., and Gu, S. (2021b). Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Adv. Neural Inform. Process. Syst.* 34, 23426–23439.

Li, Y., Yin, R., Park, H., Kim, Y., and Panda, P. (2022). Wearable-based human activity recognition with spatio-temporal spiking neural networks. *arXiv preprint arXiv:2212.02233*. doi: 10.48550/arXiv.2212.02233

Li, Y., and Zeng, Y. (2022). Efficient and accurate conversion of spiking neural network with burst spikes. *arXiv preprint arXiv:2204.13271*. doi: 10.24963/ijcai.2022/345

Liu, F., Zhao, W., Chen, Y., Wang, Z., and Jiang, L. (2022). "Spikeconverter: an efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36* (Columbia), 1692–1701. doi: 10.1609/aaai.v36i2.20061

Lobov, S. A., Mikhaylov, A. N., Shamshin, M., Makarov, V. A., and Kazantsev, V. B. (2020). Spatial properties of STDP in a self-learning spiking neural network enable controlling a mobile robot. *Front. Neurosci.* 14:88. doi: 10.3389/fnins.2020.00088

Luo, X., Qu, H., Wang, Y., Yi, Z., Zhang, J., and Zhang, M. (2022). Supervised learning in multilayer spiking neural networks with spike temporal error backpropagation. *IEEE Trans. Neural Netw. Learn. Syst.* 1–13. doi: 10.1109/TNNLS.2022.3164930

Luo, Y., Xu, M., Yuan, C., Cao, X., Xu, Y., Wang, T., et al. (2020). SiamSNN: spike-based siamese network for energy-efficient and real-time object tracking. *arXiv preprint arXiv:2003.07584*. doi: 10.1007/978-3-030-86383-8_15

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., and Luo, Z.-Q. (2022). "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12444–12453. doi: 10.1109/CVPR52688.2022.01212

Na, B., Mok, J., Park, S., Lee, D., Choe, H., and Yoon, S. (2022). AutoSNN: towards energy-efficient spiking neural networks. *arXiv preprint arXiv:2201.12738*. doi: 10.48550/arXiv.2201.12738

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

Nomura, O., Sakemi, Y., Hosomi, T., and Morie, T. (2022). Robustness of spiking neural networks based on time-to-first-spike encoding against adversarial attacks. *IEEE Trans. Circuits Syst. II* 69, 3640–3644. doi: 10.1109/TCSII.2022.3184313

Parameshwara, C. M., Li, S., Fermüller, C., Sanket, N. J., Evanusa, M. S., and Aloimonos, Y. (2021). "Spikems: deep spiking neural network for motion segmentation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3414–3420. doi: 10.1109/IROS51168.2021.9636506

Patel, K., Hunsberger, E., Batir, S., and Eliasmith, C. (2021). A spiking neural network for image segmentation. *arXiv preprint arXiv:2106.08921*. doi: 10.48550/arXiv.2106.08921

Pellegrini, T., Zimmer, R., and Masquelier, T. (2021). "Low-activity supervised convolutional spiking neural networks applied to speech commands recognition," in *2021 IEEE Spoken Language Technology Workshop (SLT)*, 97–103. doi: 10.1109/SLT48900.2021.9383587

Ponghiran, W., and Roy, K. (2022). "Spiking neural networks with improved inherent recurrence dynamics for sequential learning," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36* (Columbia), 8001–8008. doi: 10.1609/aaai.v36i7.20771

Ponulak, F., and Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. *Acta Neurobiol. Exp.* 71, 409–433.

Rançon, U., Cuadrado-Anibarro, J., Cottereau, B. R., and Masquelier, T. (2021). Stereospike: depth learning with a spiking neural network. *arXiv preprint arXiv:2109.13751*.

Rathi, N., and Roy, K. (2020). Diet-SNN: direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*.

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Sadovsky, E., Jakubec, M., and Jarina, R. (2023). "Speech command recognition based on convolutional spiking neural networks," in *2023 33rd International Conference Radioelektronika (RADIOELEKTRONIKA)* (Pardubice), 1–5. doi: 10.1109/RADIOELEKTRONIKA57919.2023.10109082

She, X., Dash, S., and Mukhopadhyay, S. (2021). "Sequence approximation using feedforward spiking neural network for spatiotemporal learning: theory and optimization methods," in *International Conference on Learning Representations*.

Shen, G., Zhao, D., and Zeng, Y. (2023). Exploiting high performance spiking neural networks with efficient spiking patterns. *arXiv preprint arXiv:2301.12356*. doi: 10.48550/arXiv.2301.12356

Stagsted, R., Vitale, A., Binz, J., Renner, A., and Sandamirskaya, Y. (2020). "Towards neuromorphic control: a spiking neural network based PID controller for UAV," in *Robotics: Science and Systems 2020* (Corvallis, OR: Oregon State University). doi: 10.15607/RSS.2020.XVI.074

Takuya, S., Zhang, R., and Nakashima, Y. (2021). "Training low-latency spiking neural network through knowledge distillation," in *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, 1–3. doi: 10.1109/COOLCHIPS52128.2021.9410323

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Tavanaei, A., and Maida, A. (2017a). "Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals," in *Neural Information Processing*, eds D. Liu, S. Xie, Y. Li, D. Zhao, and E. S. M. El-Alfy (Cham: Springer International Publishing), 899–908. doi: 10.1007/978-3-319-70136-3_95

Tavanaei, A., and Maida, A. S. (2017b). A spiking network that learns to extract spike signatures from speech signals. *Neurocomputing* 240, 191–199. doi: 10.1016/j.neucom.2017.01.088

Viale, A., Marchisio, A., Martina, M., Masera, G., and Shafique, M. (2022). "LaneSNNs: spiking neural networks for lane detection on the loihi neuromorphic processor," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 79–86. doi: 10.1109/IROS47612.2022.9981034

Wang, S., Cheng, T. H., and Lim, M.-H. (2022). "LTMD: learning improvement of spiking neural networks with learnable thresholding neurons and moderate dropout," in *Advances in Neural Information Processing Systems*, eds A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho (New Orleans, LA).

Wang, X., Lin, X., and Dang, X. (2020). Supervised learning in spiking neural networks: a review of algorithms and evaluations. *Neural Netw.* 125, 258–280. doi: 10.1016/j.neunet.2020.02.011

Wang, X., Zhang, Y., and Zhang, Y. (2023). MT-SNN: enhance spiking neural network with multiple thresholds. *arXiv preprint arXiv:2303.11127*.

Wang, Y., Zhang, M., Chen, Y., and Qu, H. (2022). "Signed neuron with memory: towards simple, accurate and high-efficient ANN-SNN conversion," in *International Joint Conference on Artificial Intelligence*. doi: 10.24963/ijcai.2022/347

Wu, J., Chua, Y., and Li, H. (2018a). "A biologically plausible speech recognition framework based on spiking neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*. doi: 10.1109/IJCNN.2018.8489535

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2021a). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 34, 446–460. doi: 10.1109/TNNLS.2021.3095724

Wu, J., Chua, Y., Zhang, M., Li, H., and Tan, K. C. (2018b). A spiking neural network framework for robust sound classification. *Front. Neurosci.* 12:836. doi: 10.3389/fnins.2018.00836

Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., and Li, H. (2019a). "Deep spiking neural network with spike count based learning rule," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–6. doi: 10.1109/IJCNN.2019.8852380

Wu, J., Pan, Z., Zhang, M., Das, R. K., Chua, Y., and Li, H. (2019b). "Robust sound recognition: a neuromorphic approach," in *Interspeech*, 3667–3668.

Wu, J., Xu, C., Han, X., Zhou, D., Zhang, M., Li, H., et al. (2021b). Progressive tandem learning for pattern recognition with deep spiking neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 7824–7840. doi: 10.1109/TPAMI.2021.3114196

Wu, J., Yılmaz, E., Zhang, M., Li, H., and Tan, K. (2020). Deep spiking neural networks for large vocabulary automatic speech recognition. *Front. Neurosci.* 14:199. doi: 10.3389/fnins.2020.00199

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33*, 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Xu, Q., Li, Y., Fang, X., Shen, J., Liu, J. K., Tang, H., et al. (2023a). Biologically inspired structure learning with reverse knowledge distillation for spiking neural networks. *arXiv preprint arXiv:2304.09500*. doi: 10.48550/arXiv.2304.09500

Xu, Q., Li, Y., Shen, J., Liu, J. K., Tang, H., and Pan, G. (2023b). Constructing deep spiking neural networks from artificial neural networks with knowledge distillation. *arXiv preprint arXiv:2304.05627*. doi: 10.48550/arXiv.2304.05627

Xu, Y., Zeng, X., Han, L., and Yang, J. (2013). A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Netw.* 43, 99–113. doi: 10.1016/j.neunet.2013.02.003

Yamazaki, K., Vo-Ho, V.-K., Bulsara, D., and Le, N. (2022). Spiking neural networks and their applications: a review. *Brain Sci.* 12:863. doi: 10.3390/brainsci12070863

Yang, Q., Wu, J., Zhang, M., Chua, Y., Wang, X., and Li, H. (2022). Training spiking neural networks with local tandem learning. *arXiv preprint arXiv:2210.04532*. doi: 10.48550/arXiv.2210.04532

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., et al. (2021). "Temporal-wise attention spiking neural networks for event streams classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10221–10230. doi: 10.1109/ICCV48922.2021.01006

Yao, X., Li, F., Mo, Z., and Cheng, J. (2022). GLIF: a unified gated leaky integrate-and-fire neuron for spiking neural networks. in *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*. (New Orleans, LA).

Yin, B., Corradi, F., and Boht, S. (2021). Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat. Mach. Intell.* 3, 905–913. doi: 10.1038/s42256-021-00397-w

Yin, B., Corradi, F., and Bohté, S. M. (2020). "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, 1–8. doi: 10.1145/3407197.3407225

Yu, L., Zhang, X., Liao, W., Yang, W., and Xia, G.-S. (2022). Learning to see through with events. *IEEE Trans. Pattern Anal. Mach. Intell.* 1–18. doi: 10.1109/TPAMI.2022.3227448

Yu, Q., Gao, J., Wei, J., Li, J., Tan, K. C., and Huang, T. (2022a). Improving multispike learning with plastic synaptic delays. *IEEE Trans. Neural Netw. Learn. Syst.* 1–12. doi: 10.1109/TNNLS.2022.3165527

Yu, Q., Song, S., Ma, C., Pan, L., and Tan, K. C. (2022b). Synaptic learning with augmented spikes. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1134–1146. doi: 10.1109/TNNLS.2020.3040969

Zambrano, D., and Bohte, S. M. (2016). Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv preprint arXiv:1609.02053*. doi: 10.48550/arXiv.1609.02053

Zenke, F., and Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* 30, 1514–1541. doi: 10.1162/neco_a_01086

Zhang, D., Zhang, T., Jia, S., Wang, Q., and Xu, B. (2022). Recent advances and new frontiers in spiking neural networks. *arXiv preprint arXiv:2204.07050*. doi: 10.24963/ijcai.2022/790

Zhang, J., Dong, B., Zhang, H., Ding, J., Heide, F., Yin, B., et al. (2022). "Spiking transformers for event-based single object tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8801–8810. doi: 10.1109/CVPR52688.2022.00860

Zhang, M., Luo, X., Chen, Y., Wu, J., Belatreche, A., Pan, Z., et al. (2020a). An efficient threshold-driven aggregate-label learning algorithm for multimodal information processing. *IEEE J. Select. Top. Signal Process.* 14, 592–602. doi: 10.1109/JSTSP.2020.2983547

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2022). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1947–1958. doi: 10.1109/TNNLS.2021.3110991

Zhang, M., Wu, J., Belatreche, A., Pan, Z., Xie, X., Chua, Y., et al. (2020b). Supervised learning in spiking neural networks with synaptic delay-weight plasticity. *Neurocomputing* 409, 103–118. doi: 10.1016/j.neucom.2020.03.079

Zhang, M., Wu, J., Chua, Y., Luo, X., Pan, Z., Liu, D., et al. (2019). MPD-Al: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons. *Proc. AAAI Conf. Artif. Intell.* 33, 1327–1334. doi: 10.1609/aaai.v33i01.33011327

Zhang, W., and Li, P. (2020). Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Adv. Neural Inform. Process. Syst.* 33, 12022–12033.

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). "Going deeper with directly-trained larger spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35*, 11062–11070. doi: 10.1609/aaai.v35i12.17320

Zhou, S., Chen, Y., Li, X., and Sanyal, A. (2020). Deep SCNN-based real-time object detection for self-driving vehicles using lidar temporal data. *IEEE Access* 8, 76903–76912. doi: 10.1109/ACCESS.2020.2990416

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S. T., and Sanyal, A. (2021). "Temporal-coded deep spiking neural network with easy training and robust performance," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35*, 11143–11151. doi: 10.1609/aaai.v35i12.17329

Zhu, L., Wang, X., Chang, Y., Li, J., Huang, T., and Tian, Y. (2022). "Event-based video reconstruction via potential-assisted spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3594–3604. doi: 10.1109/CVPR52688.2022.00358

Zhu, R. -J., Zhao, Q., Zhang, T., Deng, H., Duan, Y., Zhang, M., et al. (2022). TCJA-SNN: Temporal-channel joint attention for spiking neural networks. *arXiv [Preprint]*. arXiv: 2206.10177. Available online at: https://arxiv.org/pdf/2206.10177.pdf

Zhu, Y., Yu, Z., Fang, W., Xie, X., Huang, T., and Masquelier, T. (2022). "Training spiking neural networks with event-driven backpropagation," in *36th Conference on Neural Information Processing Systems (NeurIPS 2022)* (New Orleans, LA).

Zimmer, R., Pellegrini, T., Singh, S. F., and Masquelier, T. (2019). Technical report: supervised training of convolutional spiking neural networks with pytorch. *arXiv preprint arXiv:1911.10124*.

Zou, S., Mu, Y., Zuo, X., Wang, S., and Cheng, L. (2023). Event-based human pose tracking by spiking spatiotemporal transformer. *arXiv preprint arXiv:2303.09681*. doi: 10.48550/arXiv.2303.09681

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 26 July 2023
DOI 10.3389/fnins.2023.1213720

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Guozhang Chen,
Graz University of Technology, Austria
Yishu Zhang,
Zhejiang University, China

*CORRESPONDENCE
Zhenzhi Wu
✉ zhenzhi.wu@lynxi.com
Yansong Chua
✉ caiyansong@cnaeit.com

RECEIVED 28 April 2023
ACCEPTED 22 June 2023
PUBLISHED 26 July 2023

CITATION
Wu Z, Shen Y, Zhang J, Liang H, Zhao R, Li H, Xiong J, Zhang X and Chua Y (2023) BIDL: a brain-inspired deep learning framework for spatiotemporal processing.
Front. Neurosci. 17:1213720.
doi: 10.3389/fnins.2023.1213720

BIDL: a brain-inspired deep learning framework for spatiotemporal processing

Zhenzhi Wu[1]*, Yangshu Shen[1,2], Jing Zhang[1], Huaju Liang[3], Rongzhen Zhao[1], Han Li[1], Jianping Xiong[2], Xiyu Zhang[4] and Yansong Chua[3]*

[1]Lynxi Technologies, Co. Ltd., Beijing, China, [2]Department of Precision Instruments and Mechanology, Tsinghua University, Beijing, China, [3]Neuromorphic Computing Laboratory, China Nanhu Academy of Electronics and Information Technology (CNAEIT), Jiaxing, Zhejiang, China, [4]School of Automation Science and Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi, China

Brain-inspired deep spiking neural network (DSNN) which emulates the function of the biological brain provides an effective approach for event-stream spatiotemporal perception (STP), especially for dynamic vision sensor (DVS) signals. However, there is a lack of generalized learning frameworks that can handle various spatiotemporal modalities beyond event-stream, such as video clips and 3D imaging data. To provide a unified design flow for generalized spatiotemporal processing (STP) and to investigate the capability of lightweight STP processing via brain-inspired neural dynamics, this study introduces a training platform called brain-inspired deep learning (BIDL). This framework constructs deep neural networks, which leverage neural dynamics for processing temporal information and ensures high-accuracy spatial processing via artificial neural network layers. We conducted experiments involving various types of data, including video information processing, DVS information processing, 3D medical imaging classification, and natural language processing. These experiments demonstrate the efficiency of the proposed method. Moreover, as a research framework for researchers in the fields of neuroscience and machine learning, BIDL facilitates the exploration of different neural models and enables global-local co-learning. For easily fitting to neuromorphic chips and GPUs, the framework incorporates several optimizations, including iteration representation, state-aware computational graph, and built-in neural functions. This study presents a user-friendly and efficient DSNN builder for lightweight STP applications and has the potential to drive future advancements in bio-inspired research.

KEYWORDS

spatiotemporal processing framework, spiking neural network, global-local co-learning, synaptic plasticity, video recognition, brain-inspired computing, leaky integrate and fire, reward-modulated STDP

1. Introduction

Humans can perceive the continuously changing world, including static features such as object shapes and colors, as well as dynamic features such as motion trajectories and waveforms. These perceptions require high processing precision and need to be performed in real time with low computational power requirements. It is worth noting that brain-inspired computing features rich neural dynamics for efficiently processing temporal signals (Carlos et al., 2019) and deep learning technologies provide the capability for high-precision spatial information processing, which leads to the widespread development of the deep spiking

neural networks (DSNNs) be deployed for spatiotemporal processing (Gu et al., 2019; Wu et al., 2021, 2022b). However, most of the current DSNN research focuses on dynamic vision sensor (DVS) processing or image recognition tasks. For most non-event-stream scenarios, a DSNN still lacks satisfactory accuracy. There are also many high-accuracy spatiotemporal networks, including two-stream networks (Simonyan and Zisserman, 2014a), convolutional 3D networks (Tran et al., 2015), and video transformer networks (Neimark et al., 2021) for video clip processing, networks (Han et al., 2019) for 3D imaging (image sequence) processing, long short-term memory (LSTM) (Greff et al., 2016), and transformer (Vaswani et al., 2017) networks for natural language processing (NLP) etc. The key issue is that the computational complexity and parameter size of these networks are much larger than pure spatial processing, which makes them hard for real-time high-throughput perception. Therefore, in this study, we aim to extend DSNN to lightweight generalized STP. The bio-inspired neurons are integrated into deep artificial neural networks (ANNs) and enable spatiotemporal processing with a computational complexity approaching pure spatial processing level, by introducing lightweight neural dynamics. It is revealed that temporal processing can be achieved with limited computational cost and memory footprint. We further verified that the accuracy can be improved by advanced neural models, especially for video clip processing. From this point, we develop a generalized spatiotemporal processing methodology via neural dynamics and deep neural networks (DNNs) and then designed a framework named brain-inspired deep learning (BIDL) that can adapt to a variety of modalities, such as video, DVS, text, sensor signals, etc., and finally achieves a real-time high accuracy processing. Therefore, this study extends the DSNN application domain to a much wider scope.

In another aspect, BIDL aims to provide a research platform for DSNNs. Existing frameworks mostly cater to either neuroscience researchers or machine learning researchers but not both. For computational neuroscience researchers, it is essential to have the flexibility to configure neural models, synaptic plasticity, and network structure in a research platform. Additionally, realistic applications are crucial for validating their ideas. From the perspective of neural simulators, there are already several simulators available, ranging from high-accuracy simulation (Carnevale and Hines, 2006) to large-scale simulation (Gewaltig and Diesmann, 2007). Some frameworks also accelerate network simulation using GPUs (Yavuz et al., 2016; Golosio et al., 2021). In terms of applications, Bekolay et al. (2014) and Wang et al. (2022) provided rich examples of bio-inspired networks and dynamics. BIDL offers similar design flexibility to these frameworks and enables vectorized acceleration for neuron populations and synapse connections using PyTorch. Furthermore, BIDL enables back propagation through time (BPTT) training for advanced neural models.

Machine learning researchers require a fast development platform for DSNNs with a deep neural network (DNN) design style. Support for global-local learning is essential for validating brain-inspired local learning with gradient-based global learning. Among the frameworks that fulfill these requirements, some frameworks such as Hazan et al. (2018), Rasmussen (2019), and Bohte et al. (2000) enable spiking neural network (SNN) designs using deep learning frameworks such as Jax, PyTorch, and TensorFlow. Among them, Fang et al. (2020) enabled BPTT learning, which achieves high accuracy. Inspired by these frameworks, BIDL provides a rapid development platform for designing and training DSNNs using a DNN-style approach. BIDL also introduces a configuration file that integrates networks with datasets and pre-processing pipelines, similar to OpenMMLab (MMCV-Contributors, 2018), enabling efficient DSNN design. Additionally, BIDL introduces a unified flow for global-local co-learning in DSNNs, including BPTT learning and programmable generalized synaptic plasticity. Therefore, BIDL serves as a unified research framework for both neuroscience and machine learning researchers.

Moreover, BIDL incorporates a range of optimizations to fit the designed network to neuromorphic chips, particularly focusing on the iteration representation. Since most neuromorphic chips operate in a timestep-driven manner, some frameworks are specifically designed for such cases (Gewaltig and Diesmann, 2007; Davison et al., 2009; Wang et al., 2022). However, for speeding up GPU processing, an iteration within each temporal layer proves to be a better choice, as applied in SpikingJelly (Fang et al., 2020). Consequently, BIDL supports both internal and external iteration methods, allowing training within a single framework while utilizing the same design flow.

The main contributions of this study are as follows:

(1). BIDL provides a unified DSNN design flow for a range of STP applications, including video clip processing, moving trajectory processing, dynamic vision sensor (DVS) recognition, 3D medical imaging classification, and NLP task. These experiments demonstrate the efficiency of BIDL, achieving high accuracy while consuming significantly less computation than traditional convolutional 3D (Conv3D) or LSTM approaches. Finally, real-time processing of these experiments on embedded platform is realized.

(2). As a research framework for neuroscience and machine learning researchers, BIDL facilitates the exploration of various differentiable neural models such as the leaky integrate-and-fire (LIF) and LIF+ models. It also supports different neural configurations, including analog spike and residual membrane potential (RMP). Furthermore, BIDL enables global-local co-learning through the use of back-propagation through Time (BPTT) for global learning and generalized synaptic plasticity rules for local learning. To demonstrate the exploration capability of BIDL, we provide an anti-noise example utilizing BPTT and localized plasticity co-learning as well as a DVS recognition example employing various LIF+ neurons.

(3). To ensure compatibility with neuromorphic chips and GPUs, the BIDL framework incorporates both internal iteration and external iteration of timesteps into a unified design flow. Additionally, we propose a state-variable indicated computational graph as a representation of the networks, which facilitates seamless integration with downstream SNN compilers.

The article is organized as follows: Section 2 illustrates the generalized spatiotemporal processing network through neural dynamics. Section 3 discusses the characteristics of the BIDL research platform, catering to the needs of researchers in the field.

Section 4 presents optimizations and considerations for deploying BIDL on neuromorphic chips. Section 5 provides diverse examples of spatiotemporal processing, advanced neural models, and global-local co-learning. Section 6 presents a discussion on the design choices of the BIDL framework. Finally, Section 7 concludes the study.

2. Generalized spatiotemporal processing via neural dynamics

Generalized lightweight spatiotemporal processing requires several criteria. First, it needs to be a generalized method that can adapt to a variety of modalities, such as video, DVS, text, and sensor signals. Second, it should have high accuracy in processing both spatial and temporal information. Third, it should be lightweight in terms of computation and memory usage, enabling real-time low-latency processing. In this section, we will discuss how BIDL meets these requirements.

For a generalized spatiotemporal framework, the neural networks in BIDL treat all spatiotemporal source signals as spatiotemporal tensors. These tensors contain both spatial and temporal information, forming a spatio-temporal (ST) tensor with the shape $[B, T, C, H, W]$. In this study, B represents the batch size, T denotes the total timesteps, H and W represent height and width, respectively, and C denotes the number of channels. This spatiotemporal tensor format allows the representation of spatial information in $[C, H, W]$ and temporal information with T timesteps.

BIDL requires a data pre-processing procedure to convert the source data into the ST tensor format, as illustrated in Figure 1. For a video clip, the temporal tensor represents a sequence of frames. The video frames undergo a sampling rate conversion followed by image preprocessing to derive the temporal tensor. For DVS signals, the event format represented as (x, y, t, ps) over a time duration $t \in t_s$ is collected and forms a frame. In this study, x and y denote the pixel location, and two channels represent the increase and decrease of light intensity, respectively. Each pixel is represented by an integer value indicating the count of events at that location. To satisfy the spiking format for SNN, it can also be transformed into a binary format, where one or several neurons represent a pixel. For 3D imaging data, each slice of the 3D imaging can be treated as a "frame," allowing the 3D data direct transformation into a temporal tensor. For purely temporal signals, such as text or voice, they can be represented in the 5D format with $H = 1$ and $W = 1$. However, for easier understanding in most cases, we use a 3D tensor format with the shape $[B, T, S]$, where S represents the dimension of hidden states. Therefore, multiple modalities can be converted into a unified ST tensor format for neural network processing. BIDL also provides a set of data pre-processing pipelines for converting data formats into the unified ST format. It is important to note that this procedure is defined by the user, and the methods for processing new data sources may vary. Detailed information about the pre-processing for each experiment can be found in Section 5.

For high-accuracy spatiotemporal processing, BIDL utilizes a DSNN, where ANN layers and SNN layers are interleaved and stacked in a network. With the advantages of deep learning technologies, ANN layers, including convolution, batch normalization, pooling, dropout, linear, residual connections, and attention mechanisms, are proficient in spatial (image) processing, as shown in Figure 1. Additionally, the backbone network of DSNN can be directly adopted from DNN, such as ResNet (He et al., 2016) or VGG (Simonyan and Zisserman, 2014b). On the contrary, SNN layers, such as LIF, can be inserted into these DNNs to introduce temporal processing capability. Therefore, ANN layers are responsible for spatial processing, while SNN layers handle temporal processing. It has been demonstrated that the neural dynamics of these neural models can effectively extract temporal information (Wu et al., 2021, 2022a). In particular, the ConvLIF is an ST block that incorporates convolution, batch normalization, and LIF, making it suitable for lightweight spatiotemporal processing. The ConvLIAF block is an improved ST block that replaces spike activations with analog activations while maintaining the neural dynamics of LIF (Wu et al., 2021), thereby enhancing spatial signal transfer accuracy. These ConvLIF/ConvLIAF blocks can be considered as the fundamental building blocks for constructing spatiotemporal networks, as shown in Figure 2. We can also leverage basic backbone networks from DNN designs and insert neural models to enable temporal processing, such as ResNet-LIF or VGG-LIAF. These networks can be trained in BIDL using global learning methods, such as BPTT, or learned through local learning methods, such as Hebb or spike-timing-dependent plasticity (STDP). BPTT provides high-accuracy processing through supervised learning, while local methods offer unsupervised or weakly supervised learning, which can be used for adapting to new tasks or environments. The top-level architecture of the proposed framework is illustrated in Figure 3.

2.1. Definition of the LIF/LIAF neural model

The definition of LIF, Leaky Integrate and Analog Fire (LIAF), and Residual Membrane Potential (RMP) applied in this study has been previously illustrated in Han et al. (2020), Wu et al. (2021), and Wu et al. (2022b). For an easy understanding of the proposed framework, we reintroduced the definitions as follows.

The original LIF model is described in a differential function (Ferré et al., 2018; Roy et al., 2019) to reveal the neural dynamics, following the equation

$$\tau \frac{dV_j(t)}{dt} = -(V_j(t) - V_{rest}) + RI_j(t), \tag{1}$$

where j represents the neuron index. τ is the timing factor of the neuron, V_{rest} is the resting potential, and $I_j(t)$ is the input current. When $V_j(t)$ reaches a certain threshold V_{th}, a spike is emitted, and the $V_j(t)$ is reset to an initial value V_{reset}.

We introduce the Euler method (Wu et al., 2018; Neftci et al., 2019) on Eq. 1 to obtain an iterative representation in discrete-time for easy inference and training. We define Δt as the sampling duration which is a small fraction of time, with the Euler method, the equation can be solved numerically:

$$V_j(t) = V_j(t - \Delta t) + \frac{\Delta t}{\tau}\left(-V_j(t - \Delta t) + V_{rest} + RI_j(t - \Delta t)\right). \tag{2}$$

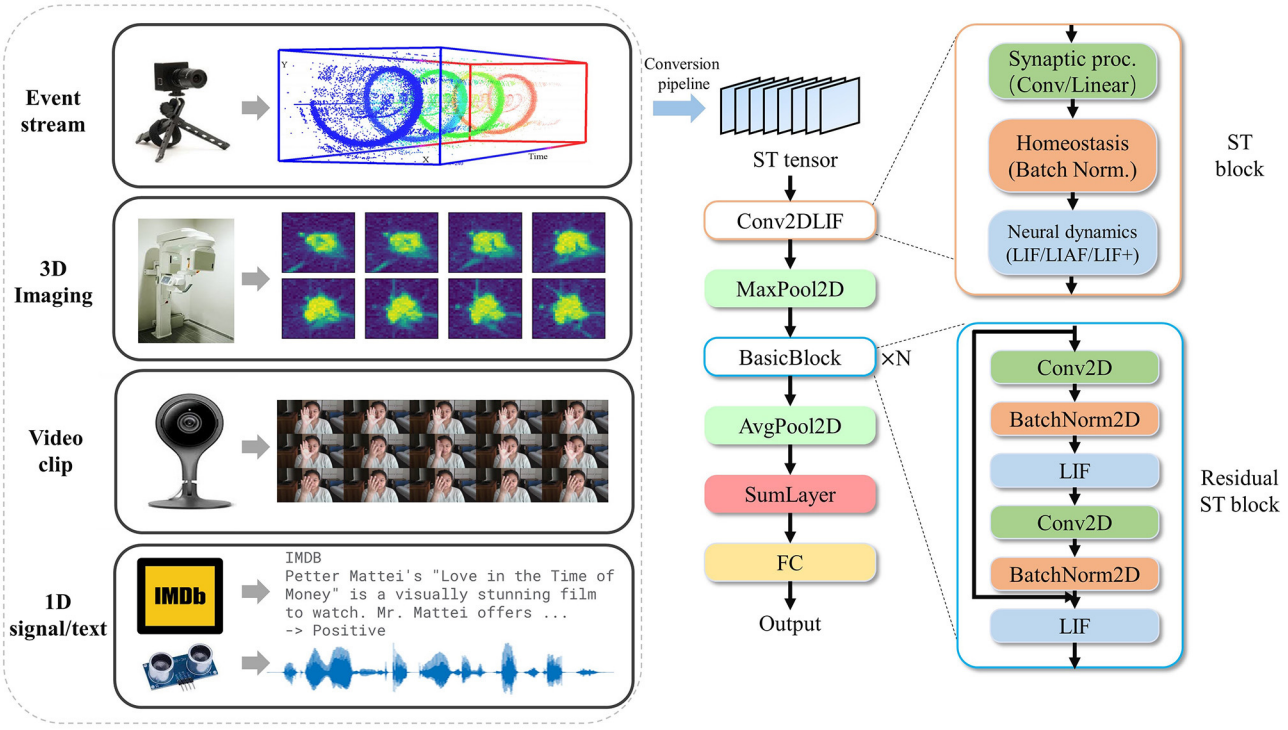


FIGURE 1
Proposed ST processing on various input modalities, including event-stream, video clip, 3D imaging, and 1D signals (including text). The data are first converted to an ST tensor via a conversion pipeline and then processed by a DSNN with ST blocks. The ST block consists of synaptic processing, such as convolution or linear layers, homeostasis, such as batch normalization, and neural dynamics, including LIF/LIAF/LIF+. The event-stream demonstration sub-figure is modified from Hinz et al. (2017).

After sampling with a sampling rate of $1/\Delta t$, we denote the timestep as t_n, where $t = t_n \Delta t$, then we have

$$V_j(t_n \Delta t) = V_j((t_n - 1)\Delta t)(1 - \frac{\Delta t}{\tau}) + \frac{\Delta t}{\tau} V_{rest} + \frac{\Delta t}{\tau} R I_j((t_n - 1)\Delta t). \tag{3}$$

For simplicity, we further define $\alpha = 1 - \frac{\Delta t}{\tau}$, $\beta = \frac{\Delta t}{\tau} V_{rest}$, and $r = \frac{R}{\alpha} \frac{\Delta t}{\tau}$, then Eq. 3 can be written as

$$V_j(t_n \Delta t) = \alpha(V_j((t_n - 1)\Delta t) + r I_j((t_n - 1)\Delta t)) + \beta \tag{4}$$

In the discrete form, we skip the notation Δt, therefore, we get

$$V_j^{t_n} = \alpha(V_j^{t_n - 1} + r I_j^{t_n - 1}) + \beta. \tag{5}$$

Therefore, we obtained the LIF representation in a discrete form. In the remaining part of this article, all the expressions are in a discrete form, and we use t instead of t_n to represent the timestep for simplicity.

We further pack a group of neurons in a tensor for a more compact expression, where $\boldsymbol{I}^t$, $\boldsymbol{V}^t$, $\boldsymbol{V}_{th}$, $\boldsymbol{V}_{reset}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{r}$ are all in a tensor format, and we have re-written LIF/LIAF in following calculation procedure:

(a). Accumulate input current with the previous membrane potential:

$$\boldsymbol{V}_m^t = \boldsymbol{V}^{t-1} + \boldsymbol{r} \cdot \boldsymbol{I}^t, \tag{6}$$

where $\boldsymbol{V}^{t-1}$ and $\boldsymbol{V}^t$ refer to the previous and current membrane potential, respectively. The input current usually comes from convolutional synaptic calculation or linear (fully connected) synaptic calculation.

(b). Compare with the threshold and fire:

$$\boldsymbol{F}^t = \boldsymbol{V}_m^t \geq \boldsymbol{V}_{th}, \tag{7}$$

where $\boldsymbol{F}^t$ is the fire signal. For each F_j^t in $\boldsymbol{F}^t$, $F_j^t = 1$ indicates a firing event; otherwise, $F_j^t = 0$.

(c). Reset the membrane potential when fired:

$$\boldsymbol{R}^t = \boldsymbol{F}^t \cdot \boldsymbol{V}_{reset} + (\boldsymbol{1} - \boldsymbol{F}^t) \cdot \boldsymbol{V}_m^t \tag{8}$$

When using residual membrane potential (RMP) (Han et al., 2020), a soft reset instead of a hard reset is applied:

$$\boldsymbol{R}^t = \boldsymbol{F}^t \cdot (\boldsymbol{V}_m^t - \boldsymbol{V}_{th}) + (\boldsymbol{1} - \boldsymbol{F}^t) \cdot \boldsymbol{V}_m^t \tag{9}$$

(d). Perform leakage:

$$\boldsymbol{V}^t = \boldsymbol{\alpha} \cdot \boldsymbol{R}^t + \boldsymbol{\beta} \tag{10}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ represent the multiplicative decay and additive decay, respectively.

(e). Output:

$$\boldsymbol{Y}^t = \begin{cases} \boldsymbol{F}^t, & \text{for LIF} \\ f(\boldsymbol{V}^t), & \text{for LIAF,} \end{cases} \tag{11}$$

where $f(x)$ is the analog activation function.

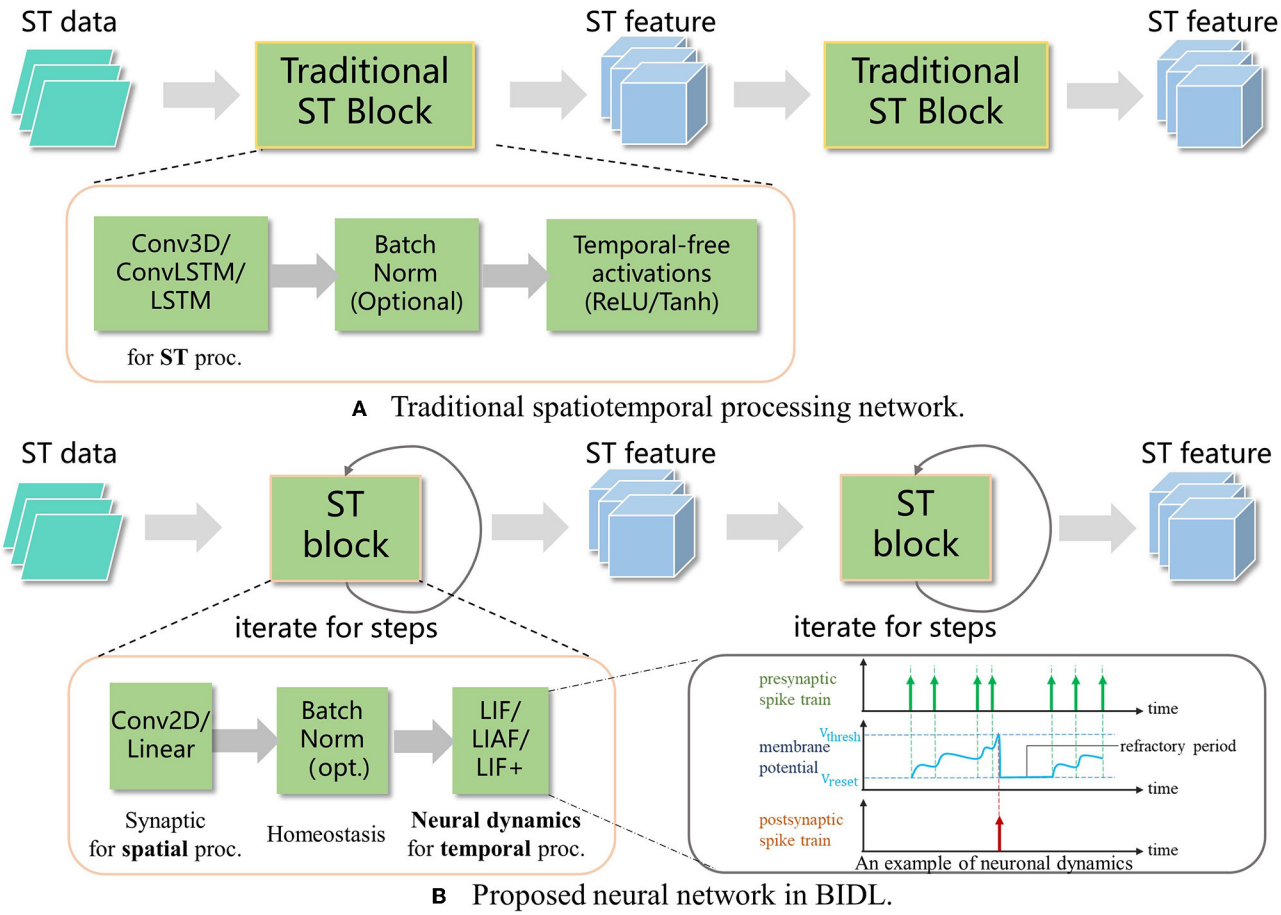


FIGURE 2
Proposed spatiotemporal processing by utilizing neural dynamics for temporal processing. **(A)** Traditional spatiotemporal processing via 3D convolution or Convolutional LSTM. **(B)** The proposed ST block (such as ConvLIF/ConvLIAF) is applied for spatiotemporal processing, where 2D convolution or linear operations are employed for spatial processing, and bio-inspired neurons with dynamics are introduced for temporal processing. Due to the lower computational requirements of 2D convolution and bio-inspired neurons compared to Conv3D or ConvLSTM, the network becomes lightweight and enables real-time processing.

In the following illustrations, we term convolutional integration with LIAF/LIF as ConvLIAF/ConvLIF, respectively. We use an RMP flag (such as ConvLIF-RMP) when a soft reset is used. In addition, parameters $\boldsymbol{V}_{th}$, $\boldsymbol{V}_{reset}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$ may vary for each convolutional channel termed as channel sharing mode (CSM) or be the same for all neurons termed as all sharing mode (ASM). Since $\boldsymbol{r}$ is a resistance to the input current, and the input current is derived from synaptic integration, we discarded it since it can be represented as a gain factor of the synaptic weights. There are also many variations for the proposed LIF, which are noted as LIF+. Please refer to Section 3.1.1 for details.

2.2. Lightweight processing

To verify the lightweight characteristics of the proposed networks in BIDL, we summarized the computational cost and memory cost of the proposed ST block, comparing it with traditional ST blocks such as Conv3D and ConvLSTM. The comparison is shown in Table 1.

We assume that the hidden state and output have the same spatiotemporal tensor size, denoted as $[B, T, H, W, C]$, where $B = 1$. (I, J) represents the convolution kernel size of Conv2D, ConvLIF, ConvLIAF, and ConvLSTM, while (U, I, J) represents the convolution kernel size of Conv3D. The variable K denotes the size of input channels. For the sake of comparison, in Table 1, we use R and Q to represent $T \cdot H \cdot W \cdot C$ and $I \cdot J \cdot K$, respectively. We also fix these parameters to a set of typical values and plot the corresponding values of computational operations and weight parameters, as shown in Figure 4.

The results from Table 1 and Figure 4 reveal that the computational overhead of ConvLIAF is not significantly different from that of Conv2D (time-distributed). Furthermore, ConvLIF consumes fewer multiplications thanks to its spiking format. It can be observed that, for the same temporal tensor shape, both ConvLIAF and ConvLIF achieve a computational reduction of 3x compared to Conv3D and 8x compared to ConvLSTM, as shown in Figure 4. Therefore, compared with Conv3D and ConvLSTM,

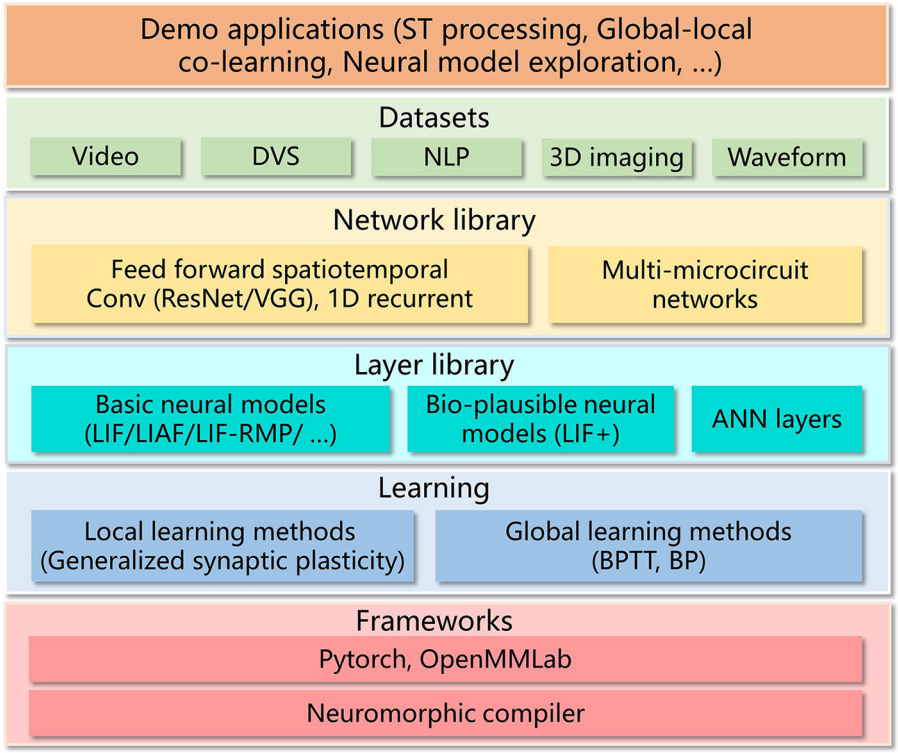


FIGURE 3
Top-level architecture of BIDL. BIDL is based on PyTorch and OpenMMLab (MMCV-Contributors, 2018). It consists of learning modules, a layer library (neural models), a network library, datasets, and demo applications. The networks designed by BIDL can be ported to the neuromorphic compiler via a state-aware computational graph representation.

ConvLIAF can save a significant amount of computational resources and storage overhead. The impact on accuracy is reported in literature Shi et al. (2020) and Wu et al. (2021), and is also discussed in our experiment section.

3. BIDL: an easy-to-use platform for SNN researchers

The BIDL platform is designed to cater to two main types of researchers: computational neuroscience researchers and machine learning researchers.

3.1. For computational neuroscience researchers

Computational neuroscience researchers primarily focus on building various neural models, exploring synaptic plasticity rules, and studying network structures. Hence, BIDL provides support for these features, and spatiotemporal applications serve as experimental examples to support their theoretical viewpoints.

3.1.1. Neural model support

Unlike most computational neuroscience frameworks such as Nest (Gewaltig and Diesmann, 2007) and Neuron (Carnevale and Hines, 2006), where neurons cannot be directly trained using gradient-based learning methods, BIDL introduces a group of neurons called LIF+ that can be trained directly using backpropagation through time (BPTT) with surrogate gradients. The LIF+ neurons are derived from the original leaky integrate and fire (LIF) model by incorporating improvements in key aspects of the differential equations to enhance neurodynamics, as inspired by Lee et al. (2018). Figure 5 illustrates the decomposition of LIF+ into five stages, each offering several choices for improving the similarity to biological neural dynamics. Detailed definitions of each mode can be found in Lee et al. (2018). Additionally, BIDL supports customized neural models, allowing users to define their own models expressed as sub-networks in PyTorch.

3.1.2. Generalized synaptic plasticity rules

Exploring various synaptic plasticity approaches is common among computational neuroscience researchers. In BIDL, we introduce a local learning module, which is a customizable programmable module that supports Hebb, spike-timing dependent plasticity (STDP), and reward-modulated STDP (R-STDP), as shown in Figure 6. This module receives spikes and optionally membrane potentials from the current neuron population (e.g., a LIF layer) as well as the previous neuron population. It also receives the reward signal from the external environment and reads the

TABLE 1 Formulas for calculating the computational complexity and the weights of different spatiotemporal layers.

Layer	MUL	ADD	Weights
ConvLIAF	$(Q+1)\cdot R$	$(Q+2)\cdot R$	$(Q+1)\cdot C$
ConvLIF	R	$(Q+2)\cdot R$	$(Q+1)\cdot C$
Conv2D (TD)	$Q\cdot R$	$Q\cdot R$	$(Q+1)\cdot C$
Conv3D	$U\cdot Q\cdot R$	$U\cdot Q\cdot R$	$(U\cdot Q+1)\cdot C$
ConvLSTM	$(4\cdot(Q+I\cdot J\cdot C)+3)\cdot R$	$(4\cdot(Q+I\cdot J\cdot C)+1)\cdot R$	$(Q+I\cdot J\cdot C+1)\cdot 4\cdot C$

TD refers to the time-distributed operation, i.e., duplicate over time.

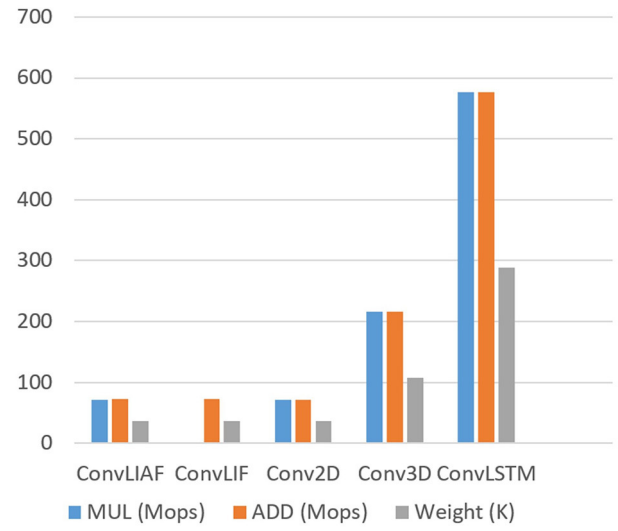


FIGURE 4
Computational complexity and number of weights of a representative ST block for an example parameter setting (T = 8, H = W = 16, C = K = 64, U = I = J = 3). It reveals that ConvLIF/ConvLIAF consumes similar computational resources and parameters to (time-distributed) Conv2D, while much fewer than Conv3D and ConvLSTM.

previous weights of the synaptic array. Using the user-defined update function *SP*, the module calculates the weight update value Δw.

3.2. For machine learning researchers

3.2.1. A deep learning style SNN builder

For machine learning researchers, their focus is on how SNNs can assist DNNs in efficient spatiotemporal processing and designing SNNs using a DNN building approach. BIDL leverages popular DNN designing frameworks such as PyTorch and OpenMMLab (MMCV-Contributors, 2018) to provide a familiar development environment. Researchers can reuse DNN backbones such as ResNet and VGG for building SNNs in BIDL. The networks can be trained using backpropagation through time (BPTT) similar to pure DNNs, enabling DNN researchers to seamlessly transition to SNN work with minimal adaptation required.

3.2.2. Global-local learning support

Global learning methods such as BPTT excel in learning from supervised information and achieve high accuracy for many applications. However, they come with a high computational burden and memory usage since the membrane potentials and activations of each timestep in the forward propagation need to be recorded for the backward propagation. On the contrary, local learning methods based on synaptic plasticity rules offer better computational energy efficiency but often suffer from lower accuracy. In this regard, we propose a hybrid training approach that integrates global-local learning into a unified flow, allowing interleaved operation of global learning and local learning.

During global learning, the synaptic plasticity learning module is detached, and the network is trained using BPTT. During local learning, the weights are adjusted via local plasticity, and BPTT is not required. This hybrid approach is particularly useful in pre-training-finetuning scenarios, where the network is first trained globally using BPTT and then fine-tuned using reward-modulated STDP (R-STDP) to adapt to changes in the environment.

4. Mapping optimizations for neuromorphic chips

BIDL can also serve as an application builder for neuromorphic chips, taking into consideration their unique characteristics and constraints. It generates a computational graph that can be used by subsequent neuromorphic compiling tools. Currently, neuromorphic chips accept computational descriptions with specific constraints, including being timestep driven, caching state variables (membranes) for use in the next timestep, and parameter quantization for memory savings.

(1). Most neuromorphic chips operate in a timestep driven manner, where a timestep iteration is located outside the neural network. The device computes all the layers in each timestep and then switches to the next timestep. In contrast, GPU-based SNN frameworks usually locate the timestep iteration within each layer, resulting in outputs with an extra temporal dimension. To address this difference, we designed two modes of operation: internal iteration mode (IIM) and external iteration mode (EIM), shown in Figure 7.

In IIM, single-step layers such as convolutions and LIF neurons are wrapped by a timestep iteration and a reset phase, forming iterated layers. The network is built based on these iterated layers, which have a temporal dimension on both the input and output. In EIM, the single-step network is built directly using single-step

10.3389/fnins.2023.1213720

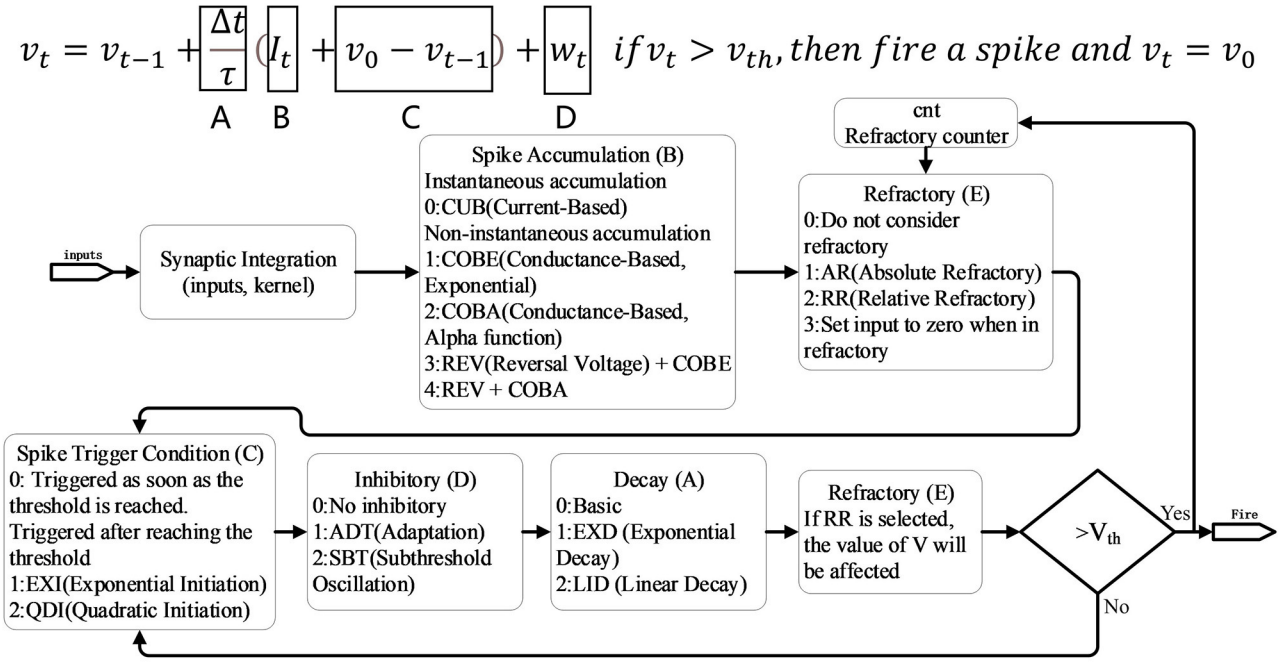


FIGURE 5
LIF+ neuron with configurable modes. Each mode can be treated as an improvement of the original simplified LIF model, including A: decay, B: spike accumulation, C: spike trigger condition, D: inhibitory, and E: refractory. Detailed definitions can be found in Lee et al. (2018).

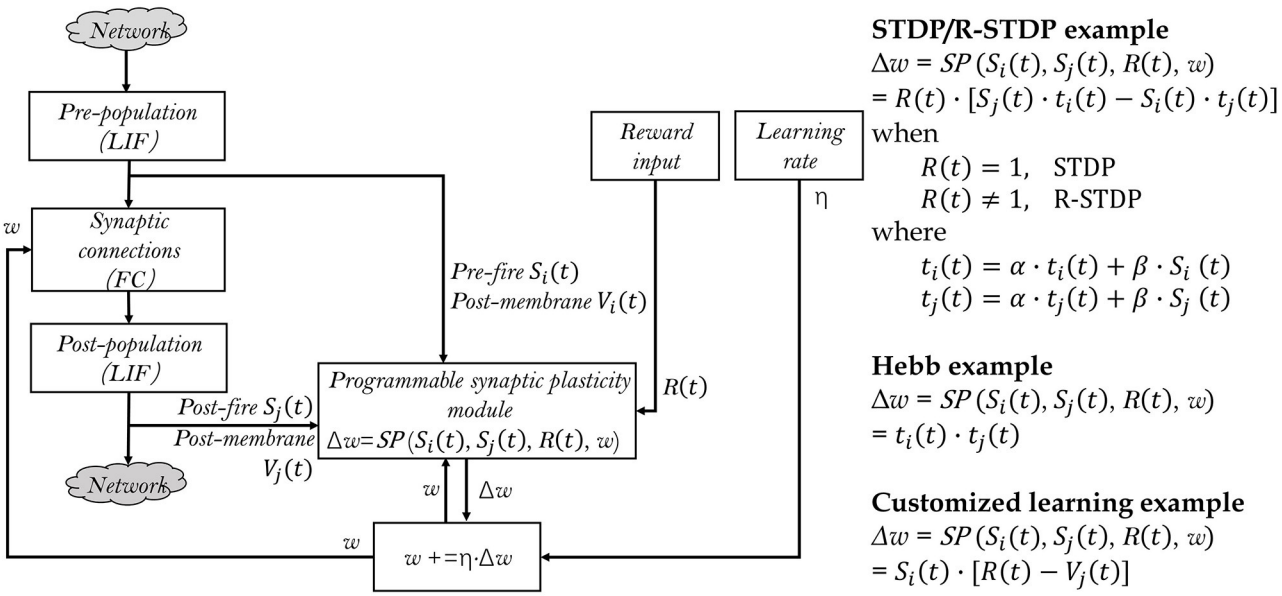


FIGURE 6
Generalized local learning module, which can be configured to support STDP, R-STDP, Hebb, or user-defined synaptic plasticity rules. The two LIF populations and synaptic connections are part of the neural network, while the remaining parts of the neural network are denoted as "Network" for simplicity. This also indicates that only a subset of the synaptic weights of interest are adjusted during local learning.

layers, and the network is invoked T times, where T is the number of timesteps. Both modes can be trained using BPTT and achieve similar accuracy.

IIM offers more flexibility as each iterated layer can have its own total number of timesteps, and temporal transforms (such as decreasing the number of timesteps) or attention operations can be applied between timesteps for temporal information aggregation.

For classification tasks, where there is a single timestep at the end of the network, a temporal aggregation layer (e.g., sum or average) is applied before the classification head to reduce

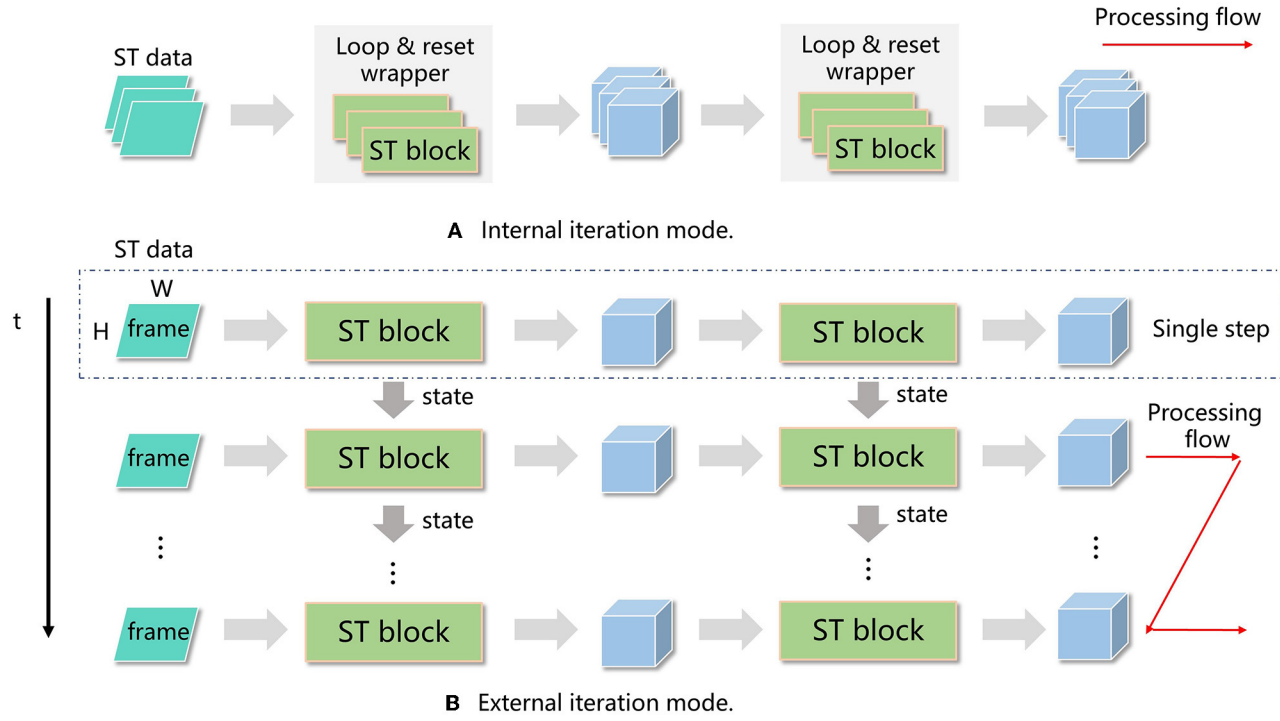


FIGURE 7
Processing flow of internal iteration (IIM) **(A)** and external iteration (EIM) **(B)**. In IIM, a loop and reset wrapper is introduced in each spatiotemporal (ST) block, and all the timesteps for the current block are calculated before moving to the next block. In EIM, the timestep iteration is located outside the network, and all the layers in the network are processed before moving to the next timestep.

the number of timesteps to 1. This is straightforward in IIM, but for EIM, all layers share the same timestep setting, making direct temporal aggregation impractical. In such cases, we use an accumulator instead of temporal aggregation. Mathematically, instead of calculating the output as the sum of inputs over timesteps, i.e., $Out = \sum_{i=0}^{T-1} Input(i)$, we use an accumulator $Acc(i)$ defined as $Acc(i) = Acc(i-1) + Input(i)$, with a reset value of $Acc(-1) = 0$, and the output is obtained as $Out = Acc(T-1)$.

(2). When compiling the network for neuromorphic chips, we utilize EIM, and only a single-iteration network is compiled. We represent the network in a computational graph format, where each node corresponds to a functional layer, and each edge represents an intermediate tensor. In a traditional computational graph, these tensors can be destroyed after the graph computation is finished, meaning all the variables survive only at the current timestep. However, some state tensors, such as membrane potentials, need to be carried over to the next timestep and updated in place. To handle this, we introduce two additional nodes to the graph: the load node and the save node. The load node can be associated with a constant tensor node, which provides the initial value during the initialization stage or is reset by the user. Resetting is typically done when calculating a new sample and requires erasing the membrane potential from the previous sample. To distinguish between different state tensors in the graph, we assign them unique string identifiers, generated as universally unique identifiers (UUIDs). The load and save nodes are represented as customized layers in PyTorch.

(3). The compiled network is executed on the neuromorphic chip using a runtime tool. Within a single input sample, the runtime tool iterates through each timestep, provides input data to the device, performs the network inference, and obtains the output. The input data consists of a sequence of frames, with one frame processed at each timestep. For networks that generate single-step outputs (e.g., classification or detection), only the output of the last timestep is required. After executing all the timesteps, a state reset command is issued to reset the membrane potential, preparing for the next sample.

(4). Using PyTorch layers to represent the computational process of neurons may introduce additional nodes to the graph, making it more complex and less recognizable for optimization on neuromorphic chips. In most devices, basic neural models such as LIF neurons have dedicated circuits for implementation. Therefore, we rewrote the LIF neuron in PyTorch using two customized layers specifically for inference (not for training). These layers, cmp_and_fire and reset_with_decay, represent the compare-to-threshold and firing process and the membrane reset and decay process, respectively. These blocks can be treated as black boxes in the computational graph and recognized directly by neuromorphic chips, enabling circuit-level optimization.

Figure 8 illustrates the implementation of LIF neurons using the hardware-defined built-in blocks for acceleration on neuromorphic chips. The previous membrane potential is loaded and then added to the input post-synaptic current. The cmp_and_fire block is used to calculate the spike, and

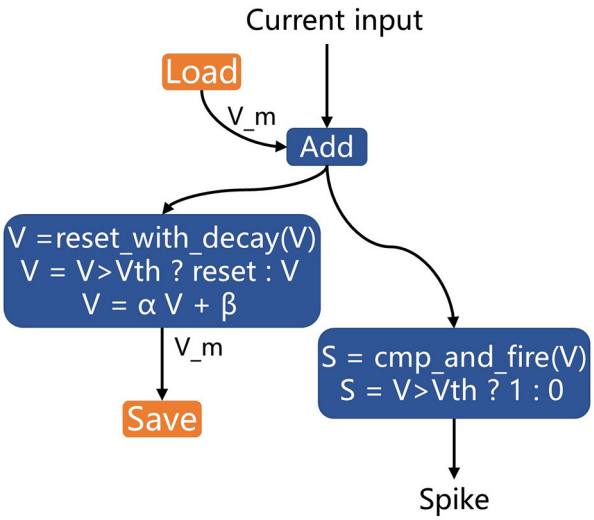


FIGURE 8
LIF neuron implementation with hardware-defined built-in blocks for neuromorphic chip acceleration.

the reset_with_decay block is applied to update the membrane potential and perform decay.

(5). To reduce memory footprint and computational energy, quantization is applied. BIDL provides float16 training, which utilizes the float16 support in PyTorch with the loss amplified by a user-defined setting, such as 512. These mechanisms discussed in this sub-section have been deployed in the brain-inspired chip Lynchip KA200.[1]

5. Experiments

In this section, we demonstrate several spatiotemporal applications in BIDL across multiple modalities, including video, DVS, 3D imaging, and NLP. We also evaluate several models in LIF+ with various neuronal variations to illustrate the neural modeling capabilities. Finally, we showcase local-globalized co-learning for high-accuracy transfer learning.

5.1. Applications

5.1.1. Video processing

Video gesture recognition (Jester): Currently, there is limited literature discussing video processing with spiking neural networks, and most existing networks for video processing have high computational complexity, consuming more power than image processing. In this study, we propose an SNN approach, ResNet18-LIF, which achieves a similar computational cost as the traditional ResNet-18, to demonstrate its lightweight processing capability.

We chose Jester dataset (Materzynska et al., 2019) for our experiments. Jester is a dataset of video clip gesture recognition collected using an ordinary camera, consisting of 27 types of hand gestures recorded by 1,376 participants in unconstrained environments, including different rooms, rotating fans, and moving animals. To the best of our knowledge, this is the largest video clip dataset showing human gestures, with 1,48,092 short video clips, each lasting for 3 s. We split the dataset into training/validation/test sets with a ratio of 80%/10%/10%. Many of the actions in this dataset are symmetric, such as "move finger to the left" and "move finger to the right," which require strong temporal modeling capabilities for accurate action recognition.

We conducted two versions of experiments with different image resolutions and network architectures. Version 1 focuses on low-cost processing, while version 2 prioritizes high accuracy.

Each action is represented as a sequence of multi-frame RGB images. For each frame, the image is resized to 112 × 112 for version 1 and 224 × 224 for version 2. We take 16 frames (T = 16) and perform simple data augmentation before inputting them into the network. The input data format for the network is [B, 16, 3, 112, 112] for version 1 and [B, 16, 3, 224, 224] for version 2.

The neural network architecture follows a structure similar to ResNet-18, with LIAF used as the neural module for temporal processing (Figure 9). In version 1, the results of all timesteps are summed at the SumLayer and then divided by T for temporal dimension aggregation. The classified output is obtained through the fully connected (FC) layer. In version 2, we refine the network by substituting LIAF with the electrical coupling LIAF-RMP neural model (Wu et al., 2022b). Although the LIAF-RMP model achieves better accuracy, it incurs higher computational cost and a larger parameter size (the electrical synapses lead to a network weight size increase from 12×10^6 to 24×10^6).

We trained the network on the training set using a learning rate of 1e-1, weight decay of 1e-4, and momentum stochastic gradient descent (SGD) optimizer with a momentum value of 0.9. The training process utilized a cosine annealing learning rate tuning strategy. After training for 200 epochs, the top-1 classification accuracy reached 93.7% and 95.0% on the validation set for version 1 and version 2, respectively.

Moving trajectory processing (RGB-gesture): In some cases, we are only interested in the trajectory of the target and do not focus on static image features. In such situations, we can use the differential frame sequence as input, resulting in a simpler network. In this study, we propose the RGB-gesture dataset for this purpose. The dataset is collected using an ordinary camera and contains 10 moving gestures captured for each person similar to the DVS128 Gesture dataset (Amir et al., 2017). The video is decoded into frame data at a frequency of 25 frames per second and stored. The RGB-gesture dataset includes 760 training samples and 102 validation samples.

The RGB frames are first converted to grayscale images. We obtain a differential image by subtracting the corresponding pixel values of adjacent frames. If the pixel value changes are below a threshold, they are considered as background. Significant changes indicate moving objects and are marked

1 The commercial version of BIDL with Lynchip support is named LYNBIDL.

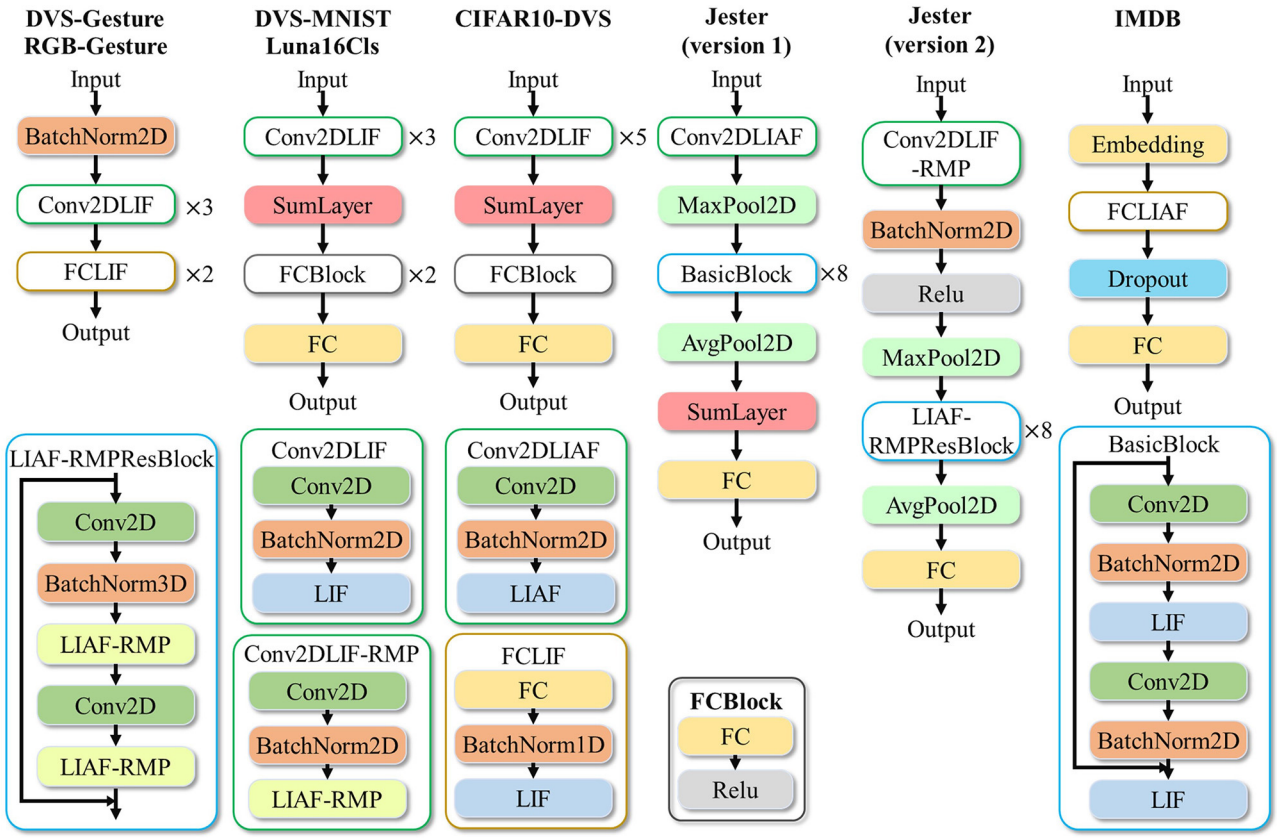


FIGURE 9
Designed neural network structures for the proposed experiments.

as foreground. The differential results in two image channels: enhancement and weakening. After preprocessing, each sample in RGB-gesture has a dimension of $[B, 60, 2, 40, 40]$, with $T = 60$.

The model is trained using the Adam optimizer with a learning rate of 1e-3 and weight decay of 1e-4. We also use the pre-trained model trained on the DVS128 Gesture dataset. The model is trained for 50 epochs, achieving a top-1 classification accuracy of 97.7% on the validation set.

5.1.2. DVS signal processing

DVS is a silicon retina device that mimics the human retina's perception mechanism to perform information acquisition. The data preprocessing for converting event flow to spatiotemporal (ST) tensor is as follows: A sliding window is used to slide along the time, and an event set contains the timestamp range of events within the sliding window. Then, all events in an event set are extended into a three-dimensional vector called a frame based on their coordinate and polarity information. The positive/negative polarity events are filled in a $H \times W$ matrix according to their coordinate information in the positive/negative polarity channel, while the unfilled coordinates are set to zero. After T timesteps, an ST tensor with T frames can be obtained.

DVS classification (CIFAR10-DVS): CIFAR10-DVS (Li et al., 2017) is a dataset derived from the CIFAR10 dataset and collected using the DVS. We follow the event-to-ST tensor conversion methods described above. The ST tensor has a shape of $[B, 10, 2, 128, 128]$, where $T = 10$, and the temporal sliding window is 5 ms. The proposed network contains five Conv2DLIF layers, followed by a SumLayer for time aggregation, and two FcBlock layers. We use the Adam optimizer with a learning rate of 1e-2 and weight decay of 1e-4 to train this network. The network's neuron parameters are set to all-share mode. The model is trained for 100 epochs, achieving a top-1 classification accuracy of 68.2% on the validation set.

DVS recognition (DVS128 gesture): The DVS128 Gesture dataset (Amir et al., 2017) is recorded directly from real-world scenes using a DVS camera. The DVS128 Gesture dataset has a raw spatial pixel resolution of 128 × 128. We chose a subsampled resolution of 40 × 40 (1/3.2) to save memory. To use the network structure for training, we generated event frames of size 40 × 40 by accumulating spike sequences within each 25 ms. Then, each frame was expanded into two channels depending on whether the brightness of each pixel was weakened or strengthened. Finally, multiple adjacent event frames were stacked in chronological order to obtain samples of shape $[B, 60, 2, 40, 40]$. This network structure includes three Conv2DLIF modules and two FCLIF modules. The

neuron parameters were trained in the all-share mode. We used the Adam optimizer with a learning rate of 1e-2 and weight decay of 1e-4 to train this network and employed a learning rate tuning strategy during training. The network was trained for 100 epochs, achieving a top-1 classification accuracy of 94.6% (ASM) and 95.1% (CSM) on the validation set.

5.1.3. 3D medical imaging

The Luna16Cls dataset is derived from the Lung Nodule Analysis 2016 (LUNA16) dataset (Setio et al., 2017). It contains CT images of 888 patients and 1, 186 nodule labels (malignant and benign) annotated by radiologists.

The preprocessing steps for the Luna16Cls dataset are as follows: (1) convert all raw data to Hounsfield Units (HU); (2) mask extraction; (3) convex hull and dilation processing; (4) gray normalization: linearly transform HU values [−1,200, 600] to grayscale values within the range of 0 to 255; and (5) downsample the dataset to a 32 × 32 image resolution to obtain samples with shape $[8, 1, 32, 32]$. A total of 3,795 samples are processed, with 3,416 samples used for training and the remaining samples for validation.

The Luna16Cls classification network consists of three Conv2DLIF blocks. After that, a temporal average layer is used to aggregate information along the time dimension. The model ends with an FcBlock, containing three fully connected layers for classification.

We use an SGD optimizer with a learning rate of 0.03, weight decay of 1e-4, and momentum of 0.9 to train the model on the training set. The learning rate is fine-tuned during the training process. The neural parameters are set to all-share mode. The model is trained for 20 epochs, achieving a top-1 classification accuracy of 90.4% on the validation set.

5.1.4. NLP task

To test the capability of the proposed method on long sequence signal processing, such as text, a simple natural language processing (NLP) task was conducted using the IMDB dataset (Maas et al., 2011). The dataset contains 50,000 highly polarized reviews from the Internet Movie Database (IMDb). Each word in the sample data was converted into a numeric representation using a dictionary of size 1,000. Each sample data was padded to a size of 500 timesteps, resulting in a dimension of $[B, 500]$ for each sample. Binary classification was used for labeling, with 0 representing negative sentiment and 1 representing positive sentiment.

The data are embedded in a tensor with shape $[B, 500, 256]$, followed by an FCLIAF layer, and finally, classification is performed using a fully connected layer. This model does not have a time aggregation layer and only outputs the result of the last timestep.

For training, the Adam optimizer with a learning rate of 1e-3 and weight decay of 1e-4 is used. The learning rate is adjusted based on the epoch during training. The model is trained for 50 epochs, achieving a classification accuracy of 82.9% on the validation set.

TABLE 2 List of the experiments.

Source	Dataset	Network	Iter. mode	Computations ($\times 10^9$ Ops)	Weights ($\times 10^6$)	Accuracy (%)	GPU V100 speed (fps)	Xavier speed (fps)	GPU V100 power (W)	Xavier power (W)
Video	Jester (ver. 1)	ResNet18-LIAF	EIM	ADD: 7.8 MUL: 7.8	11	93.7± 0.1	133± 5	65± 4	26.6±5.9	9.4±2.6
Video	Jester (ver. 2)	ResNet18-LIAF-RMP	IIM	ADD: 37.07 MUL: 37.07	24	95.0±0.1	88±5	47±1	29.7±6	18±3
Video	RGB-Gesture	ConvLIAF	EIM	ADD: 1.0 MUL: 1.0	2	97.7±0.1	538±90	270±3	34.8±3.3	7.7±1.4
DVS	DVS128 Gesture	ConvLIF	EIM	ADD: 1.0 MUL: 1.0	2	94.6±0.6	508±75	274±2	34.8±3.7	7.7±1.4
DVS	DVS128 Gesture	ConvLIF+CSM	EIM	ADD: 1.0 MUL: 1.0	2	95.1±0.8	558±22	258±6	36.1±2.8	7.8±1.1
DVS	CIFAR10-DVS	ConvLIAF	EIM	ADD: 0.84 MUL: 0.84	2.6	68.2±0.5	389±65	192±1	35.4±2.9	7.6±1.3
3D Img.	Luna16Cls	ConvLIAF	EIM	ADD: 0.089 MUL: 0.089	1.2	90.4±0.2	620±91	267±4	34.5±3.5	7.4±0.9
Text	IMDB	FCLIAF	EIM	ADD: 0.0024 MUL: 0.0024	0.0051	82.9±0.1	2140±290	1055±22	34.9±3.7	6.5±0.3

We evaluated implementations in BIDL with different applications, network structures (We use ASM by default), neural models, iteration modes, and platforms. It shows that all these implementations achieve real-time processing on GPU servers and embedded platforms with streaming processing settings (batchsize = 1). Note that frames per second (fps) is equal to timesteps per second.

TABLE 3 Comparison of proposed solutions with other approaches in terms of accuracy and computational costs.

Proposals	Network	Computations ($\times 10^9$ Ops)	Accuracy (%)
Video processing (Jester)			
Meng et al. (2021)	LSTM	ADD: 14.7 MUL: 14.7	93.5
Meng et al. (2021)	AdaFuse (TSN+ResNet18)	ADD: 7.6 MUL: 7.6	93.7
Jiang et al. (2019)	STM (ResNet50)	-	96.7
Zhang et al. (2020b)	STSNN (Optical flow+RGB)	-	95.7
Zhang et al. (2020a)	PAN (TSM+ResNet101)	ADD: 503.4 MUL: 503.4	97.4
This study (version 1)	ResNet18-LIAF	ADD: 7.8 MUL: 7.8	93.7
This study (version 2)	ResNet18-LIAF-RMP	ADD: 37.07 MUL: 37.07	95.0
Video processing (RGB-Gesture)			
This study	ConvLIAF	ADD: 1.0 MUL: 1.0	97.7
DVS signal processing (DVS128 Gesture)			
Massa et al. (2020)	SNN converted from CNN on Loihi	-	89.6
Amir et al. (2017)	CNN on TrueNorth	-	94.6
Kugele et al. (2020)	SNN converted from ANN	-	95.6
Khoei et al. (2020)	Converted CNN	-	95.1
Wang et al. (2019)	PointNet++	-	95.3
Bi et al. (2020)	Residual graph CNN+Res.3D	ADD: 14 MUL: 14	97.2
Wu et al. (2021)	ConvLIF	ADD: 6.8 MUL: 0.013	94.1
Wu et al. (2021)	ConvLIAF	ADD: 6.8 MUL: 6.8	97.6
This study	ConvLIF	ADD: 1.0 MUL: 1.0	94.6
This study	ConvLIF+CSM	ADD: 1.0 MUL: 1.0	95.1
DVS signal processing (CIFAR10-DVS)			
Cannici et al. (2019)	Attention Mechanisms	-	44.1
Sironi et al. (2018)	HATS	-	52.4
Wu et al. (2019)	iterative LIF + NeuNorm	ADD: 8.1 MUL: 8.1 *	60.5
Wu et al. (2021)	ConvLIF	ADD: 3.8 MUL: 0.21	63.5
Wu et al. (2021)	ConvLIAF	ADD: 3.8 MUL: 3.3	70.4
This study	ConvLIAF	ADD: 0.84 MUL: 0.84	68.2
3D medical imaging processing (Luna16Cls)			
Yan et al. (2017)	Vanilla 3D CNN	-	87.3
Shen et al. (2017)	Multi-crop CNN	-	87.4
Zhu et al. (2018)	Deep 3D DPN	ADD: 26 MUL: 26 *	87.1
Dey et al. (2018)	DenseNet	ADD: 0.14 MUL: 0.14 *	88.4
Dey et al. (2018)	MoDenseNet	ADD: 0.14 MUL: 0.14 *	90.4
Shi et al. (2020)	LIF-classification Net	-	94.1
This study	ConvLIAF	ADD: 0.089 MUL: 0.089	90.4
Text processing (IMDB)			
This study	LSTM	ADD: 0.4 MUL: 0.4	85.7
This study	FCLIAF	ADD: 0.0024 MUL: 0.0024	82.9

* Indicates that we calculate the data based on the network structure of the corresponding study.

5.2. Experiment results analysis

We chose these examples since they include multiple modalities, different network structures (ResNet-LIF and sequential LIF), iteration modes (IIM and EIM), neural parameter sharing mode (CSM and ASM), and various application domains, which are listed in Table 2. We also evaluated the streaming processing speed (with batch size = 1) vs. the accuracy performance of these models. It can be revealed that all the applications can work in real time both in Nvidia GPU V100 and Nvidia Jetson Xavier with streaming processing capability. Jester version 2 achieves better performance than version 1 while it suffers more computational cost. The two DVS128 Gesture implementations show that CSM achieves slightly better performance than ASM.

We further compared the accuracy and the computational complexity of the proposed implementations with other related work, and the comparison is revealed in Table 3. For the Jester dataset, our implementation (version 1) achieves lightweight processing which consumes half of the computations compared to the LSTM approach and similar computation to the lightweight CNN approach AdaFuse (Meng et al., 2021). Higher accuracy can also be achieved via Jester version 2, which has comparable performance to other high accuracy approaches. For DVS128 Gesture, we also achieve a better balance of performance and computational cost. For CIFAR10-DVS, we achieve better accuracy than most of the other approaches while reducing the computation by more than 4 times. For Luna16Cls, we achieve the smallest cost while maintaining accuracy. For IMDB, the proposed approach consumes only 7% of the computation compared to LSTM while maintaining accuracy (with less than 3% performance loss). In conclusion, the proposed framework can process various spatiotemporal signals with guaranteed performance and better computational efficiency.

5.3. Global-local co-learning

Recent studies have shown that global-local co-learning is more resistant to noise (Wu et al., 2022a). In this study, we conducted an experiment to demonstrate the application of co-learning on a spatiotemporal network to achieve improved noise resistance performance.

To introduce noise into the data, we added background noise to the preprocessed frames of the DVS128 Gesture validation set. First, we defined a noise ratio $\alpha \in [0, 1]$. For each validation sample, we randomly selected $n = \alpha \times W \times H$ pixels on each channel of every frame and set their values to 1, thereby introducing background noise. The training set remained unchanged. The network used

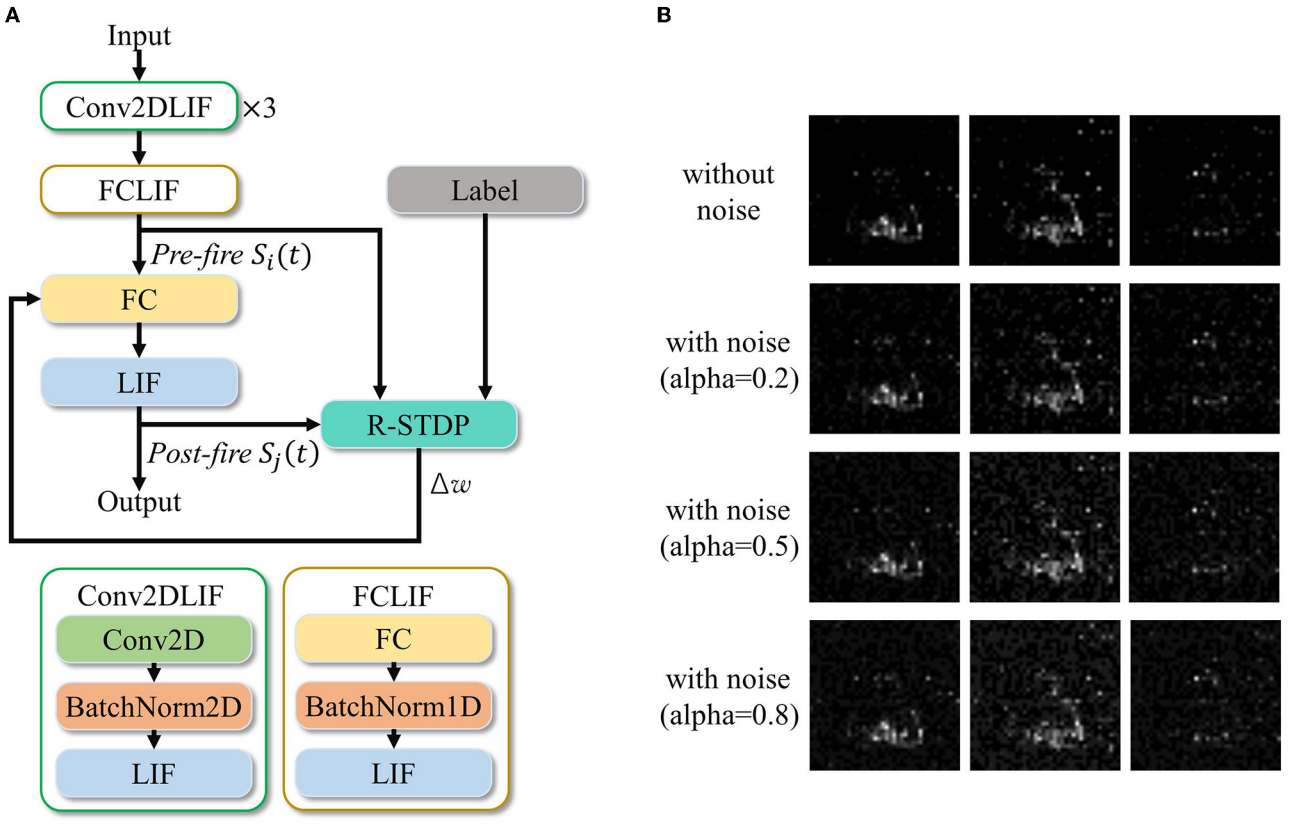


FIGURE 10
An example implementation of the BPTT and R-STDP co-learning for improved anti-noise performance. **(A)** Network architecture with R-STDP finetuning. **(B)** Visualization of the three ST frames with different noise ratios.

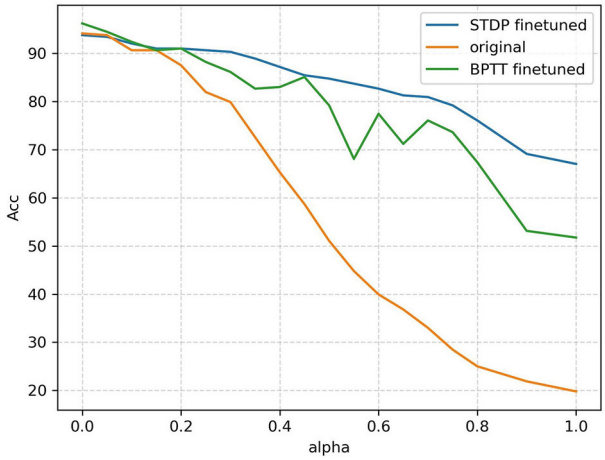


FIGURE 11
Accuracy versus noisy ratio (alpha) curves of the three approaches: BPTT without population coding (original), BPTT with population coding (BPTT finetuned), BPTT+STDP with population coding (STDP finetuned). When population coding is introduced, seven neurons represent one classification category; otherwise, one neuron represents a category.

for co-learning is similar to the one described in Section 3.2.2 and Section 5.1.2.

The experiment followed the procedure outlined below. First, we trained the network using BPTT, as described in Section 3.2.1. For comparison purposes, we performed inference on this network directly without local training on the noisy validation set, varying the value of α, and recorded the corresponding inference accuracies.

Next, we employed a local learning method to further refine the trained network. We adjusted the weights W of the last FCLIF layer using R-STDP. In the figure, W is represented by a 256x11 matrix, and we used seven neurons to represent each category, resulting in a total of 11 categories. The weight update process was streamed, with each sample being updated at every timestep and a batch size of one. The specific architecture of the R-STDP local learning is depicted in Figure 10A, and the noisy frames are visualized in Figure 10B.

The R-STDP method we propose consists of the following three steps:

(1) Reward calculation: Calculate the mean spikes of neurons belonging to each category as $O_i = \frac{1}{N}\sum_{i=0}^{N-1} O_{ij}$, where $i \in [0, 10]$ represents the 11 categories, and $N = 7$ indicates the seven neurons in each category (i.e., population coding). Then, calculate the reward r_{ij} as the difference between the label and the output spikes: $r_{ij} = L - O_i$, where L denotes the label.

(2) Weight update based on R-STDP: Calculate ΔW using the input spike S_{in} and output spike S_{out} as follows:

$$\Delta W = t_{in} \cdot S_{out} - S_{in} \cdot t_{out}. \tag{12}$$

Here, the traces t_{in} and t_{out} are derived from spikes and are updated as

$$t_{in/out} = \Theta \cdot t_{in/out} + \eta \cdot S_{in/out}. \tag{13}$$

(3) Update weights: $w = w + lr \cdot \Delta w$.

TABLE 4 LIF+ performance with selected configurations on the CIFAR10-DVS dataset, where the "All default" configuration is equivalent to a LIF model, and other configurations are defined in Section 3.1.1.

Configuration	Accuracy (%)
All default (A=0, B=0, C=0, D=0, E=0)	67.86
A=0, B=3, C=0, D=0, E=0	67.43
A=0, B=3, C=1, D=0, E=0	67.63
A=0, B=3, C=2, D=0, E=0	67.56

We set $lr = 0.001$, $\Theta = 0.95$ and $\eta = 1.0$.

After the training, we evaluated the accuracy of the noisy validation set. The accuracies obtained by the two methods are plotted in Figure 11. The three approaches compared are BPTT training without fine-tuning and 11 output neurons (original curve); BPTT with R-STDP fine-tuning and 77 output neurons (STDP fine-tuned curve); and BPTT with BPTT fine-tuning and 77 output neurons (BPTT fine-tuned). The results demonstrate that in a noisy environment, global-local co-learning achieves better anti-noise performance compared to a pure BPTT-trained network with the same number of output neurons and significantly outperforms the non-population coding BPTT version.

5.4. LIF+ neural models

To verify the performance impact of various neural models, we re-implemented CIFAR10-DVS using several LIF+ configurations while keeping all other settings the same as illustrated in Section 3.1.1. We tested a subset of configurations and present the results in Table 4. While some neural dynamics are not compatible with gradient backpropagation and result in lower performance, certain configurations still achieve similar results to the default LIF model. This indicates the admissibility of exploring neural models for spatiotemporal tasks. Future studies can focus on identifying neural models with improved performance.

6. Discussions

6.1. Single frame (image) processing

Processing images within this framework is possible by encoding the image into a sequence of specialized frames using a coding scheme, such as binary frames for SNN processing. However, we assume that SNN's multiple timestep processing may require more computation compared to single-frame processing in CNNs. Since most of the computation in CNNs is attributed to the convolutional operation, which does not benefit significantly from sparse (event-driven) processing, the computational cost of an SNN for processing a frame is unlikely to be significantly lower than an equivalent-sized CNN. Therefore, our framework does not aim to process image sources.

6.2. Neural coding

For the entire network, spikes are emitted based on neural dynamics and trained using gradient-based or local algorithms. Thus, no dedicated neural coding is assigned to any neuron, and the neurons fire based on their dynamics. Regarding input data, it is possible to represent temporal-free data using neural coding in the temporal domain. For instance, a pixel in the input image can be represented by a spike train with a corresponding firing rate (rate coding) or first firing time (temporal coding). The network's output in BIDL is primarily in analog format, but it can be encoded into a spike train.

6.3. Direct training vs. conversion methods

In this framework, we utilize direct training for DSNN. Conversely, there are approaches (Tran et al., 2015; Xingjian et al., 2015) that convert artificial neural networks (ANNs) to SNNs, achieving high accuracy. However, these conversion methods are primarily designed for image processing without temporal domain information in the source data. The temporal domain is generated using neural coding during the conversion stage. Our study focuses on spatiotemporal processing, where the temporal information already exists in the source data. Additionally, direct training achieves better performance and enables fine-tuning, offering more flexibility compared to post-training conversion methods.

6.4. Strengths and limitations of BIDL

Prior studies (Chen and Gong, 2022; Chen et al., 2022) have also proposed spatiotemporal investigations aiming to establish brain-inspired models and verify visual processing functions with biological evidence. In comparison, BIDL focuses more on solving real-world spatiotemporal tasks with DSNNs. It explores the utilization of brain-inspired technologies for spatiotemporal applications, emphasizing computational efficiency and real-time processing. Unlike some DSNN works (Wu et al., 2018; Shen et al., 2022), BIDL incorporates multiple modalities such as video and 3D imaging data, flexible neuron models, and global-local co-learning. However, BIDL has limitations in modeling sparse brain networks, especially with synaptic delays, as the computation is performed in a dense tensor format. Furthermore, it does not gain significant benefits from event-driven processing due to tensor-based convolution/linear operations. Nevertheless, BIDL achieves better efficiency through neural dynamics, which serve as lightweight processors of temporal information compared to Conv3D and ConvLSTM.

7. Conclusion

This study introduces a brain-inspired deep learning framework, BIDL, which provides a foundation and design flow for rapidly developing spatiotemporal applications, particularly lightweight real-time video clip analysis and dynamic vision sensor (DVS) applications. BIDL also serves as a research platform for investigating neuron models, synaptic plasticity, global-local co-learning, and network structure. Networks designed using BIDL can be easily deployed on GPU platforms and neuromorphic chips. We hope that BIDL will inspire further research in the design, exploration, and application development of bio-inspired neural networks.

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 authors.

Author contributions

ZW: conceptualization, methodology, investigation, and writing. YS: software development and investigation. JZ: software development, investigation, and data curation. HLia: writing and conceptualization. RZ: software development and methodology. HLi: software development (neuromorphic mapping). JX: review and editing. XZ: investigation (jester). YC: project administration, conceptualization, and supervision. All authors contributed to the article and approved the submitted version.

Funding

This study was supported by the Science and Technology Innovation 2030 - Key Project of New Generation Artificial Intelligence under Grant No. 2020AAA0109100, the National Key Research and Development Program of China (Grant No. 2021ZD0200300), Sichuan Science and Technology Program (No. 2021YFG0333), Zhongguancun Science and Technology Park Management Committee of Disruptive Technology Research and Development Project (202005012), Beijing Science and Technology Plan, China (Z221100007722020), and National Natural Science Foundation of China (U22A20103).

Acknowledgments

The authors would like to thank Xiaodong Hu, Han Yuan, Bingyang Hou, Wei He, Yinsong Yu, Weijiao Xiang, and Hongbing Qiu for their valuable suggestions, software development, and software testing on this study.

Conflict of interest

ZW, YS, JZ, RZ, and HLi are employed by Lynxi Technologies, Co. Ltd.

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

Publisher's note

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* 7243–7252. doi: 10.1109/CVPR.2017.781

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform.* 7, 48. doi: 10.3389/fninf.2013.00048

Bi, Y., Chadha, A., Abbas, A., Bourtsoulatze, E., and Andreopoulos, Y. (2020). Graph-based spatio-temporal feature learning for neuromorphic vision sensing. *IEEE Trans. Image Proces.* 29, 9084–9098. doi: 10.1109/TIP.2020.3023597

Bohte, S. M., Kok, J. N., and La Poutré, J. A. (2000). "SpikeProp: backpropagation for networks of spiking neurons," in *ESANN* (Bruges) 419–424.

Cannici, M., Ciccone, M., Romanoni, A., and Matteucci, M. (2019). "Attention mechanisms for object recognition with event-based cameras," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE) 1127–1136. doi: 10.1109/WACV.2019.00125

Carlos, D., Juan, H., Antelis, J. M., and Falcón, L. (2019). Spiking neural networks applied to the classification of motor tasks in EEG signals. *Neur. Netw.* 122, 130–143. doi: 10.1016/j.neunet.2019.09.037

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

Chen, G., and Gong, P. (2022). A spatiotemporal mechanism of visual attention: Superdiffusive motion and theta oscillations of neural population activity patterns. *Sci. Adv.* 8, eabl4995. doi: 10.1126/sciadv.abl4995

Chen, G., Scherr, F., and Maass, W. (2022). A data-based large-scale model for primary visual cortex enables brain-like robust and versatile visual processing. *Sci. Adv.* 8, eabq7592. doi: 10.1126/sciadv.abq7592

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. Neuroinform.* 11, 2008. doi: 10.3389/neuro.11.011.2008

Dey, R., Lu, Z., and Hong, Y. (2018). "Diagnostic classification of lung nodules using 3D neural networks," in 2018 *IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)* (IEEE) 774–778. doi: 10.1109/ISBI.2018.8363687

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). *SpikingJelly*. Available online at: https://github.com/fangwei123456/spikingjelly (accessed April 26, 2023).

Ferré, P., Mamalet, F., and Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based STDP. *Front. Comput. Neurosci.* 12, 24. doi: 10.3389/fncom.2018.00024

Gewaltig, M.-O., and Diesmann, M. (2007). NEST (neural simulation tool). *Scholarpedia* 2, 1430. doi: 10.4249/scholarpedia.1430

Golosio, B., Tiddia, G., De Luca, C., Pastorelli, E., Simula, F., and Paolucci, P. S. (2021). Fast simulations of highly-connected spiking cortical models using GPUs. *Front. Comput. Neurosci.* 15, 627620. doi: 10.3389/fncom.2021.627620

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE Trans. Neur. Netw. Learn. Syst.* 28, 2222–2232. doi: 10.1109/TNNLS.2016.2582924

Gu, P., Xiao, R., Pan, G., and Tang, H. (2019). "Stca: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *IJCAI* 1366–1372. doi: 10.24963/ijcai.2019/189

Han, B., Srinivasan, G., and Roy, K. (2020). "RMP-SNN: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/CVPR42600.2020.01357

Han, X.-F., Laga, H., and Bennamoun, M. (2019). Image-based 3D object reconstruction: State-of-the-art and trends in the deep learning era. *IEEE Transac. Patt. Analy. Mach. Intell.* 43, 1578–1604. doi: 10.1109/TPAMI.2019.2954885

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., et al. (2018). BindsNET: A machine learning-oriented spiking neural networks library in Python. *Front. Neuroinform.* 12, 89. doi: 10.3389/fninf.2018.00089

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* (IEEE) 770–778. doi: 10.1109/CVPR.2016.90

Hinz, G., Chen, G., Aafaque, M., Rohrbein, F., Conradt, J., Bing, Z., et al. (2017). "Online multi-object tracking-by-clustering for intelligent transportation system with neuromorphic vision sensor," in *KI 2017: Advances in Artificial Intelligence: 40th Annual German Conference on AI, Dortmund* (Springer), 142–154.

Jiang, B., Wang, M., Gan, W., Wu, W., and Yan, J. (2019). "STM: Spatiotemporal and motion encoding for action recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision 2000–2009*. doi: 10.1109/ICCV.2019.00209

Khoei, M. A., Yousefzadeh, A., Pourtaherian, A., Moreira, O., and Tapson, J. (2020). "Sparnet: Sparse asynchronous neural network execution for energy efficient inference," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* 256–260. doi: 10.1109/AICAS48895.2020.9073827

Kugele, A., Pfeil, T., Pfeiffer, M., and Chicca, E. (2020). Efficient processing of spatio-temporal data streams with spiking neural networks. *Front. Neurosci.* 14, 439. doi: 10.3389/fnins.2020.00439

Lee, D., Lee, G., Kwon, D., Lee, S., and Kim, J. (2018). "Flexon: A flexible digital neuron for efficient spiking neural network simulations," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. doi: 10.1109/ISCA.2018.00032

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-DVS: an event-stream dataset for object classification. *Front. Neurosci.* 11, 309. doi: 10.3389/fnins.2017.00309

Maas, A. L., Daly, R. E., Pham, P. T., Dan, H., and Potts, C. (2011). "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* 142–150.

Massa, R., Marchisio, A., Martina, M., and Shafique, M. (2020). An efficient spiking neural network for recognizing gestures with a DVS camera on the Loihi neuromorphic processor. *arXiv preprint arXiv*:2006.09985. doi: 10.1109/IJCNN48605.2020.9207109

Materzynska, J., Berger, G., Bax, I., and Memisevic, R. (2019). "The jester dataset: A large-scale video dataset of human gestures," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. doi: 10.1109/ICCVW.2019.00349

Meng, Y., Panda, R., Lin, C. C., Sattigeri, P., Karlinsky, L., Saenko, K., et al. (2021). Adafuse: Adaptive temporal fusion network for efficient action recognition. *arXiv preprint arXiv:2102.05775*. doi: 10.1007/978-3-030-58571-6_6

MMCV-Contributors (2018). *MMCV: OpenMMLab computer vision foundation*. Available online at: https://github.com/open-mmlab/mmcv (accessed July 3, 2023).

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

Neimark, D., Bar, O., Zohar, M., and Asselmann, D. (2021). "Video transformer network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision* 3163–3172. doi: 10.1109/ICCVW54120.2021.00355

Rasmussen, D. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics* 17, 611–628. doi: 10.1007/s12021-019-09424-z

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Setio, A. A. A., Traverso, A., De Bel, T., Berens, M. S., Van Den Bogaard, C., Cerello, P., et al. (2017). Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: the luna16 challenge. *Med. Image Analy.* 42, 1–13. doi: 10.1016/j.media.2017.06.015

Shen, G., Zhao, D., and Zeng, Y. (2022). Backpropagation with biologically plausible spatiotemporal adjustment for training deep spiking neural networks. *Patterns* 3, 100522. doi: 10.1016/j.patter.2022.100522

Shen, W., Zhou, M., Yang, F., Yu, D., Dong, D., Yang, C., et al. (2017). Multi-crop convolutional neural networks for lung nodule malignancy suspiciousness classification. *Patt. Recogn.* 61, 663–673. doi: 10.1016/j.patcog.2016.05.029

Shi, Y., Li, H., Zhang, H., Wu, Z., and Ren, S. (2020). Accurate and efficient LIF-Nets for 3D detection and recognition. *IEEE Access* 8, 98562–98571. doi: 10.1109/ACCESS.2020.2995886

Simonyan, K., and Zisserman, A. (2014a). "Two-stream convolutional networks for action recognition in videos," in *Advance in Neural Information Processing Systems* 27.

Simonyan, K., and Zisserman, A. (2014b). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "HATS: Histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* 1731–1740. doi: 10.1109/CVPR.2018.00186

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). "Learning spatiotemporal features with 3D convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision* 4489–4497. doi: 10.1109/ICCV.2015.510

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). "Attention is all you need," in *Advances in Neural Information Processing Systems* 30.

Wang, C., Chen, X., Zhang, T., and Wu, S. (2022). BrainPy: a flexible, integrative, efficient, and extensible framework towards general-purpose brain dynamics programming. *bioRxiv* 2022–10. doi: 10.1101/2022.10.28.514024

Wang, Q., Zhang, Y., Yuan, J., and Lu, Y. (2019). "Space-time event clouds for gesture recognition: from RGB cameras to event cameras," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE) 1826–1835. doi: 10.1109/WACV.2019.00199

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence* 1311–1318. doi: 10.1609/aaai.v33i01.33011311

Wu, Y., Zhao, R., Zhu, J., Chen, F., Xu, M., Li, G., et al. (2022a). Brain-inspired global-local learning incorporated with neuromorphic computing. *Nat. Commun.* 13, 65. doi: 10.1038/s41467-021-27653-2

Wu, Z., Zhang, H., Lin, Y., Li, G., Wang, M., and Tang, Y. (2021). LIAF-Net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing. *IEEE Trans. Neur. Netw. Learn. Syst.* 33, 6249–6262. doi: 10.1109/TNNLS.2021.3073016

Wu, Z., Zhang, Z., Gao, H., Qin, J., Zhao, R., Zhao, G., et al. (2022b). Modeling learnable electrical synapse for high precision spatio-temporal recognition. *Neur. Netw.* 149, 184–194. doi: 10.1016/j.neunet.2022.02.006

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-C. (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems* 802–810.

Yan, X., Pang, J., Qi, H., Zhu, Y., Bai, C., Geng, X., et al. (2017). "Classification of lung nodule malignancy risk on computed tomography images using convolutional neural network: A comparison between 2D and 3D strategies," in *Computer Vision-ACCV 2016 Workshops: ACCV 2016 International Workshops* (Taipei, Taiwan: Springer) 91–101. doi: 10.1007/978-3-319-54526-4_7

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

Zhang, C., Zou, Y., Chen, G., and Gan, L. (2020a). PAN: Towards fast action recognition via learning persistence of appearance. *arXiv preprint arXiv:*2008.03462. doi: 10.1145/3343031.3350876

Zhang, W., Wang, J., and Lan, F. (2020b). Dynamic hand gesture recognition based on short-term sampling neural networks. *IEEE/CAA J. Autom. Sinica* 8, 110–120. doi: 10.1109/JAS.2020.1003465

Zhu, W., Liu, C., Fan, W., and Xie, X. (2018). "Deeplung: Deep 3D dual path nets for automated pulmonary nodule detection and classification," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE) 673–681. doi: 10.1109/WACV.2018.00079

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 07 August 2023
DOI 10.3389/fnins.2023.1223262

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Zhuo Zou,
Fudan University, China
Steffen Albrecht,
The University of Auckland, New Zealand
Arindam Basu,
City University of Hong Kong, Hong Kong SAR, China

*CORRESPONDENCE
Jiaxin Huang
✉ Jiaxin.Huang@infineon.com

RECEIVED 15 May 2023
ACCEPTED 13 July 2023
PUBLISHED 07 August 2023

CITATION
Huang J, Kelber F, Vogginger B, Liu C, Kreutz F, Gerhards P, Scholz D, Knobloch K and Mayr CG (2023) Efficient SNN multi-cores MAC array acceleration on SpiNNaker 2.
Front. Neurosci. 17:1223262.
doi: 10.3389/fnins.2023.1223262

Efficient SNN multi-cores MAC array acceleration on SpiNNaker 2

Jiaxin Huang[1]*, Florian Kelber[2], Bernhard Vogginger[2], Chen Liu[2], Felix Kreutz[1], Pascal Gerhards[1], Daniel Scholz[1], Klaus Knobloch[1] and Christian G. Mayr[2,3]

[1]Infineon Technologies Dresden, Dresden, Germany, [2]Highly-Parallel VLSI-Systems and Neuro-Microelectronics, Faculty of Electrical and Computer Engineering, Institute of Principles of Electrical and Electronic Engineering, Technische Universität Dresden, Dresden, Germany, [3]Centre for Tactile Internet with Human-in-the-Loop (CeTI), Cluster of Excellence, Technische Universität Dresden, Dresden, Germany

The potential low-energy feature of the spiking neural network (SNN) engages the attention of the AI community. Only CPU-involved SNN processing inevitably results in an inherently long temporal span in the cases of large models and massive datasets. This study introduces the MAC array, a parallel architecture on each processing element (PE) of SpiNNaker 2, into the computational process of SNN inference. Based on the work of single-core optimization algorithms, we investigate the parallel acceleration algorithms for collaborating with multi-core MAC arrays. The proposed Echelon Reorder model information densification algorithm, along with the adapted multi-core two-stage splitting and authorization deployment strategies, achieves efficient spatio-temporal load balancing and optimization performance. We evaluate the performance by benchmarking a wide range of constructed SNN models to research on the influence degree of different factors. We also benchmark with two actual SNN models (the gesture recognition model of the real-world application and balanced random cortex-like network from neuroscience) on the neuromorphic multi-core hardware SpiNNaker 2. The echelon optimization algorithm with mixed processors realizes 74.28% and 85.78% memory footprint of the original MAC calculation on these two models, respectively. The execution time of echelon algorithms using only MAC or mixed processors accounts for $\leq$24.56% of the serial ARM baseline. Accelerating SNN inference with algorithms in this study is essentially the general sparse matrix-matrix multiplication (SpGEMM) problem. This article explicitly expands the application field of the SpGEMM issue to SNN, developing novel SpGEMM optimization algorithms fitting the SNN feature and MAC array.

KEYWORDS

SpiNNaker 2, SNN, MAC array, SpGEMM, multi-core load balancing deployment

1. Introduction

Coupling spatial and temporal information, the SNN shows promise in simulating biologically related models more comprehensively and efficiently. The CPU-based system is widely used for simulating these brain-inspired neural networks by taking advantage of flexibility. However, the efficient input spike encoding way is still in the exploration stage, and a gap still exists between the current encoding efficiency and that of the human brain, which reduces the expected sparsity of the input signal and extends the CPU running time. Moreover, to accommodate the serial operation mechanism, the model needs to introduce additional information when deployed to the hardware, such as the storage address of neurons, extra memory occupation owing to non-equivalent connections, and intermediate

state storage buffers. This not only burdens the memory space but also inevitably requires more time to execute the corresponding pre- and post-neuron matching algorithm for information transfer and neural update, which is detrimental to the operation of real-time SNN inference.

To address these issues caused by pure CPU systems, we introduced the parallel computing concept into SNN inference. The feasibility of parallel architecture processing SNN lies in the neurons of SNN being typically governed by the same type of equations (Yavuz et al., 2016). As a result, the single-instruction-multiple-data (SIMD) architecture of MAC fits SNN calculation. This study targets a wholly parallel calculation based on the more efficient matrix parallelism. We emulate the SNN inference on SpiNNaker 2 (Mayr et al., 2019), which integrates the MAC array in each processing element (PE). The parallelism of this integrated hardware component has the potential of speeding up SNN inference in a sufficiently parallel manner. Nevertheless, there are two challenges to tackle:

- `Memory alignment`: The memory alignment for catering to the MAC array architecture triggers an issue of data volume surges, blocking the possibility of deploying more neurons and synapses on limited hardware resources.
- `Multi-core distribution`: The unconsidered multi-core distribution of the large-scale model can differentiate the spatial-temporal overhead among activated PEs, wasting the resources in space and time and affecting the performance of applications with strict requirements.

This study addresses these two challenges by lossless densifying the memory-aligned model information and splitting matrix multiplication operands into multiple PEs in a spatial-temporal load-balancing way.

Essentially, accelerating SNN inference with our algorithms is the SpGEMM problem, as explained in Section 2.2. SpGEMM is very popular in high-performance computing, mainly used in algebra and graph analysis (Gao et al., 2020). The vast majority of the relevant studies, such as Davis (2018), Zhang et al. (2020), and An and Çatalyürek (2021), are based on the "row-wise" algorithm proposed by Gustavson (1978), also known as compressed sparse row format (CSR) or Yale sparse matrix format. This traditional algorithm is unsuitable for using MAC array accelerating SNN, so we propose a brand-new optimization algorithm set, which can accelerate the SNN processing when alleviating the ineffective memory footprint. This algorithm set, consisting of four algorithms up to now, provides an alternative to the traditional method for solving the SpGEMM problem. To the best of our best knowledge, our work is the first to build a bridge between the concept of SpGEMM and SNN, expand the application field of SpGEMM to SNN, and tackle the SNN inference using the MAC array with new SpGEMM algorithms.

As a follow-up to our previous study that states three algorithms of information densification (Huang et al., 2023), this study proposes Echelon Reorder, filling in the unoptimized aspects of that work, completing the optimization algorithm set to fully resist the data sparsity caused by the SNN characteristics and fixed MAC array hardware structure. The corresponding splitting and deployment strategies proposed in this study extend the application range of the whole optimization algorithm set from single PE to multi-core, enabling accelerating the larger model on SpiNNaker 2 effectively. Furthermore, the compact splitting strategy fully uses each PE's memory resource, paving the way for the subsequent high-performance multiple tasks deployment on this multi-core neuromorphic platform.

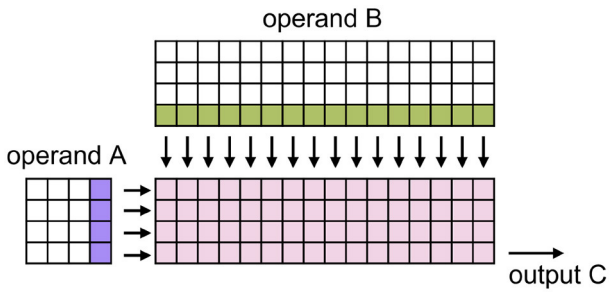


FIGURE 1
Schematic of the MAC array (Huang et al., 2023). Pink squares express 4 × 16 MAC calculation units. In each clock cycle, the SpiNNaker 2 bus system can convey four values from operand A and 16 from operand B and feed them into the MAC array as the arrow indicates for executing matrix multiplication. To deploy this structure, we should align the row and column number of operand A to a multiple of 4 and operand B to a multiple of 4 and 16.

This study briefly introduces the hardware and software cornerstones in Section 2. Then, based on them, we elaborate on the Echelon Reorder algorithm for weight and input pure and mixture processor splitting strategies and also multi-core role-based SNN model deployment in Section 3. Next, Section 4 evaluates the performance of this proposed processing chain. Finally, we conclude this article in Section 5.

2. Prerequisite

This section provides the hardware and software foundations for the next section concerning the MAC array architecture of SpiNNaker 2 and the stacked matrix-multiplication operands essential for accelerating SNN inference.

2.1. MAC array

SpiNNaker 2 is a neuromorphic multi-core system. Each core contains 64 MAC units in a 4 × 16 layout (Yan et al., 2021; Zeinolabedin et al., 2022), which we call the MAC array. For executing matrix multiplication, operands are supposed to be memory aligned, as Figure 1 illustrates. The alignment shapes originate from the fixed hardware architecture of the MAC array and data access bandwidth of the SpiNNaker 2 system. The precision of the operands could be 8 bits or 16 bits. For output precision, 8 bits, 16 bits, and 32 bits can be configured.

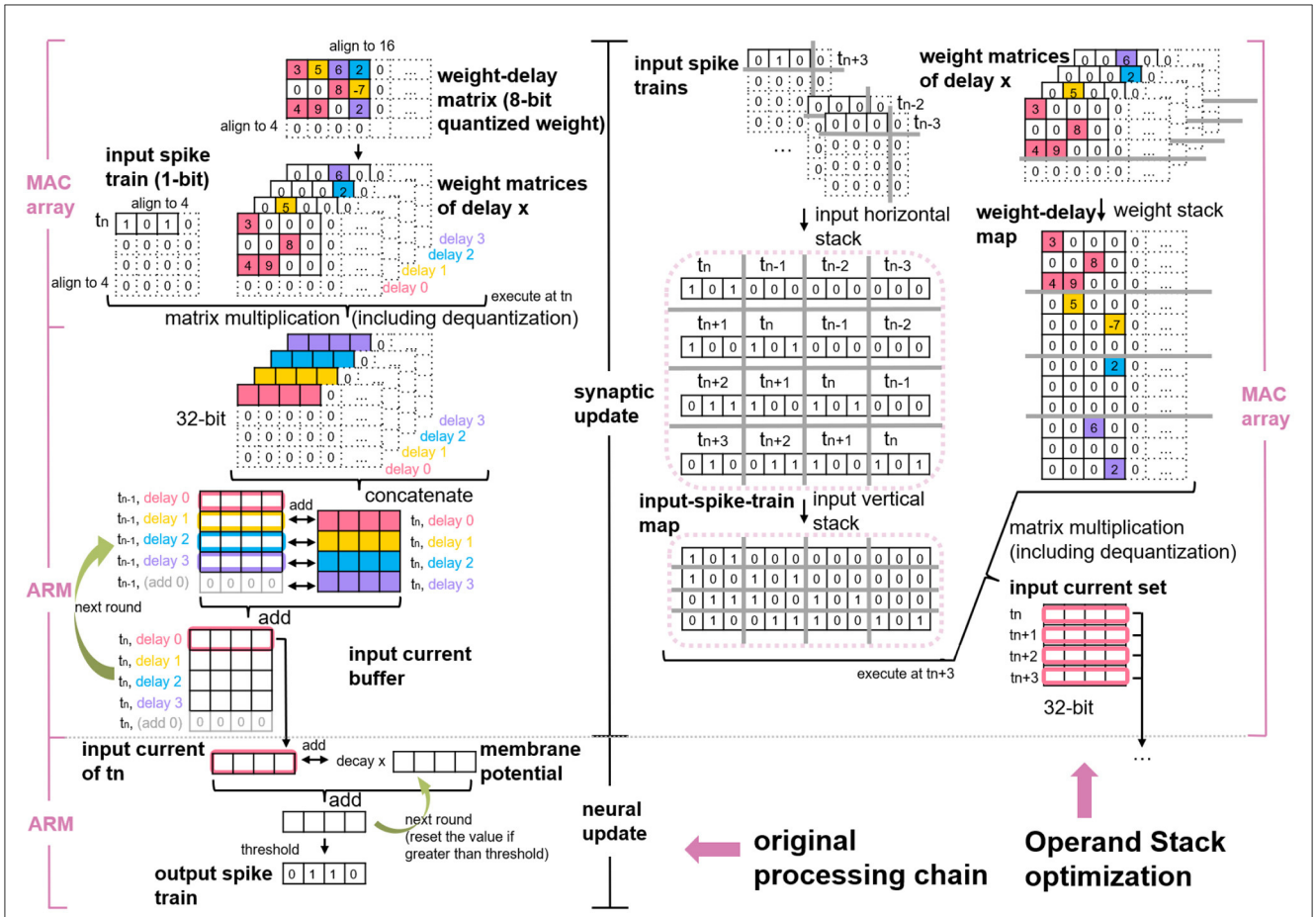


FIGURE 2
Original processing chain of MAC array calculating SNN **(left)** and the schematic of Operand Stack optimization algorithm **(right)** (Huang et al., 2023).

2.2. Operand Stack

Conducting SNN inference generally consists of two consecutive steps: synaptic processing and neural update. The following equation defines the dynamics of the arguably most prominent neuron model in the brain, that is, the leaky integrate-and-fire (LIF):

$$V_j^{t+1} = \Sigma_i W_{ji} x_i^{t-d(j,i)} + \alpha V_j^t - z_j^t V_{th} \quad (1)$$

In the synaptic processing step, weights that connect pre-neuron i and post-neuron j are summed if the spikes arrive at the post-neuron after traveling across the synapse for time interval $d(j, i)$. Then, in the neural update step, the membrane potential decays by factor α (equals to e^{1/τ_m}, τ_m denotes the membrane time constant) and is updated by checking the neuron states z_j^t at time t. If the sum of the first two terms exceeds the threshold V_{th}, neuron states are set to 1 and neuron spikes, otherwise 0. This equation refers to (Bellec et al., 2020) and its supplementary, with factor $1-\alpha$ removed. Unlike the original equation, the input and recurrent synaptic processing share the same term.

Because of the high data precision requirement of the neural update, the MAC array primarily contributes to accelerating the synaptic processing of the SNN inference. As with the serial synaptic processing mentioned by Rhodes et al. (2018), the parallel synaptic processing also contains processing input spikes and advancing the input current buffer of each delay, as shown on the left side of Figure 2. To be specific, first, the memory-aligned weight-delay matrix is divided into several weight matrices, each of which has the same delay attribute. Then weight matrices is transmitted to the MAC array one by one serving as the operand B and simultaneously conveying the memory-aligned input spike train to MAC as the operand A. Finally, the MAC calculated results is added to the input current buffer to update the input current of each delay. Here, the "delay," or "synaptic delay" precisely, is the time for conducting a signal across a synapse, that is, the interval between the arrival of the spike and the start of the membrane potential.

The delay stack algorithm proposed in our previous study (Huang et al., 2023) simplifies these conventional synaptic processing steps to only one step (matrix multiplication), so there is no need to consider the weight matrix division and the input current accumulation. As depicted on the right side of Figure 2, this algorithm stacks sequential input spike trains of t_{n-3} to t_{n+3} into an input-spike-train map and stacks weight matrices along the delay into a weight-delay map. These two maps act as the new operands for matrix multiplication on the MAC array. SNN features sparse input to mimic the working mechanism in the mammalian brain but decouples weight sparsity. By applying delay stack, the weight matrix is divided into multiple sparse matrices, and the merged

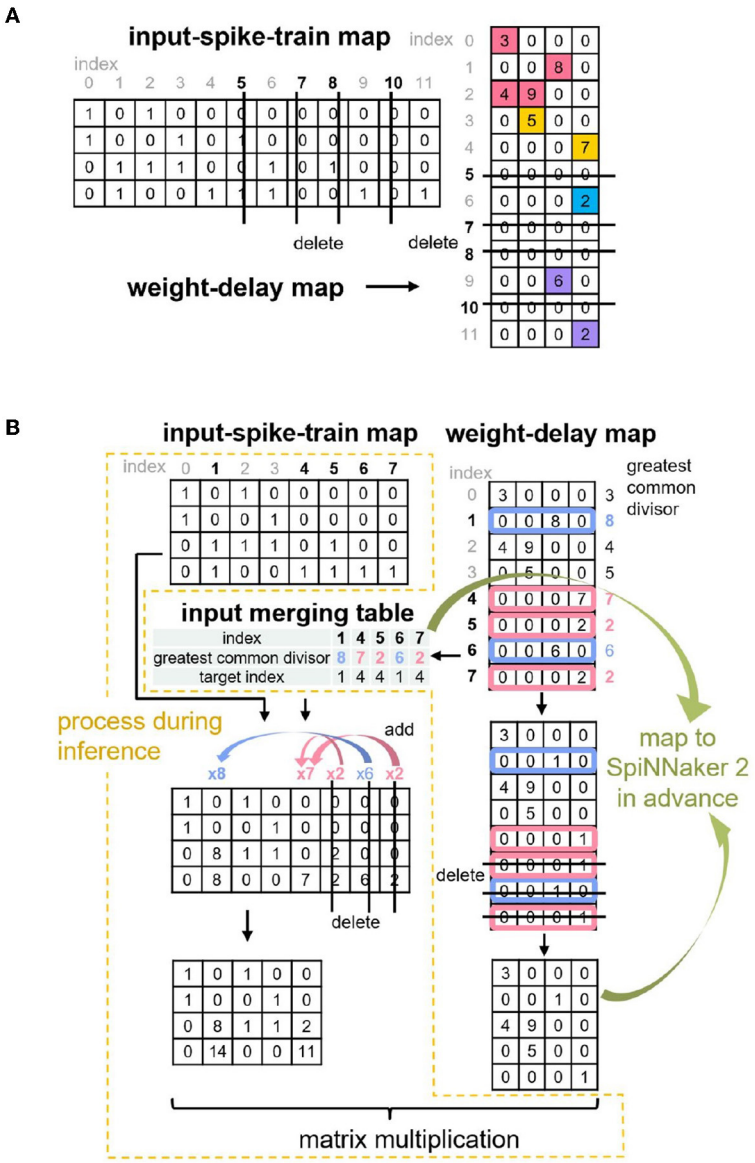


FIGURE 3
Two optimization algorithms proposed in Huang et al. (2023): **(A)** Zero Elimination **(B)** Proportion Merger.

weight-delay map exhibits sparsity as a result. Thus, the issue of accelerating SNN inference with MAC array is converted into efficient multi-core processing SpGEMM problem.

Considering the limited SRAM space on each PE of SpiNNaker 2 and the sparsity of the merged weight-delay map, our last study (Huang et al., 2023) further applies the Zero Elimination and Proportion Merger algorithms to shrink the size of the matrix-multiplication operands, as shown in Figure 3. Basically, the Zero Elimination algorithm removes the rows with all zero values in the weight-delay map against the operand sparsity, and it also removes the corresponding columns in the input-spike-train map to guarantee the result correctness of the matrix multiplication. The Proportion Merger merges rows that is proportional to each other to only one row for weight-delay map and records the proportional values (greatest common divisor), which contribute to pre-processing the input-spike-train map at runtime. This algorithm essentially migrates some weight-delay information into input operand, addressing the accuracy mismatch problem of SNN input (1-bit) and MAC operand requirement (8-bit/16-bit) and improving the memory utilization.

These three algorithms from our last study (Huang et al., 2023) tackle or alleviate most of the memory issues caused by SNN characters and memory alignment, except memory alignment alongside the column of operands B and C, as indicated in Table 1. In other words, if only deploying these three algorithms, the part from the 5th to 16th column of the weight-delay map in Figure 2 that has all values equal to zero has to be saved on SRAM of PE for matrix-multiplication calculation. According to the MAC array working mechanism mentioned in Section 2.1, the output (operand C) also requires the same number of column reserved for storing

TABLE 1 Correspondence between memory issues and optimization algorithms.

Memory issues \ Optimization algorithms			Operand Stack	Zero Elimination	Proportion Merger	Echelon Reorder
SNN characters	Sparse operand A			★	★	
	Accuracy mismatch				★	
MAC structure	Operand A	Row	★			
		Column	★			
	Operand B	Row	★			
		Column				★
	operand C	Row	★			
		Column				★
Generated during optimization	Sparse operand B			★	★	★

The first row lists four optimization strategies, and the first and second column state the memory issues to be solved when we want to accelerate SNN inference with MAC array. Stars mark which optimization algorithms can solve or alleviate which memory issues. The first three algorithms proposed in our previous study (Huang et al., 2023) ignore the memory alignment problems of column of operand B and C, highlighted with pink color. The Echelon Reorder algorithm proposed in this study can solve them.

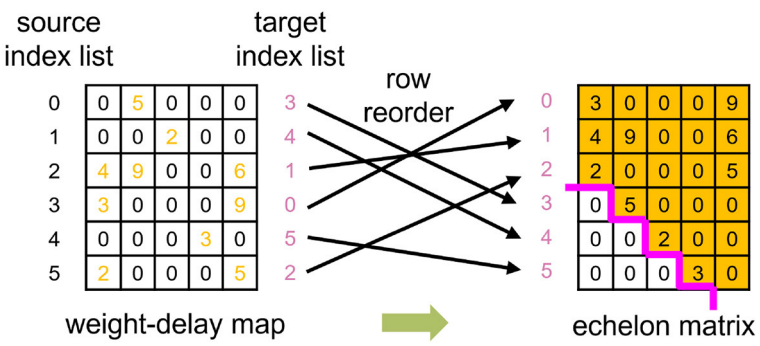


FIGURE 4
An example of utilizing the Echelon Reorder algorithm, which reorders the row of weight-delay map to form the echelon matrix that the left corner has only zeros.

matrix-multiplication results, which is unfriendly to the limited memory space. In addition, the previous study does not discuss multi-core MAC arrays collaborating on processing a large SNN model. We address these two issues in the following section.

3. Method

This section elaborates on each part of the SNN multi-core MAC array acceleration processing chain, incorporating the information densification algorithm for the weight-delay map and input-spike-train map, pure MAC and mixed processor splitting strategies, as well as the multi-core deployment.

3.1. Echelon Reorder

3.1.1. Weight-delay map

The Operand Stack algorithm from Huang et al. (2023) dilutes the original weight-delay matrix and results in a sparse weight-delay map. Our proposed Echelon Reorder algorithm takes advantage of this sparsity to isolate the meaningful weights for weight-delay map, as shown in Figure 4 and Algorithm 1. In this algorithm, we reorder the rows of the weight-delay map to form an echelon matrix, of which the lower left part has all values of zero so that the upper right part has a denser distribution of non-zero weights than the weight-delay map. The storage performance and computing power improvement are foreseeable if only the upper right part is stored and calculated. We will discuss the specific storage approach in Sections 3.2 and 3.3.

3.1.2. Input-spike-train map

To guarantee the correctness of the matrix multiplication result, it is necessary to adjust the operand A (input-spike-train map) according to the modification of operand B (weight-delay map) which is discussed in Section 3.1.1. To be specific, we record how the row of operand B is reordered and apply it to the column reorder of the operand A. Unlike the operand B, the operand A is unknown in advance because there is no way to predict the

```
Require: weight-delay map WD_map of size row × col
Ensure: echelon matrix E_matrix, echelon matrix index
  list tgt_idx_list
  /* this algorithm reorder the rows of WD_map to
  generate E_matrix */
  /* execute this algorithm on PC before inference,
  and load the result E_matrix to PE of SpiNNaker 2 as
  operand B */

  E_matrix_row_count ← 0
  for j ← 0 to (col − 1) do
    for i ← 0 to (row − 1) do
      if WD_map[i][j]! = 0 then
        E_matrix[E_matrix_row_count] ← WD_map[i][:]
        E_matrix_row_count+ = 1
        add i to tgt_idx_list
      end if
    end for
  end for
```

Algorithm 1. Echelon Reorder algorithm.

input spike status, so we cannot reorder operand A offline and have to adjust the operand A when executing the SNN inference. This adjustment can be achieved on the host PC with Python or on SpiNNaker 2 with C language. To make a fair comparison with the pure ARM baseline in Section 4 and to extend the capability of SpiNNaker 2 of directly handling the spikes from the sensor peripheral, we propose the Circle Reverse algorithm (Algorithm 2) to online real-time process the input-spike-train map on SpiNNaker 2. It consists of two steps: find circles and reverse circles.

Find circles: During the reorder process of the operand B, we get the target index list that corresponds to the source index list, that is, which source row of the operand B should be placed in which target row in the generated echelon matrix, as presented in Figure 4. Suppose the current target index is taken as the next source index, we can retrieve the next target index iteratively until finding out the target index that equals the start source index. The indices found in this process can form a circle. All the indices of a weight-delay map can be represented by several circles. In our example, two circles are found, as demonstrated on the left side of Figure 5.

Reverse circles: Now we consider adjustment of the operand A (input-spike-train map). If we directly move columns of operand A based on the order given by original circles, the previous column overwrites the current column, and then the current column cannot assign the correct values to the next column. To solve this issue, we reverse the index order of each original circle and preserve the start column before executing the column movement for the operand A, as shown on the right side of Figure 5. Then we assign the preserved start column to the column at which the start column points. In this process, only a little extra memory is required to preserve the start column instead of a whole space with the same size as the original input-spike-train map. This algorithm does not involve the sort and search algorithms. Thus, the runtime grows linearly with the number of columns, and the time complexity is O(n).

```
Require: weight-delay map index list src_idx_list,
  echelon matrix index list tgt_idx_list
Ensure: reversed_circle_list
  /* this algorithm calculates reversed_circle_list needed
  for the input-spike-train map online reorder */
  /* execute this algorithm on PC before inference,
  and load the result reversed_circle_list to PE of
  SpiNNaker 2. It processes input data during
  inference to generate operand A */

  /* step 1: find circles */
  /* input: src_idx_list, tgt_idx_list */
  /* output: circles_list, circle_count */
  circle_count ← 0
  repeat
    start_src_index ← one element from src_idx_list
    src_index ← start_src_index
    delete src_index from src_idx_list
    add src_index to circles_list[circle_count]
    repeat
      tgt_index ← tgt_idx_list[src_index]
      src_index ← tgt_index
      delete src_index from src_idx_list
      add src_index to circles_list[circle_count]
    until tgt_index == start_src_index
    circle_count+ = 1
  until no element in src_idx_list

  /* step 2: reverse circles */
  /* input: output of step1, that is: circles_list,
  circle_count */
  /* output: reversed_circle_list */
  for i ← 0 to circle_count do
    reversed_circles_list ← reverse the elements of
  circles_list[i]
  end for
```

Algorithm 2. Circle Reverse algorithm.

3.2. Multi-core two-stage splitting

3.2.1. Pure MAC

Considering the MAC array exclusively supporting the acceleration of the rectangular matrix, we employed a set of rectangles to enclose all the meaningful data and as few zeros as possible. The length of the rectangle is a consecutive integer (1, 2, 3, etc.) multiple of 16, and the width is an integer multiple of 4. The multipliers derive from the hardware characteristic of the MAC array of SpiNNaker 2. Using the alignment splitting algorithm, we obtained a set of rectangles with the smallest total area, as Figure 6A(c) indicates. *m* in this figure represents the remainder of dividing the column number of the echelon matrix by that of MAC array (16). The data contained in this set of rectangles consume the least memory resources when deploying on SpiNNaker 2.

Now, length and width of the rectangles meet the requirements of the MAC calculation, and its time to discuss the load balancing issue. The amount of data outlined by rectangles varies. All data

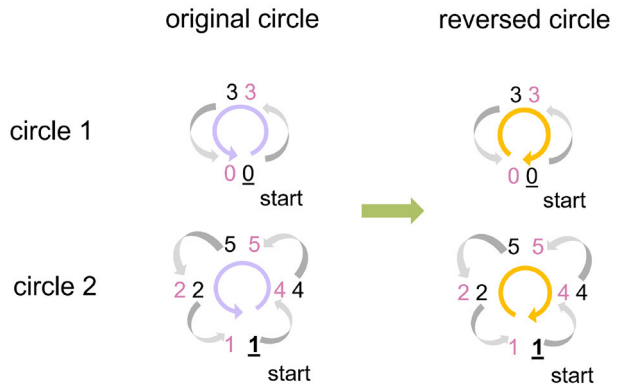


FIGURE 5
An example of utilizing Circle Reverse algorithm. The indices of the weight-delay map in Figure 4 has two circles with index order 0→ 3→ 0 and 1→ 4→ 5→ 2→ 1, arrows pointing at the direction of the row movement. To avoid the row overwrite (row of weight-delay map, equivalent to column of input-spike-train map here) and save memory footprint, we reverse the index order to: 0← 3← 0 and 1← 2← 5← 4← 1. Preserve the start point of each circle (0 and 1, respectively) beforehand and assign the preserved values to the row that the start row points at (3 and 4, respectively).

in some rectangles might occupy little memory if we simply distribute each small rectangle to one PE without combination. In contrast, all data in other rectangles may be far beyond the maximum available SRAM space of one PE. Even if the data in the largest rectangle barely fit into one PE, the unbalanced weight loading of the rectangle set among multiple PEs can prolong the overall processing time, and partial computing power of the core with a low weight load is wasted. Therefore, based on the set of rectangles obtained in the above steps, we execute the core splitting algorithm to achieve a spatial-temporal load balancing deployment on multiple cores. This step splits the rectangles obtained by alignment splitting into more and smaller rectangles horizontally and then divides them into several groups with an equal amount of weight values, as Figure 6A(e) shows. The number of the group is the number of PEs to be activated.

Equally distributing weights across multiple cores has many possibilities. Here, we put weights into as few PEs as possible to fully utilize the resources of each core. We do this for two reasons: on the one hand, matrix multiplication with MAC array has a considerable time advantage, and the marginal utility of dividing into more cores is tiny; on the other hand, the full utilization of each core is also conducive to simultaneously deploying and executing more tasks on multi-core SpiNNaker 2 platform in future.

3.2.2. MAC and ARM mixture

By observing the biggest rectangle in Figure 6A(c), we find that there is still a relatively large area with all values equal to zero, arising from memory alignment alongside the column of operand B. The concrete size of this area is subject to m, the remainder dividing the column number of the echelon matrix by 16. When m is small, these zero values employed as placeholders occupy a large memory space. We improve this situation by abstracting only the meaningful data, as Figure 6B(b) illustrates, and conducting matrix multiplication for this part with the ARM core.

As for the corresponding core splitting strategy that matches this mixture alignment splitting, we figured out the number of required PEs with the approach mentioned above and outlined the core splitting rectangles for the blue and pink marked area of Figure 6B separately. For example, we needed four PEs, so we split the whole area marked with blue into four groups equally and do the same for the pink. Later, in the deployment step, we saved one split rectangular group for MAC calculation and one for ARM into one PE. Then all the activated PEs executed matrix multiplication by leveraging the local MAC array and ARM core. Finally, the results from all activated PEs were converged to get the synaptic processing result.

This optimization eliminates the extra memory overhead originating from necessary memory alignment alongside the columns of operands B and C. When m is small, or the number of source neurons (corresponds to the row number of echelon matrix) is large, the optimization is particularly effective.

3.2.3. Pure or mixed?

The variable m represents a number ranging from 1 to 16 in various models. Therefore, it is necessary to analyze the influence of m on processor selection quantitatively. In the pink outlined area in Figure 6B(a), the memory cost of the pure MAC approach is independent of m. In contrast, the MAC and ARM mixture approach consumes only 6.25% of the memory required by pure MAC in the extreme case of m being 1, as shown in Figure 7A. With m climbs, the memory gap gradually narrows until it disappears when m reaches the maximum value of 16. As for the time comparison, we found that pure MAC outperforms ARM of the mixed approach except for several cases when m and row number are pretty small, according to Figure 7B. Consequently, if a model has a high requirement of the real-time reaction and fewer memory constraints, pure MAC is a better option in the vast majority of cases; otherwise, MAC and ARM mixture outperforms.

3.3. Multi-core authorization deployment

After the splitting process, we discuss how to deploy the split echelon matrix and where to process the original input-spike-train map. The MAC array of SpiNNaker 2 supports reading operands from other PEs. Based on this feature, we authorize the core that reads reversed input data from another PE and preloads the split weight rectangular group as the "Subordinate PE." The PE that provides reversed input data acts as the "Dominant PE." Figure 8 illustrates the multi-core authorization result and the whole processing chain. First, the Dominant PE (Core 0) receives the original input spikes, generates the reversed input-spike-train map, and waits for the reading request from Subordinate PEs. Then, the Subordinate PE reads the corresponding reversed input data and performs pure MAC operation or mixture calculation. The generated synaptic

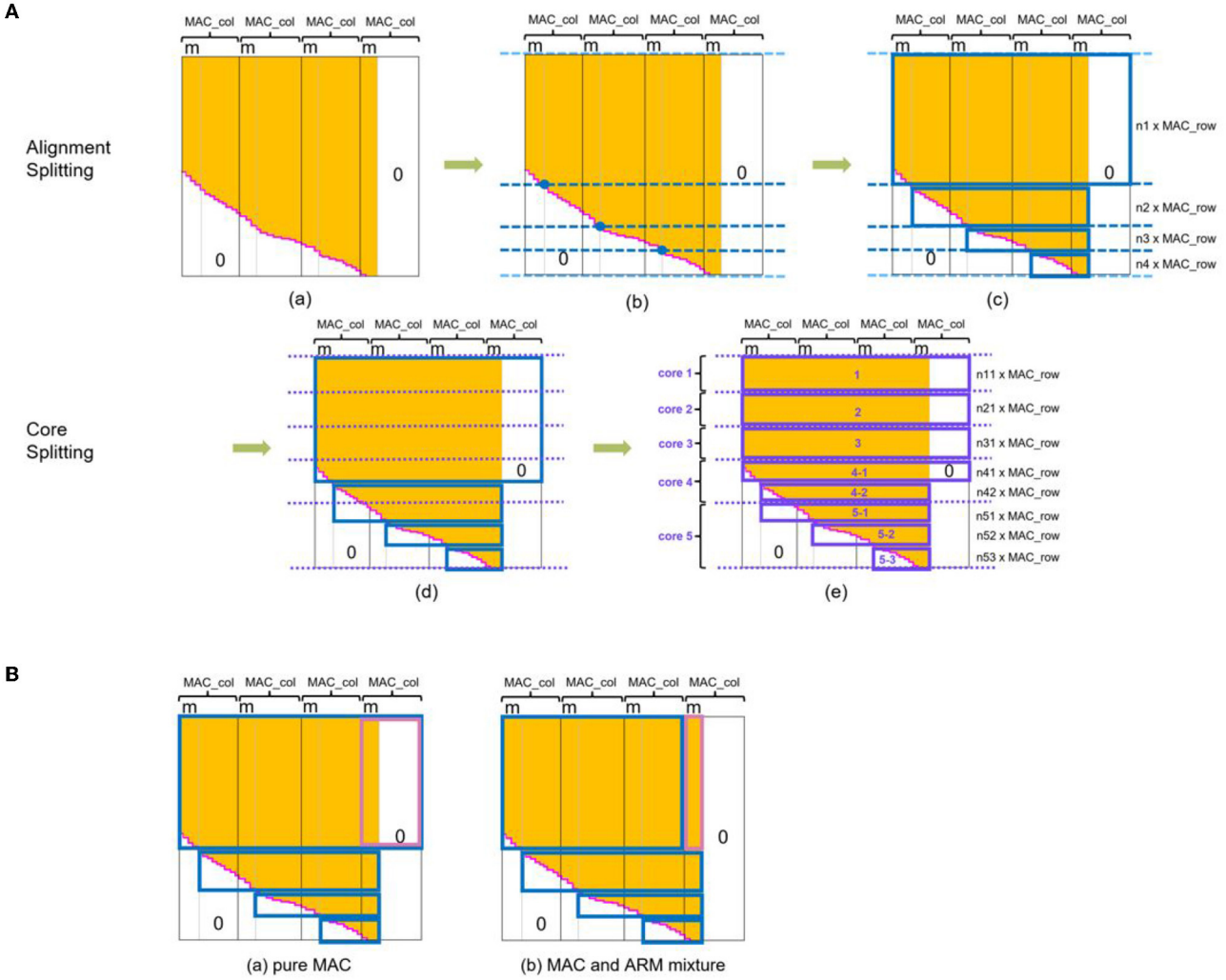


FIGURE 6
Multi-core splitting. **(A)** Two-stage splitting algorithm. The first splitting stage (alignment splitting) generates a set of rectangles that meet the memory alignment requirement from MAC array hardware structure, i.e., the length of the rectangles is a multiply of MAC array column number (16 for SpiNNaker 2) and the width of MAC array row number (4). The second splitting stage (core splitting) further disassembles and reorganizes these rectangles according to the available storage and then deploy them to multiply cores in a balanced loading way. Algorithm details (a) signifies the memory-aligned echelon matrix. For alignment splitting, first we find the intersection of the gray vertical line and rose red echelon line and determine the dark blue horizontal dividing lines in (b). In addition, add two light blue lines on the first and last rows. Starting from the second line, shift the line downwards until meeting the requirement that the number of rows between two adjacent lines is the multiply of 4. Based on them, we can outline the smallest rectangular set in (c). In the core splitting process, we divide the sum of the memory of all data in the dark blue rectangles by the available memory per PE for getting the number of PEs (supposing 5 in this example). Purple lines in (d) cut the memory into equal five portions at first, and a shift downwards is also necessary if two adjacent lines are not the multiply of 4. Purple rectangles surround the area as (e) depicts. Finally, we can save weights in these rectangles into corresponding PEs for direct MAC calculating with no need for further data format adjustment. **(B)** Two kinds of splitting approaches for echelon matrix. We store the weights enclosed by colored borders into PE. Approach of (a) fits pure MAC acceleration and has the speed advantage. (b) Deploy mixed processors to execute matrix multiplication. The weights outlined in blue utilize MAC array, while the pink area, which is different from (a), uses ARM processor. This approach consumes more running time but less memory footprint. The white area represents zero values, the orange area marks denser weights, and the rose red echelon line is the dividing line of them. *m* represents the remainder of dividing the column number of the echelon matrix by that of MAC array (16). *n* in (c) and (e) of subfigure **(A)** is the non-zero integer.

processing results are eventually accumulated and written back to the Dominant PE, serving as the input for the subsequent neural update step.

4. Experiment and result

To evaluate the performance of the information densification and splitting algorithms and the feasibility of the deployment strategy from Section 3, we benchmarked it with constructed and actual SNN models in this section.

4.1. Constructed SNN models

The optimization performance of our proposed approaches relies on the following five factors: delay range, number of pre-neurons, number of post-neurons, weight connection

density, and the selection of splitting strategy. The first four factors are determined by the SNN model itself. Thus, we constructed multiple SNN models based on various combinations of these four factors and explored their influence on spatial and temporal performance. To focus on the scope of our proposed approaches, we experimented on the synaptic processing part of SNN.

Figure 9 illustrates the comparison of the memory optimization rate among 30, 31, and 32 post-neurons. We define memory optimization rate as the ratio of the number of uncalculated weights to the number of memory aligned weights, i.e., the ratio of the area that is not enclosed by rectangles to the total area in Figure 6B. Figure 9A shows that the memory optimization effect gets better with the increase of delay range. Moreover, the two echelon algorithms perform better in memory cost when the weight operand is getting more sparse, and the more sparse the weight operand, the more obvious the effect. In addition, the mixed splitting approach always performs better than or equal to the pure MAC approach. This experimental result is consistent with the analysis in Section 3.2.3. The performance difference between these two splitting approaches depends on the remainder of post-neurons divided by 16, that is, the value m in Figure 6. As the remainder grows, the performance difference decreases. Until the remainder reaches 16 (back to 0), there is no difference between the two methods.

In addition to the impact of the remainder of post-neurons divided by 16 on the spatial performance difference between the two methods, the change of the post-neuron number itself also affects the memory optimization rate. Figure 9B demonstrates a 3D plot with the projections of the contours, highlighting the relation of memory optimization rate and two factors: the number of pre-neurons and post-neurons. Observing the contour on the back "wall," we find that the memory optimization rate fluctuating declines with a period of 16 with the post-neurons increasing. The increase of post-neurons brings the reduction of the optimization rate, which surges at the point where the remainder returns from 15 to 0. The reason is that the post-neuron number at this point exactly adapts to the hardware structure of the MAC array that no memory alignment is required. The projected contour on the left "wall" is almost parallel to the pre-neurons axis after the initial unstable status, implying the independence of pre-neuron number and memory optimization rate. The following formula briefly summarizes the relationship between memory optimization rate and factors:

$$r_{opt} \lesssim d, \frac{1}{p}, \frac{1}{n_{post}} \tag{2}$$

r_{opt} represents the memory optimization rate, which is approximately directly proportional to the delay range d, and inversely to weight connection density p and number of post-neurons n_{post}. The use of the approximately proportional symbol $\lesssim$ is intended to qualitatively show the positive and negative relationship of the variables before and after it. The relation details, such as the aforementioned "the more sparse the weight operand, the more obvious the effect" and "fluctuating declines with a period of 16," are not reflected in this equation.

Although the memory optimization rate is unrestricted by the pre-neuron number, the impact of the pre-neuron number on SNN inference performance is mainly reflected in the running time, which basically consists of input data pre-processing (Circle Reverse algorithm elaborated in Section 3.1.2) and synaptic processing. The synaptic processing is accelerated by the MAC array, and the input data pre-processing is only calculated by ARM. Therefore, the input data pre-processing consumes most of the temporal resources. The smaller the number of input neurons (i.e., pre-neurons), the better the algorithms proposed in this study performs in execution time. If only considering the synaptic processing part, the temporal performance of the mixed echelon approach also has an intense dependence on the number of pre-neurons. The mixed echelon algorithm sacrifices some runtime in exchange for a lower memory footprint, and this time is positively correlated with the number of pre-neurons.

4.2. Actual SNN models

We selected two actual benchmarks for evaluation of our proposed approaches: an application model from the real scenario (radar gesture recognition SNN model) and a classic structural modeling in neuroscience (balanced random cortex-like network). The experiments compared serial ARM, original MAC, pure MAC echelon, and MAC and ARM mixture approaches regarding spatial-temporal performance. The data measurement concentrates on the part where the above four approaches behave differently during SNN inference, that is, synaptic processing. As for the multi-core mapping and deployment, we evenly split the weight operand into multiple adjacent PEs and fully utilized each PE's available memory resources (120 KB in our configuration) in conjunction with other necessary data (split input, temporary output, and input current buffer) for serial ARM and original MAC. When a PE can accommodate not less than one weight matrix of one delay, we ensure the integrity of a computational unit, i.e., one weight matrix of one delay, to avoid introducing additional result fusion time. For example, if a kernel can hold 2.5 weight matrices of one delay, we assigned pure MAC echelon and mixture approaches that adopted supporting strategies from Section 3.2 and 3.3.

We adopted the following method to calculate memory cost and measure the execution time of the multi-core cooperation system:

- `Two baselines`: The memory cost of serial ARM and original MAC comes from the input placeholder, weight, output placeholder, and input current buffer in all activated PEs. Note that the input and weight operands are supposed to be memory aligned in advance. The execution time is comprised of matrix multiplication, output merging from different PEs and input current buffer update.
- `Two echelon approaches`: The memory cost consists of the footprint of the Dominant PE (input placeholder, reversed order, and output) and Subordinate PEs (weight cost and temporary output). In addition to the matrix-multiplication execution time (MAC or mixture processors) and the time of accumulating output into Dominant PE,

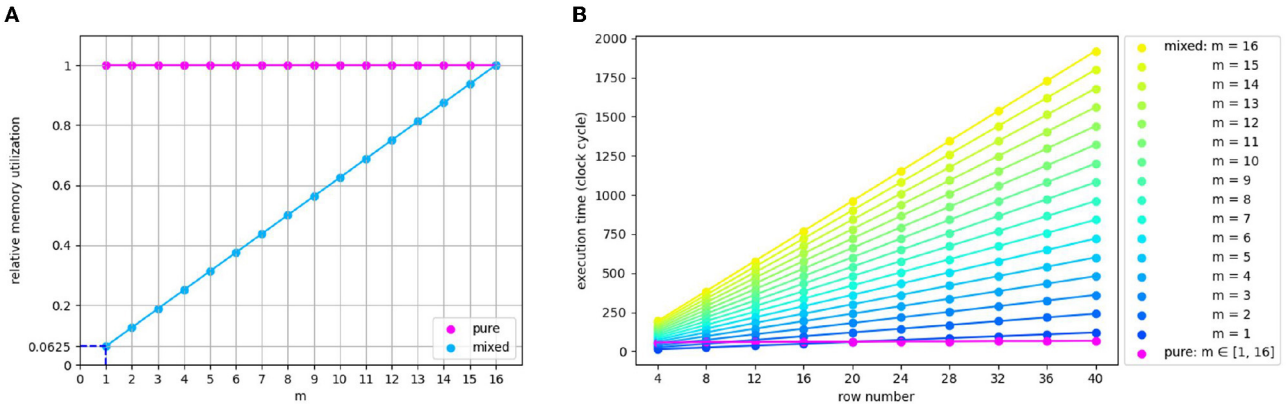


FIGURE 7
(A) The memory usage comparison of pure and mixed processor approaches. **(B)** Synaptic processing execution time comparison of pure MAC array and ARM part of the mixed approach, regarding the part that has difference when adopting the pure or mixed approach. Pink rectangles in Figure 6B mark this part. *m* represents the remainder of dividing the column number of the echelon matrix by that of MAC array (16).

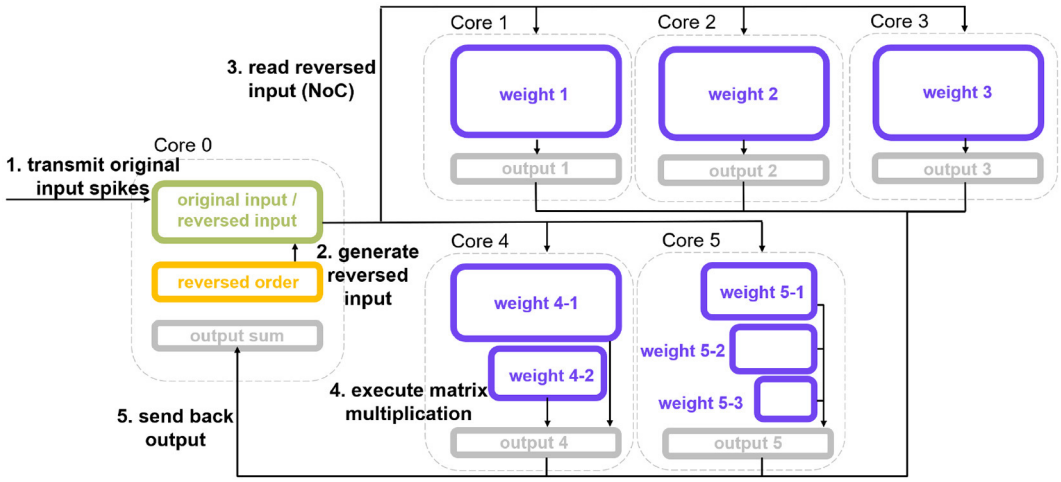


FIGURE 8
Multi-core authorization result and SNN multi-core MAC array acceleration processing chain. Core 0 is the "Dominant PE," and Core 1 to Core 5 are "Subordinate PEs." The weights deployment corresponds to the splitting outcome of Figure 6A, i.e., pure MAC splitting. This processing chain is also applicable to the outcome of MAC and ARM mixture splitting.

the execution time also incorporates the input reorder time to guarantee a fair comparison with serial ARM and the original MAC baseline. Since each calculation obtains the same number of results as the delay number instead of only one result of the serial ARM and original MAC approach as Figure 2 shows, we divide the tested total execution time by the delay number to get the processing time per frame.

4.2.1. Application model from the real scenario: radar gesture recognition SNN model

Gesture recognition is an important and active area of AI research, with relevant models and hardware deployment acting as the fundamental verification unit for more sophisticated real-world scenarios. For the first benchmark, we set up the radar-based SNN gesture recognition model with 2,048 input neurons, 20 hidden neurons, four output neurons, and four delays. This model is similar to the study in Gerhards et al. (2022) and Huang et al. (2022a,b) but introduces the concept of delay. We train this benchmarking model with our own collected radar dataset mentioned in Kreutz et al. (2021), involving three directional gestures (left, right, and push) and one environmental reference (random gesture or background noise). The experiment refers to processing synapses between the input layer and hidden layer marked with light blue in the gesture recognition model on the left top of Figure 10.

For the Echelon Reorder optimization algorithm with only MAC as the processor, the densified weight-delay map (i.e., echelon matrix) can be split and deployed into two Subordinate PEs as depicted on the right part of Figure 10A. One Dominant PE has enough space to save the input-spike-train map and the reversed order. The mixed echelon algorithm also consumes two

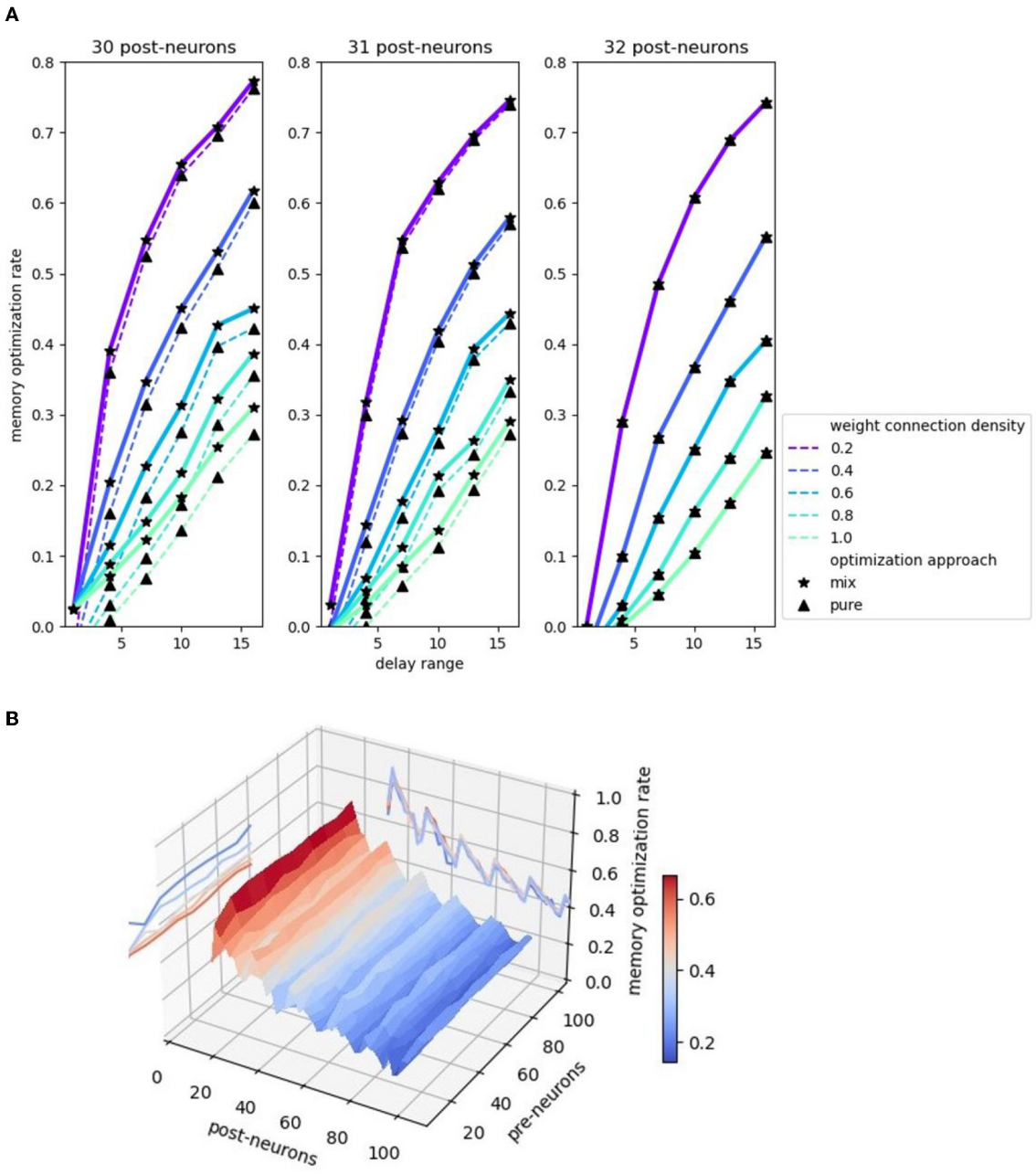


FIGURE 9
Memory optimization rate comparison. **(A)** Memory optimization rate comparison 2D plot. Three plots corresponds to 30, 31, 32 post-neurons. Pre-neuron is set to 50. Delay ranges from 1 to 16. **(B)** Memory optimization rate comparison 3D plot with the projections of the contours.

Subordinate PEs and one Dominant PE. Both serial ARM and original MAC baselines need more than one PE, given the available memory of one PE. We split the weight and input operands of these two baselines into multiple cores in a balanced and compact way for a fair comparison purpose. According to the operand scale, two and four PEs are required for these two baselines, respectively.

Analyzing the performance data reported on the bottom left of Figure 10A, the echelon optimization with mixed processors occupies 74.28% of the memory of the original MAC calculation. It is close to the serial ARM memory cost. As for execution time, it accounts for 24.56% of the serial ARM calculation, i.e., this approach optimizes 75.44% of the runtime. The echelon optimization with MAC even increases this percentage to 89.86% (1−10.14%). The remainder m and row number are calculated to be 4 and 5,500. According to the analysis in Section 3.2.3, the mixed optimization is expected to outperform the pure MAC regarding memory footprint but be inferior concerning execution time. The experiment data from Figure 10 are consistent with this deduction.

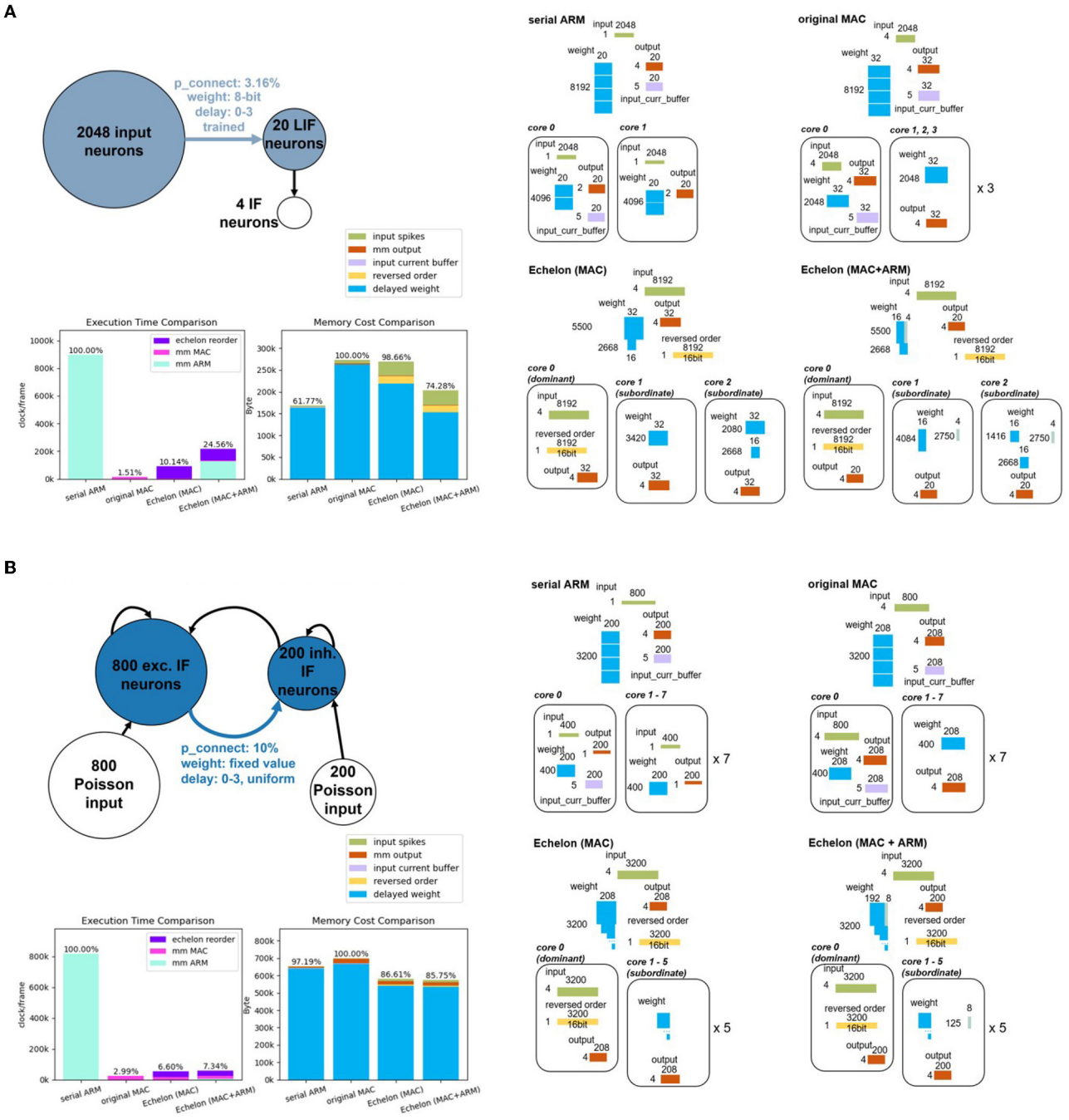


FIGURE 10
(A) Top-left is the structure of the radar gesture recognition model. The experiment part corresponds the synaptic processing of the blue-marked components, including the neural populations and the synaptic projection. The right subfigures depict what experimental memory objects are required and how they are split and deployed into multiple PEs of SpiNNaker 2 hardware according to four different synaptic processing approaches (two baselines and two proposed approaches). The two bar plots on the bottom left compares the execution time and memory cost of these four approaches. The colorful legend in the middle fits the memory cost comparison bar plot and all the subfigures on the right. **(B)** Balanced random cortex-like network has the same subfigure distribution as **(A)**. Because of the limited space, we put the specific splitting result of two echelon algorithms of the balanced random cortex-like network into Supplementary material.

4.2.2. Classic structural modeling in neuroscience: the balanced random cortex-like network

The balanced random cortex-like network, commonly used to benchmark and map to neuromorphic systems (Brüderle et al., 2010; Pfeil et al., 2013; PyNN, 2023), serves as our second benchmark, originated from Brunel's work (Brunel, 2000). Brunel devoted to models of networks with simple neurons to describe the dynamical properties of sparsely connected excitatory and inhibitory integrated-and-fire (IF) neurons. His study reports that

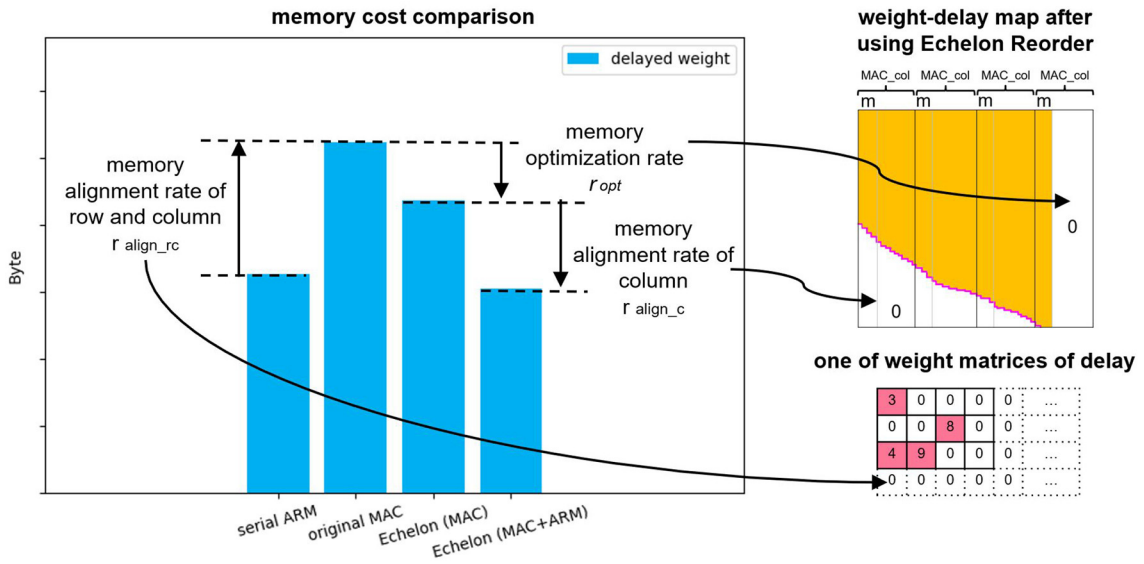


FIGURE 11
Summary of the memory cost differences of delayed weight of the four approaches (two baselines and two proposed approaches) with three "rate." The memory alignment rate of row and column r_{align_rc} equals the memory aligned area in the weight matrices of delay x from Figure 2, and other two "rate" represent the proportions of the pointed white area in the whole area of weight-delay map after using the Echelon Reorder.

the networks present rich states according to external stimulation, the proportion of excitation and inhibition, and synaptic features.

Similar to the cortex in the brain, the balanced random cortex-like networks are composed of an excitatory and inhibitory population of neurons (PyNN, 2023), which are randomly interconnected with each other with fixed probability and recurrently connected within the population. They are also stimulated with excitatory populations with spikes of Poisson distribution, as shown on the top left of Figure 10B. We choose the main part of the model, namely, the excitatory population projecting the inhibitory population marked with dark blue in the Figure to benchmark the proposed approaches in this study. Compared with the original experiment setup in Brunel (2000), the overall number of neurons is scaled to 1,000, but the proportion of excitation and inhibition remains the same (4:1), i.e., 800 excitatory and 200 inhibitory neurons. The parameter delay keeps the original value four and is uniformly distributed in the range of 0 to 3. We create the connection between excitatory and inhibitory populations with a fixed probability of 0.1.

The pure MAC echelon and mixed echelon require one Dominant PE and five Subordinate PEs according to the splitting and deployment strategies mentioned in Section 3.2 and 3.3. Both serial ARM and original MAC run on eight PEs. The right part of Figure 10B presents the specific splitting and deployment consequences.

The sub-graph in the lower left of Figure 10 compares the spatial-temporal performance of four approaches. Two echelon optimization occupies ~86% of the memory of the original MAC calculation, and both outperform serial ARM. In addition, the execution time of two echelon algorithms is smaller than 7.4% and close to the original MAC.

4.2.3. Performance comparison of the two actual models

The two benchmark actual models coming from different domains have distinct structures that are suitable for comparison. The radar gesture model features a larger number of pre-neuron (2,048) than the balanced random cortex-like network (800), as shown in Figure 10. According to the analysis in Section 4.1, the number of pre-neurons mainly influences the execution time performance dominated by ARM processing parts (Echelon Reorder and ARM calculating part of matrix-multiplication of mixed echelon). Thus, the violet and light blue-marked areas of two echelon algorithms on the execution time comparison subfigure of Figure 10A is larger than that of Figure 10B. The balanced random cortex-like network with a smaller number of pre-neuron has a better spatial optimization result by leveraging echelon algorithms.

In addition to the distinction in pre-neuron number, these two benchmark models structure different weight connection densities (3.16%, 10%) and post-neuron numbers (20, 200). The formula 2 indicates the model with smaller weight connection densities, and post-neuron numbers deserve a better memory optimization effect for weight-delay map. The blue bars of two echelon algorithms relative to the original MAC in the memory cost comparison of Figure 10A are indeed significantly smaller than that of Figure 10B. However, the overall advantage of spatial performance improvement of the radar gesture model is blocked by extra memory for storing input spikes (green bars) and reversed order (yellow bars) owing to a larger number of pre-neurons.

In addition, we introduce a definition of "memory alignment rate of column" to account for the reason for a sharper drop of the blue bar from echelon (MAC) to echelon (MAC + ARM). The memory alignment rate of the column presents the impact of memory alignment along the column of the weight-delay map for the size of the whole memory aligned weight-delay map, that is,

the rate of the right side white rectangle area to the whole area in Figure 6A(a) or

$$r_{align\_c} = \frac{16 - m}{n_{post} + 16 - m} \tag{3}$$

where m is the reminder of post-neuron number n_{post} divided by 16. With the increase of the post-neuron number, the memory alignment rate decreases. This means that the extra area brought by column memory alignment is a decreasing share of the total weight-delay map. As a result, the echelon (MAC + ARM) algorithm, functioning as optimizing this extra area as mentioned in Section 3.2.2 and 3.2.3, behaves much better in the radar gesture recognition model than in the balanced random cortex-like network. Similarly, the different ratios (we name the ratio r_{align_rc}) in comparison of serial ARM and original MAC (61.77%, 97.19%) are caused by memory alignment along the row and column of weight matrices of delay x shown in Figure 2. A larger post-neuron number leads to a smaller memory increase of memory alignment from serial ARM to original MAC.

5. Conclusion and discussion

5.1. Summary

This study describes the processing chain for accelerating SNN inference with multi-core MAC arrays, including Echelon Reorder information densification algorithm, Multi-core two-stage splitting and multi-core authorization deployment strategies. These algorithms and strategies alleviate the intrinsic memory issue of excessive usage originating from memory alignment and data sparsity. They also realize the multi-core spatial-temporal load balancing for the large SNN layer. We benchmark with constructed and actual SNN models. The former explores how model feature and algorithm selection affect the spatial-temporal optimization performance, and the latter demonstrates two actual SNN models (the radar gesture recognition SNN model and balanced random cortex-like network) on SpiNNaker 2 hardware. They prove the feasibility of the whole optimization process and achieves performance increase. Based on the theoretical analysis and the experiment result, we found those as follows:

- The proposed algorithms and strategies are applicable to various densities of matrix multiplication operands, and the memory optimization degree increases with the weight operand getting sparse.
- In addition to the weight sparsity, the memory optimization rate is also positively correlated with delay range.
- The number of post-neurons periodically affects the memory optimization rate, and the overall trend is downward.
- The number of pre-neuron is generally independent of the memory optimization performance but has intense correlation with running time.
- The echelon mixed processor algorithm behaves better regarding memory but has less temporal efficiency than the echelon pure MAC solution.

The proposed algorithms not only provide a concrete solution for accelerating SNN on the multi-core MAC arrays of SpiNNaker 2 but also has a referential value for hardware systems embedded with multi-core MAC arrays that intend to solve the SpGEMM issue.

5.2. Related work

Some researchers have introduced the parallel computing concept into SNN inference to tackle the problems caused by CPU-based parallel processing. One of the representative works of parallel processors accelerating SNN computation is GeNN (Yavuz et al., 2016), a code generation framework speeding up the SNN simulation process using the graphics processing unit (GPU). Specifically, it speeds up the synaptic processing by utilizing a serially executed atomic add operation to add weight to the delay ring-buffer after reading the index of post-neuron and weight and delay by each thread of the GPU (Yavuz et al., 2016; Knight and Nowotny, 2018). However, this mixture of thread/vector parallel processing and serial add operation does not take full advantage of the parallelism of the processor.

Another related study regarding parallel processing SNN inference is SpiNeMap (Balaji et al., 2020) and its follow-up (Balaji et al., 2021). SpiNeMap is a design methodology to partition and deploy SNNs to the crossbar-based neuromorphic hardware DYNAP-SE (Moradi et al., 2018). The proposed unrolling techniques decompose a neuron function with many presynaptic connections into a sequence of computation nodes with two presynaptic connections. This approach improves the crossbar utilization but introduces spike latency, i.e., distorts interspike intervals (ISIs) for global synapses. This issue can be relieved by reducing the number of spikes on global synapses as reported in Balaji et al. (2020), probably realized by modifying the model parameters or decreasing the input spike rate, both of which can negatively impact the accuracy of the original SNN.

Our study targets no ISIs, no accuracy loss, and a wholly parallel calculation based on the more efficient matrix parallelism rather than vector-based computing.

5.3. Macro significance

The proposed algorithms act on synaptic processing, which is part of the SNN inference. The impact of algorithms on the optimization of the whole network is a question worth discussing. To review the motivation of our study, it is discussed in Section 1 that the serial ARM baseline suffers from a large time-consuming issue and the naive parallel solution original MAC is unfriendly to limited memory space, it is necessary to explore the portion of synaptic processing of serial ARM in time and original MAC in memory in order to have a macro view of the degree of optimization of our proposed algorithms in the whole SNN inference.

SNN inference consists of synaptic processing and neural update. We estimated the time consumption rate of these two steps for the serial ARM baseline with the following equation referring to ARM Cortex-M4 Technical Reference Manual (ARM, 2023):

$$\frac{t_{synap}}{t_{neural}} = \frac{n_{pre} \times n_{post} \times d \times MLA}{n_{post} \times (MLA + COMP)} \approxeq 0.67 \times n_{pre} \times d. \quad (4)$$

This equation shows that the time consumption rate depends on the number of pre-neuron and delay range. A meaningful model is supposed to have more than one pre-neurons, so this rate is always greater than 1. For the two benchmark actual models in Section 4.2, the synaptic processing time is 5,488.64 and 2,144.0 times larger than neural update.

As for the memory comparison, suppose that the number of pre- and post-neuron is memory aligned, the decay and threshold values of the neural update of all neurons are identical, and only count one output matrix for the multi-core scenario, we have the following equations:

$$\frac{mem_{synap}}{mem_{neural}} = \frac{4 \times n_{pre} + n_{pre} \times n_{post} \times d + 4 \times n_{post} + 4 \times n_{post}}{4 \times n_{post} + 0.125 \times n_{post}}$$
$$\approxeq 1.94 + 0.24 \times n_{pre} \times (\frac{4}{n_{post}} + d). \quad (5)$$

This equation provides the rate of memory cost of synaptic processing to neural update for the original MAC baseline. The rate always greater than 1.94 means synaptic processing dominates memory cost. For the two benchmark actual models, this rate indicates the memory cost of synaptic processing is 2,066.32 and 773.78 times larger than neural update, respectively.

The above search confirms that synaptic processing has the lion's share in SNN inference, both in terms of running time and memory storage. Therefore, the optimization of synaptic processing is crucial and largely determines the optimization performance of the whole SNN inference.

5.4. Limitation and future work

By analyzing formula 2 and comparing two actual models regarding the temporal performance of synaptic processing, we found that the larger number of pre-neuron hinders the running time optimization degree that is primarily governed by ARM calculation parts of the Echelon Reorder and matrix-multiplication of the echelon mixed processor approach. Future optimization for time can be placed on finding a more efficient algorithm for the Echelon Reorder and further compressing the data of the serial operation range in the echelon mixed processor approach.

The spatial performance of our proposed optimization algorithms is limited by the smaller delay range, denser weight connection, and the larger number of post-neuron. How to further optimize the memory optimization rate r_{opt} and r_{align_c} is defined in Section 4.1 and Section 4.2.3 and increasing the area of the white area of Figure 11 are the next things that are worth investigating. This study merely elaborates on the optimization mechanism of the Echelon Reorder that is based on the data transformation of the Operand Stack. In fact, the other two optimization algorithms (Zero Elimination and Proportion Merger) in Table 1 proposed in our previous study (Huang et al., 2023) of one PE accelerating SNN inference also fit multiply PEs. The memory optimization will benefit from their synergy.

Although Section 3.3 provides the solution of authorizing PEs that contain the split echelon matrix and input buffer, an efficient multi-core deployment strategy and routing algorithm are not included in this study. Randomly deploying the six cores from Figure 8 on SpiNNaker 2 may cause a relatively low-efficient communication between the "Dominant PE" and the "Subordinate PE." This issue will be of greater concern when we extend the deployment of one SNN layer to an entire network. At that time, a reasonable multi-layer deployment topology and global routing algorithm can avoid the potential traffic congestion and reduce the communication latency by fully utilizing the bandwidth resources of PE to PE and PE to DRAM. Thus, improving the spatio-temporal efficiency of the entire SNN even multiple SNNs on SpiNNaker 2 will be one of our future research priorities.

The traditional serial processing for SNN inference, as the current mainstream method, is constantly being optimized and iterated, and the performance has been improving. It has a good performance in the condition of very sparse input spike train and weight-delay operand, which is what the approaches proposed in this study is yet to be improved. If we can find the sweet spot of SNN model structure factors including input and weight connection density between the traditional approach and our algorithms, it will help the neuromorphic community to have a deeper understanding of the serial and parallel processing methods and contribute to the mechanism of the hybrid processors jointly processing SNN inference in a more efficient way.

Data availability statement

The data analyzed in this study is subject to the following licenses/restrictions: the data that support the findings of this study are available from the corresponding author upon reasonable request. Requests to access these datasets should be directed at: Jiaxin.Huang@infineon.com.

Author contributions

JH, CL, and CM conceived the study. JH implemented the optimization system with Python (supported by FKr and PG), carried out the experiment of constructed SNN models (supported by DS), and performed the actual SNN models mapping and emulating on SpiNNaker 2, based on the low-level hardware design and communication library from FKe. BV helped JH with the basic structure of SNN middle-ware development. KK and CM supervised the findings of this study. All authors discussed the results and contributed to the final manuscript.

Funding

This study was partially funded by the German Federal Ministry of Education and Research (BMBF) within the KI-ASIC project (16ES0993 and 16ES0996) and the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy—EXC 2050/1—Project ID 390696704—Cluster of Excellence Centre for Tactile Internet with Human-in-the-Loop (CeTI) of Technische Universität Dresden.

Acknowledgments

We thank Infineon Technologies AG for supporting this research.

Conflict of interest

JH, FKr, PG, DS, and KK were employed by Infineon Technologies Dresden.

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

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2023.1223262/full#supplementary-material

References

An, X., and Çatalyürek, U. V. (2021). "Column-segmented sparse matrix-matrix multiplication on multicore CPUS," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)* (Bengaluru: IEEE), 202–211. doi: 10.1109/HiPC53243.2021.00034

ARM (2023). *Cortex-m4 Technical Reference Manual*. Available online at: https://developer.arm.com/documentation/ddi0439/b/CHDDIGAC (accessed July 2, 2023).

Balaji, A., Das, A., Wu, Y., Huynh, K., Dell'Anna, F. G., Indiveri, G., et al. (2020). Mapping spiking neural networks to neuromorphic hardware. *IEEE Trans. Very Large Scale Integr.* 28, 76–86. doi: 10.1109/TVLSI.2019.2951493

Balaji, A., Song, S., Das, A., Krichmar, J., Dutt, N., Shackleford, J., et al. (2021). Enabling resource-aware mapping of spiking neural networks via spatial decomposition. *IEEE Embed. Syst. Lett.* 13, 142–145. doi: 10.1109/LES.2020.3025873

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. doi: 10.1038/s41467-020-17236-y

Brüderle, D., Bill, J., Kaplan, B., Kremkow, J., Meier, K., Müller, E., et al. (2010). "Simulator-like exploration of cortical network architectures with a mixed-signal VLSI system," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)* (Paris: IEEE), 2784–8787. doi: 10.1109/ISCAS.2010.5537005

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

Davis, T. A. (2018). "Graph algorithms via suitesparse: graphBLAS: triangle counting and k-truss," in *2018 IEEE High Performance extreme Computing Conference (HPEC)* (Waltham, MA: IEEE), 1–6. doi: 10.1109/HPEC.2018.8547538

Gao, J., Ji, W., Tan, Z., and Zhao, Y. (2020). A systematic survey of general sparse matrix-matrix multiplication. *arXiv*. [preprint]. doi: 10.48550/arXiv.2002.11273

Gerhards, P., Kreutz, F., Knobloch, K., and Mayr, C. G. (2022). "Radar-based gesture recognition with spiking neural networks," in *2022 7th International Conference on Frontiers of Signal Processing (ICFSP)* (Paris: IEEE), 40–44. doi: 10.1109/ICFSP55781.2022.9924676

Gustavson, F. G. (1978). Two fast algorithms for sparse matrices: multiplication and permuted transposition. *ACM Trans. Math. Softw.* 4, 250–269. doi: 10.1145/355791.355796

Huang, J., Gerhards, P., Kreutz, F., Vogginger, B., Kelber, F., Scholz, D., et al. (2022a). "Spiking neural network based real-time radar gesture recognition live demonstration," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Incheon: IEEE), 500–500. doi: 10.1109/AICAS54282.2022.9869943

Huang, J., Kelber, F., Vogginger, B., Gerhards, P., Kreutz, F., Kelber, F., et al. (2023). "Efficient algorithms for accelerating spiking neural networks on mac array of SpiNNaker 2," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Hangzhou: IEEE). doi: 10.1109/AICAS57966.2023.10168559

Huang, J., Vogginger, B., Gerhards, P., Kreutz, F., Kelber, F., Scholz, D., et al. (2022b). "Real-time radar gesture classification with spiking neural network on SpiNNaker 2 prototype," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Incheon: IEEE), 362–365. doi: 10.1109/AICAS54282.2022.9869987

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

Kreutz, F., Gerhards, P., Vogginger, B., Knobloch, K., and Mayr, C. (2021). "Applied spiking neural networks for radar-based gesture recognition," in *2021 7th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)* (Krakow: IEEE), 1–4. doi: 10.1109/EBCCSP53293.2021.9502357

Mayr, C., Hoeppner, S., and Furber, S. (2019). SpiNNaker 2: a 10 million core processor system for brain simulation and machine learning. *arXiv*. [preprint]. doi: 10.48550/arXiv.1911.02385

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

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

PyNN (2023). *Running PyNN simulations on SpiNNaker*. Available online at: https://spinnakermanchester.github.io/spynnaker/4.0.0/RunningPyNNSimulationsonSpiNNaker-LabManual.pdf (accessed March 25, 2023).

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

Yan, Y., Stewart, T., Choo, X., Vogginger, B., Partzsch, J., Höppner, S., et al. (2021). Comparing loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control. *Neuromorphic Comput. Eng.* 1, 16. doi: 10.1088/2634-4386/abf150

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep.* 6, 18854. doi: 10.1038/srep18854

Zeinolabedin, S. M. A., Schüffny, F. M., George, R., Kelber, F., Bauer, H., Scholze, S., et al. (2022). A 16-channel fully configurable neural soc with 1.52 μw/ch signal acquisition, 2.79 μw/ch real-time spike classifier, and 1.79 tops/w deep neural network accelerator in 22 nm FDSOI. *IEEE Trans. Biomed. Circuits Syst.* 16, 94–107. doi: 10.1109/TBCAS.2022.3142987

Zhang, Z., Wang, H., Han, S., and Dally, W. J. (2020). "Sparch: efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (San Diego, CA: IEEE), 261–274. doi: 10.1109/HPCA47549.2020.00030

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 09 November 2023
DOI 10.3389/fnins.2023.1275944

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Yujie Wu,
Tsinghua University, China
Alberto Patiño-Saucedo,
Spanish National Research Council (CSIC), Spain
Manolis Sifalakis,
Imec, Netherlands
Qi Xu,
Dalian University of Technology, China

*CORRESPONDENCE
Yansong Chua
✉ caiyansong@cnaeit.com

RECEIVED 10 August 2023
ACCEPTED 23 October 2023
PUBLISHED 09 November 2023

CITATION
Sun P, Chua Y, Devos P and Botteldooren D (2023) Learnable axonal delay in spiking neural networks improves spoken word recognition. *Front. Neurosci.* 17:1275944. doi: 10.3389/fnins.2023.1275944

Learnable axonal delay in spiking neural networks improves spoken word recognition

Pengfei Sun[1], Yansong Chua[2]*, Paul Devos[1] and Dick Botteldooren[1]

[1]Department of Information Technology, WAVES Research Group, Ghent University, Ghent, Belgium, [2]Neuromorphic Computing Laboratory, China Nanhu Academy of Electronics and Information Technology, Jiaxing, China

Spiking neural networks (SNNs), which are composed of biologically plausible spiking neurons, and combined with bio-physically realistic auditory periphery models, offer a means to explore and understand human auditory processing-especially in tasks where precise timing is essential. However, because of the inherent temporal complexity in spike sequences, the performance of SNNs has remained less competitive compared to artificial neural networks (ANNs). To tackle this challenge, a fundamental research topic is the configuration of spike-timing and the exploration of more intricate architectures. In this work, we demonstrate a learnable axonal delay combined with local skip-connections yields state-of-the-art performance on challenging benchmarks for spoken word recognition. Additionally, we introduce an auxiliary loss term to further enhance accuracy and stability. Experiments on the neuromorphic speech benchmark datasets, NTIDIDIGITS and SHD, show improvements in performance when incorporating our delay module in comparison to vanilla feedforward SNNs. Specifically, with the integration of our delay module, the performance on NTIDIDIGITS and SHD improves by 14% and 18%, respectively. When paired with local skip-connections and the auxiliary loss, our approach surpasses both recurrent and convolutional neural networks, yet uses 10× fewer parameters for NTIDIDIGITS and 7× fewer for SHD.

KEYWORDS

axonal delay, spiking neural network, speech processing, supervised learning, auditory modeling, neuromorphic computing

1. Introduction

Artificial neural networks (ANNs) have excelled in speech-processing tasks, relying on optimization algorithms, deep architectures, and powerful feature extraction methods like MFCC. Nevertheless, these typical feature extraction methods do not fully replicate the biologically realistic model of cochlear processing (Wu et al., 2018a,b). Additionally, both ANNs and rate-based Spiking Neural Networks (SNNs) struggle with spiking inputs from biologically inspired cochlear models due to their sparse distribution and high temporal complexity (Wu et al., 2021). The high energy consumption of ANNs further limits their deployment in mobile and wearable devices, hindering the development of sound classification systems (Davies et al., 2018; Wu et al., 2018b). Thus, there is a growing demand for bio-inspired SNN architectures capable of handling the outputs of bio-physically realistic cochlear models.

Despite considerable progress in translating insights from non-spiking ANNs to SNNs (Wu et al., 2021; Xu et al., 2023a,b) and the emergence of enhanced architectures (Xu et al., 2018, 2021, 2022) along with sparse training methods (Shen et al., 2023), the primary application has applied to static datasets or non-stream datasets. While earlier research (Mostafa, 2017; Hong et al., 2019; Zhang et al., 2021) has shown encouraging results on such datasets using temporal encoding algorithms, their potential for large-scale time-series datasets remains a question. Contrastingly, noteworthy advancements has been made by algorithms that directly handle event-driven audio tasks with a temporal dimension (Wu et al., 2019, 2020; Zhang et al., 2019; Blouw and Eliasmith, 2020; Yılmaz et al., 2020). A notable method is the refinement of spike timing precision in models and the exploration of intricate architectures that meld both ANN insights and biological understanding. SNNs, which incorporate adjustable membrane and synaptic time constants (Fang et al., 2021; Perez-Nieves et al., 2021), as well as advanced and optimized firing thresholds (Yin et al., 2021; Yu et al., 2022), have shown substantial promise, especially in integrating precise spike timing to achieve top-tier classification accuracy. Although past methods have placed significant emphasis on the importance of spike-timing, believing that information is intricately embedded within the spatio-temporal structure of spike patterns (Wu et al., 2018c), there has been a conspicuous gap in research concerning the specific effects of event transmission, notably axonal delay (Taherkhani et al., 2015). Neurophysiological studies (Carr and Konishi, 1988; Stoelzel et al., 2017) highlight axonal delay's potential role in triggering varied neuronal responses. It is worth noting that axonal delay is a learnable parameter within the brain, extending beyond the realm of synaptic weights (Seidl, 2014; Talidou et al., 2022). Neuromorphic chips such as SpiNNaker (Furber et al., 2014), IBM TrueNorth (Akopyan et al., 2015), and Intel Loihi (Davies et al., 2018) facilitate the programming of the delay module.

These developments have spurred the exploration of jointly training synaptic weights and axonal delay in deep SNNs. While earlier research mainly centered on fixed delays with trainable weights (Bohte et al., 2002) and the concurrent training of synaptic weights and delays in shallow SNNs featuring a single layer (Taherkhani et al., 2015; Wang et al., 2019; Zhang et al., 2020), there has recently been a degree of investigation into the joint training of the synaptic weights and axonal delays in deep SNNs (Shrestha and Orchard, 2018; Shrestha et al., 2022; Sun et al., 2022, 2023a; Hammouamri et al., 2023; Patiño-Saucedo et al., 2023). Our prior effort (Sun et al., 2022) stands as one of the initial successful attempts in applying this method to deep SNNs, achieving promising results in tasks characterized by high temporal complexity.

In this current work, we focus on spiking spoken word recognition tasks, namely NTIDIDIGITS (Anumula et al., 2018) and SHD (Cramer et al., 2020). These tasks are temporally complex (Iyer et al., 2021) and are encoded as spikes through an audio-to-spiking conversion procedure inspired by neurophysiology. In pursuit of enhancing these tasks, we introduce a learnable axonal delay mechanism to govern the transmission process and achieve precise synchronization of spike timing. Alongside the axonal delay module, we delved into various intricate structures, showcasing their synergy with the delay module. Specifically, we propose a novel local skip-connection mechanism designed to mitigate information loss during the reset process, an endeavor that relies heavily on the precise availability of spike timing information. Additionally, we integrate an auxiliary loss to curb unwarranted neuron membrane potentials upon firing. Our results underscore the seamless integration of these intricate components with the delay modules, resulting in substantial performance enhancements. Our methods achieve state-of-the-art performance while requiring fewer parameters, as demonstrated by our experimental studies.

The rest of the paper is organized as follows. We provide a detailed description of the proposed methods in Section 2. In Section 3, we demonstrate the effectiveness of our algorithms on two event-based audio data-sets and compare them with other SNNs and ANNs. We conclude and discuss future work in Section 4.

2. Materials and methods

In this section, we begin by introducing the spiking neuron model utilized in this work. After that, we present the Variable Axonal Delay (VAD) and Local Skip-Connection methods. The introduction of the Variable Axonal Delay is loosely inspired by neurophysiology, as we argue that the variation of delays observed in the biological system could be advantageous for aligning temporal information on a millisecond time scale. As a result, transient sensory inputs can be condensed into specific spike bursts corresponding to their transience. Next, we introduce the concept of a local skip-connection architecture, which holds the potential to mitigate information loss during the reset mechanism, thereby enhancing the dynamic behavior of the neuron model. Finally, we demonstrate that the suppressed loss further enhances performance, improving the network's discriminative capabilities for target differentiation.

2.1. Spiking neuron model

An SNN employs a spiking neuron as the basic computational unit with input and output in the form of spikes, maintaining an internal membrane potential over time. In this paper, we adopt the Spike Response Model (SRM) which phenomenologically describes the dynamic response of biological neurons.

Consider an input spike, $s_j^{l-1}(t) = \delta(t - t_j^{(l-1)})$. Here $t_j^{(l-1)}$ denotes a firing time of pre-synaptic neuron j in layer $l-1$ and δ the spike function. In the SRM model, the incoming spike $s_j^{l-1}(t)$ is converted into spike response signals by convolving with the spike response kernel $\epsilon(t)$ and is then scaled by the synaptic weight to generate the Post Synaptic Potential (PSP). Likewise, the refractory period can be represented as $(\nu * s_j^l)(t)$ which describes the characteristic recovery time needed before the neuron regains its capacity to fire again after having fired at time t. The neuron's membrane potential, is the sum of all PSPs and refractory response

$$u_i^l(t) = \sum_j W_{ij}^{l-1}(\epsilon * s_j^{l-1})(t) + (\nu * s_i^l)(t) \tag{1}$$

where $u_i^l(t)$ is the membrane potential of neuron i and W_{ij}^{l-1} is the synaptic weight from neuron j to neuron i.

A firing output is generated wherever $u_i(t)$ crosses the predefined firing threshold θ_u. This generation process can be formulated by a Heaviside step function Θ as follows

$$s_i^l(t) = \Theta(u_i^l(t) - \theta_u). \tag{2}$$

2.2. Variable axonal delay (VAD) module

As shown in Figure 1, a VAD is added to the output of each spiking neuron in layer l. Let N be the number of neurons at layer l, thus, the set of spike trains $s^l(t)$ can be represented as follows

$$s^l(t) = \{s_1^l(t), ..., s_N^l(t)\} \tag{3}$$

The forward pass of the delay module can be described as

$$s_d^l(t) = \delta(t - \hat{d}^l) * s^l(t) \tag{4}$$

Where $\hat{d}^l$ is the set of learnable delays $\{\hat{d}_1, \hat{d}_2, .., \hat{d}_N\}$ in layer l, and $s_d^l(t)$ is the spike trains output by the delay module. From the system point of view, limiting the axonal delay of each neuron to a reasonable range can speed up the training convergence. Thus, we clip the delay to the specified range during training and round down after each backpropagation.

$$\hat{d}^l = Min(Max(0, round(\hat{d})), \theta_d) \tag{5}$$

Here, the θ_d refers to the upper bound of the time delay of the spiking neuron.

2.3. Local skip-connection as compensation for loss of information due to reset

The structure of the local skip-connection within a given layer is depicted in Figure 2. In mapping from input spikes to output spikes, The SRM utilizes a refractory kernel to characterize the refractory mechanism, represented by the equation $\nu(t) = -\alpha_r\theta_u \frac{t}{\tau_r} \exp(1 - \frac{t}{\tau_r})\Theta(t)$. One challenge that persists is identifying the ideal refractory scale α_r for specific tasks. If the refractory scale is too small, its effect is diminished, while an overly large refractory scale risks information loss at certain time junctures. To address this, our study introduces the concept of a local skip-connection. This design compensates for information lost during the reset mechanism in a dynamic fashion. Our results show that this connection can operate effectively using the same refractory scale, offering a solution to the intricate task of selecting an optimal refractory scale for various tasks. The output membrane potential of the local skip-connection can be formulated as

$$\hat{u}_i^l(t) = \sum_j V_{ij}^l(\epsilon * s_{d,j}^l)(t) + (\nu * \hat{s}_i^l)(t) \tag{6}$$

V_{ij}^l is the locally connected synaptic weight from neuron j to neuron i at the same layer. Unlike a skip connection, the local skip-connection adds an extra layer of processing to the output spikes generated in layer l. It then directs these locally processed output spikes, denoted as $\hat{s}^l$ with the same index as the original output spikes s_d^l, to follow the same axon line within layer l. As a result, both the local spike trains $\hat{s}^l$ and the original output spikes s_d^l utilize the same weights W_{ij}^l and are channeled to the succeeding layer. This can be equivalently expressed as $s^l = s_d^l + \hat{s}^l$.

2.4. Loss layer

The loss of an SNN compares the output spikes with the ground truth. However, in classification tasks, decisions are typically made based on the spike due to the absence of precise timing. Considering the spike rate over the time interval T, the loss function L can be formulated as follows:

$$L = \frac{1}{2}(\int_0^T \tilde{s}(\tau)\, d\tau - \int_0^T s^{n_l}(\tau)\, d\tau)^2 \tag{7}$$

Here, L measures the disparity between the target spike train $\tilde{s}(t)$ and output spike train $s^{nl}(t)$ at the last layer n_l across the simulation time T. Given the lack of precise spike timing in our tasks, we measure the output spikes through the integration of $s^{n_l}(t)$ over T. For different task scenarios, the target firing rate is set as $\int_0^T \tilde{s}(\tau)d\tau$.

To further exploit temporal information in classification, an auxiliary loss termed the suppressed loss L_{Mem} is introduced:

$$L_{Mem}(t) = \frac{1}{2} \cdot (s^{n_l}(t) \cdot Mask \cdot (u^{n_l}(t - \Delta t) - u_\theta))^2 \tag{8}$$

This loss function is designed to reduce the firing probability of incorrect neurons right when they activate. Compared to previous lateral inhibition methods using learnable or fixed kernels, this loss function achieves a winner-takes-all effect by acting as a regularizer. Importantly, this loss is only applied to false neurons. Here, the spike train s^{n_l} and membrane potential u^{n_l} are functions of time. Moreover, $u^{n_l}(t - \Delta t)$ refers to the membrane potential right before a spike occurs. When a neuron is activated, indicated by $s^{n_l}(t) = 1$, its potential is referred to as $u^{n_l}(t - \Delta t)$. This value is then subtracted from a predetermined membrane potential u_θ, controlled by the suppressing factor λ_u and defined as $u_\theta = \lambda_u\theta_u$. Lastly, to ensure that the suppressed membrane potential loss is limited only to undesired (or false) neurons, a mask $Mask \in \mathbb{R}^C$ is employed, where C is the number of target neurons:

$$Mask = \begin{cases} 0 & \text{True Class} \\ 1 & \text{False Classes} \end{cases} \tag{9}$$

2.5. Backpropagation

The surrogate gradient algorithm in combination with the Backpropagation-Through-Time (BPTT) (Werbos, 1990) in SNN has shown excellent performance on temporal pattern recognition tasks.

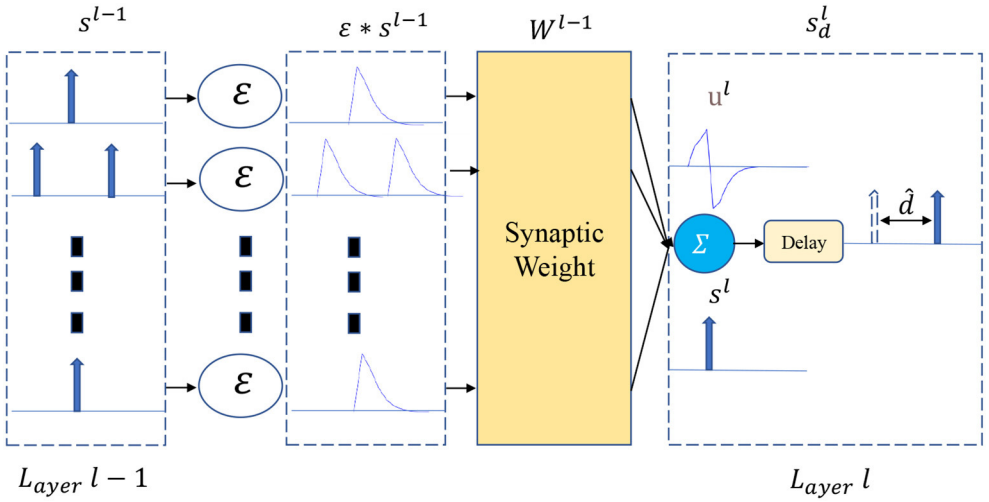


FIGURE 1
Illustration of the flow chart of the axonal delay. The output spike is delayed by $\hat{d}$.

In this work, we discretise the temporal dimension with the sampling time T_s such that $t = nT_s$ where n denotes the time step of the simulation. We also define $(N_s+1)T_s$ as the total observation time. For the Heaviside step function, we adapt the SLayer function (Shrestha and Orchard, 2018) to formulate the proxy gradient, which is defined as

$$\hat{f_s'} = \tau_{scale} \exp(-|u(t) - \vartheta|/\tau_\vartheta) \tag{10}$$

Here, τ_{scale} and τ_ϑ are two parameters that control the sharpness of the surrogate gradient. Similarly, the gradient of the axonal delay is given by

$$\nabla_{\hat{d}^l} E = T_s \sum_{n=0}^{N_s} \frac{\partial L[n]}{\partial \hat{d}^l} \tag{11}$$

Using the chain rule and noting that the loss at time-step n depends on all previous timesteps, we get

$$\begin{aligned} \nabla_{\hat{d}^l} E &= T_s \sum_{n=0}^{N_s} \sum_{m=0}^{n} \frac{\partial s_d^l[m]}{\partial d^l} \frac{\partial L[n]}{\partial s_d^l[m]} \\ &= T_s \sum_{n=0}^{N_s} \sum_{m=0}^{n} \frac{s_d^l[m] - s_d^l[m-1]}{T_s} \frac{\partial L[n]}{\partial s_d^l[m]} \end{aligned} \tag{12}$$

Here, the finite difference approximation $\frac{s_d^l[m]-s_d^l[m-1]}{T_s}$ is used to numerically estimate the gradient term $\frac{\partial s_d^l[m]}{\partial d^l}$. As part of the backpropagation process, the gradient of delay is propagated backward, and then the delay value is subsequently updated. Similarly, we also formulate the gradient term of the suppressed loss.

$$\frac{\partial \mathcal{L}_{Mem}}{\partial u^{n_l}} = s^{n_l} \cdot Mask \cdot (u^{nl} - u_\theta) \tag{13}$$

As shown in Figure 2, beginning from the input layer, the spike trains compute forward and the error propagates backward.

3. Experiments and results

In this section, we first evaluate the effectiveness of the proposed delay module and novel architecture on two event-based audio datasets: NTIDIDIGITS and SHD. Additionally, we assess the impact of the novel auxiliary loss in boosting performance. Finally, we compare our results with several state-of-the-art networks, including feedforward SNNs, recurrently connected SNNs (RSNNs), and Recurrent Neural Networks (RNNs).

3.1. Implementation details

The experiments are conducted using PyTorch as a framework, and all reported results are obtained on 1 NVIDIA Titan XP GPU. Each network and proposed architecture is trained with the Adam optimizer (Kingma and Ba, 2014) and has the same training cycle. The simulation time step T_s is 1 ms, and the firing threshold θ_u is set at 10 mV. The chosen response kernel is $\epsilon(t) = \frac{t}{\tau_s}\exp(1-\frac{t}{\tau_s})\Theta(t)$, and the refractory kernel is $\nu(t) = -\alpha_r\theta_u \frac{t}{\tau_r}\exp(1-\frac{t}{\tau_r})\Theta(t)$. The time constant of the response kernel τ_s and refractory kernel τ_r is set to 5 for NTIDIDIGITS and 1 for SHD datasets. The suppressed factor λ_u is set to 0.995 to suppress the membrane potential of the firing undesired neurons below the threshold. For the proxy gradient, we adopt the Slayer (Shrestha and Orchard, 2018). Table 1 lists other hyperparameters used.

The following notation is used to describe the network architecture: "FC" stands for a fully-connected layer, "VAD" means Variable Axonal Delay module, "Local" denotes the local skip-connection architecture, and L_{Mem} implies the use of the suppressed loss in addition to the spike rate loss. For example, `Input-128FC-VAD-Local-128FC-VAD-Local-Output + L_{Mem}`indicates that there are two dense layers with 128 neurons, each implementing the VAD and Local module. The loss is measured by the spike rate and suppressed membrane

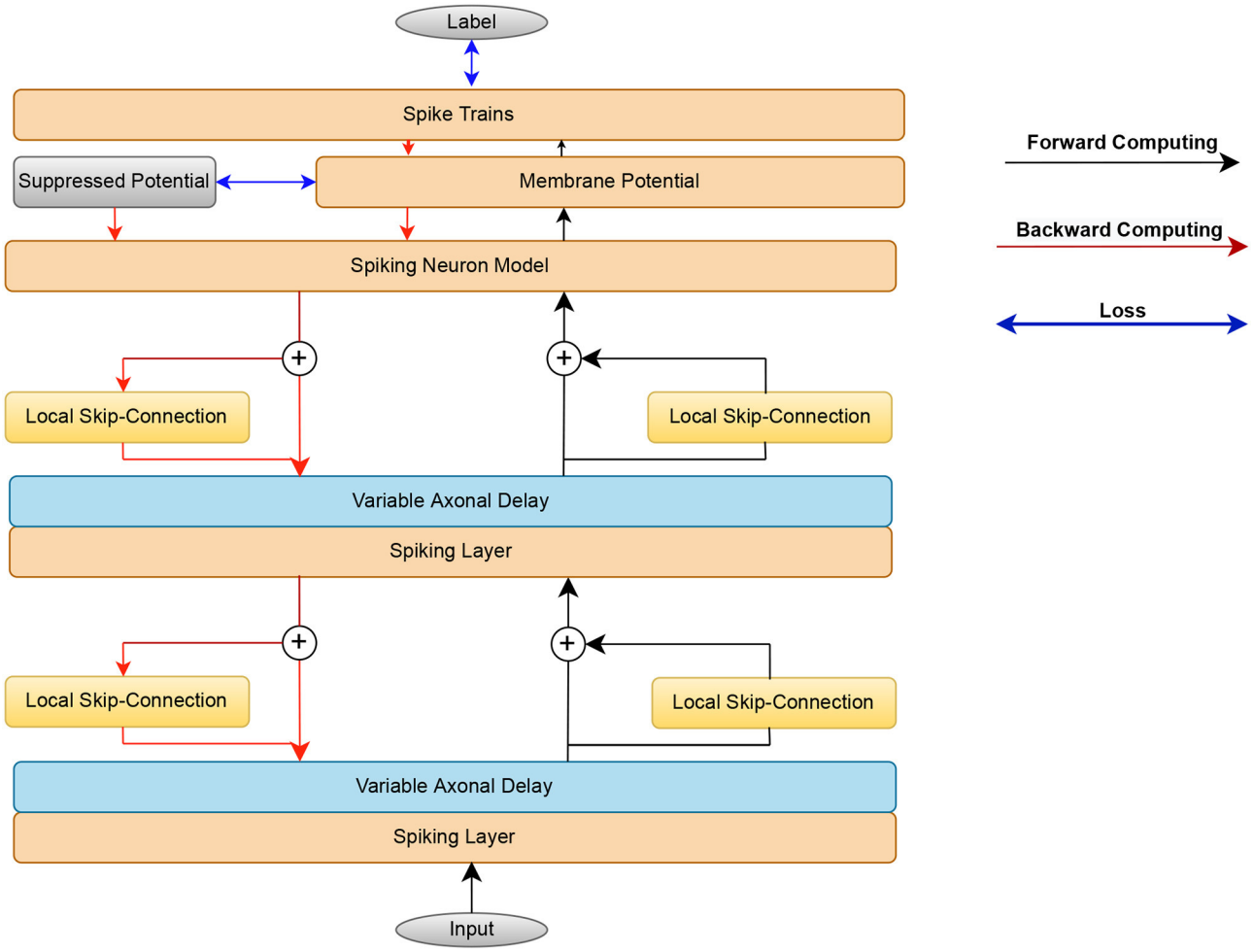


FIGURE 2
Flow chart illustrating the proposed methods. In the forward pass, the input spikes are mapped by the SRM, axonal delay module, and local skip-connection to the output spikes. The error consists of the spike rate loss from the last layer and the suppressed loss from the false neuron's membrane potential. The spiking layer consists of the Spiking neuron model and membrane potential layer. The error gradients are passed backward through time to update the weight and axonal delay parameters.

TABLE 1 Detailed hyper-parameter settings.

Hyper-parameter	N-TDIDIGITS18	SHD
Batch size	128	128
Learning rate	0.1	0.1
Time constant τ_s	5	1
Time constant τ_r	5	1
Membrane threshold θ_u	10	10
Refractory scale α_r	2	2
Delay threshold θ_d	128	64
Suppressed factor λ_u	0.995	0.995

TABLE 2 Name and corresponding network structure. L2 denotes the l2 regularizer for delay values.

Name	Network structure
D128-SNN	Input-128FC-VAD-128FC-VAD-Output
DL128-SNN	Input-128FC-VAD-Local-128FC-VAD-Local-Output
DL128-SNN-Dloss	Input-128FC-VAD-Local-128FC-VAD-Local-Output + L_{Mem}
DL256-SNN-Dloss	Input-128FC-VAD-Local-256FC-VAD-Local-Output + L_{Mem}
DL128-SNN-Dloss-L2	Input-128FC-L2(VAD)-Local-128FC-L2(VAD)-Local-Output + L_{Mem}

potential. Table 2 summarizes the abbreviations for different architectures and methods.

The number of spikes generated from the last layer is compared to the desired spikes in dedicated output nodes, serving as the primary loss measurement. In order to implement the suppressed membrane potential loss function, the model is pre-trained for 20 epochs to generate the target spike trains used for L_{Mem} definition. For a fair comparison, all the experiments are run for 5 independent trials, and the average performance and standard deviation are reported.

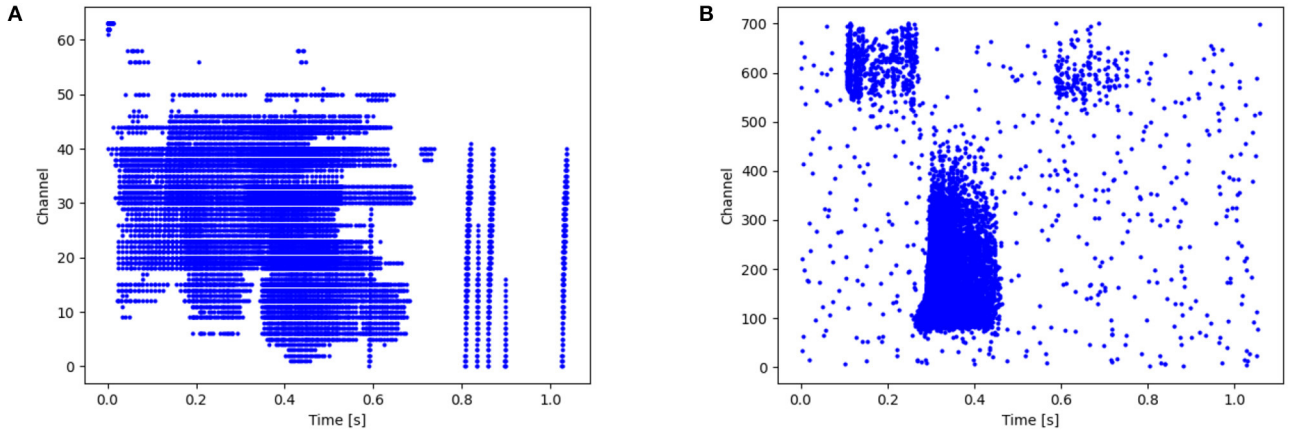


FIGURE 3
(A) Raster plot of Spiketrains of input from a single sample (label 0) of the NTIDIDIGITS dataset. The y-axis represents the channels of the cochlear model while the x-axis indicates the time. **(B)** An Illustration of one raw example (word "six") from the SHD dataset.

TABLE 3 Comparison of classification and parameter count of proposed methods on the NTIDIDIGITS and SHD Test sets.

Dataset	Method	Params	Accuracy (%)
N-TDIDIGITS18	GRU-RNN (Anumula et al., 2018)†	0.11M	90.90
	Phased-LSTM (Anumula et al., 2018)†	0.61M	91.25
	ST-RSBP (Zhang and Li, 2019)	0.35M	93.90
	SrSc-SNNs-IP (Zhang and Li, 2021)	0.61M	95.07
	DL128-SNN-Dloss	**0.06M**	**95.22**
SHD	Feed-forward SNN (Cramer et al., 2020)	0.09M	48.1
	RSNN (Cramer et al., 2020)	1.79M	83.2
	RSNN with adaption (Yin et al., 2020)	0.14M	84.40
	Heterogeneous RSNN (Perez-Nieves et al., 2021)	0.11M	82.78
	RSNN with attention (Yao et al., 2021)	0.14M	91.08
	DMUC (Sun et al., 2023b)†	0.24 M	91.48%
	CNN (Cramer et al., 2020)†	1.01M	92.40
	RadLIF (Bittar and Garner, 2022)	3.9M	94.62
	DCLS (Hammouamri et al., 2023)*	0.21M	**95.07**
	SNN with delays (Patiño-Saucedo et al., 2023)	0.1M	90.04
	DL128-SNN-Dloss	0.14M	92.56
	DL256-SNN-Dloss	0.21M	93.55

†Non-SNN implementation.
*Channel reduction. Bold values are the best results.

TABLE 4 Ablation studies for different architecture and learning methods.

Dataset	Network	Params	Accuracy (%)
NTIDIDIGITS	Input-128FC-128FC-11	26,251	78.52
	Input-128FC-Local-128FC-Local-11	59,275	79.36
	D128-SNN	26,507	92.99
	DL128-SNN	59,531	94.70 ± 0.35
	DL128-SNN-Dloss	59,531	95.22 ± 0.08
	DL128-SNN-Dloss-L2	59,531	94.85 ± 0.08
SHD	Input-128FC-128FC-20	108,820	67.05
	Input-128FC-Local-128FC-Local-20	141,844	65.55
	D128-SNN	109,076	85.73
	DL128-SNN	142,100	91.52 ± 0.84
	DL128-SNN-Dloss	142,100	92.56 ± 0.56
	DL128-SNN-Dloss-L2	142,100	92.44 ± 0.09

3.2. Datasets

Tests are performed on the speech classification datasets NTIDIDIGITS and Spiking Heidelberg Digits (SHD). Both datasets represent events in the form of spikes, containing rich temporal information that is naturally suited to be directly processed by an SNN. These datasets are considered benchmarks, allowing us to focus on the architecture and learning algorithm of the SNN without considering the spike generation method.

3.2.1. NTIDIDIGITS

The NTIDIDIGITS (Anumula et al., 2018) dataset was created by playing the TDIDIGITS (Leonard and Doddington, 1993) to the 64 response channel silicon cochlea. The dataset includes single digits and connected digit sequences, all of which contain the 11 spoken digits ("oh," and the digits "0" to "9"). For the n-way classification problem (single digits), there are a total of 55 male and 56 female speakers with 2,463 training samples, and 56 male and 53 female speakers in the testing set with a total of 2,486 samples. As shown in Figure 3A, the time resolution is in *ms* level and the channel ranges from 0 to 63.

3.2.2. SHD

The SHD is the spiking version of the Heidelberg Digits (HD) audio dataset that is converted by a biologically inspired cochlea model (Cramer et al., 2020). There are 8,156 and 2,264 spoken samples for training and testing, respectively. It contains 10-digit utterances from "0" to "9" in English and German, with a total of 20 classes presented by 12 speakers. Figure 3B shows an example of this audio spike stream. Each sample duration ranges from 0.24 to 1.17 s. Here, the time is resampled to speed up the training (Yin et al., 2020). Each channel has at most 1 spike every 4 *ms* and shorter samples are padded with zeros.

3.3. Overall results

This section demonstrates the benefits of the proposed innovations and assesses the effects of the VAD, Local skip-connection, and Suppressed loss individually to validate their impact on boosting performance. The basic SNN consists of 2 hidden layers, followed by the VAD module, Local skip-connection in each layer, and the suppressed loss module in the readout layer's membrane potential (Figure 2).

1) NTIDIDIGITS. As shown in Table 3, non-spiking approaches such as GRU-RNN and Phased-LSTM (Anumula et al., 2018) achieve 90.90 and 91.25% accuracy, respectively. However, these RNNs rely on the event synthesis algorithm and cannot fully exploit sparse event-based information. Zhang and Li (2019) directly train the spike-train level features with recurrent layers through the ST-RSBP method, and Zhang and Li (2021) further propose the SrSc-SNNs architectures that consist of three self-recurrent layers with skip-connections, training this SNN using backpropagation-based intrinsic plasticity, achieving state-of-the-art (SOTA) performance. We show that with the proposed VAD module, local skip-connection, and suppressed loss, our method achieves 95.30% accuracy with a mean of 95.22% and a standard deviation of 0.08%, making it the best result in this classification task. Furthermore, our model uses the least parameters and is 10× smaller compared to the second-best result.

2) SHD. For this dataset, we compare our methods with recent advancements. In Cramer et al. (2020), the single feed-forward SNN and Recurrent SNN are both trained using BPTT. Their results show that the recurrent architecture outperforms the homogeneous feed-forward architecture in this challenging work, underscoring the potential advantages of intricate SNN designs. Several studies have ventured into specialized SNN architectures. For instance, some explore the effectiveness of the heterogeneous recurrent SNNs (Perez-Nieves et al., 2021), while others delved into attention-based SNNs (Yao et al., 2021). As detailed in Table 3, our proposed method produces a competitive performance of 92.56% in a two-layer fully connected network of 128 neurons each. Notably, this performance is competitive compared to these results that employ the same data processing methods and network architecture. Patiño-Saucedo et al. (2023) introduce axonal delays in tandem with learnable time constants, enabling a reduction in model size to a mere 0.1 M while preserving competitive performance.

Additionally, RadLIF (Bittar and Garner, 2022) combines an adaptive linear LIF neuron with the SG strategy, achieving a performance of 94.62%. This achievement is realized through the utilization of three recurrent spiking layers, each containing 1024 neurons. On the other hand, DCLS, introduced in Hammouamri et al.'s research (Hammouamri et al., 2023), capitalizes on several key innovations. It incorporates learnable position adjustments within the kernel, employs advanced data augmentation techniques (like the 5-channel binning), and incorporates batch normalization methods. As a result, DCLS achieves an accuracy of 95.07% using two feedforward spiking layers, each comprising 256 neurons. Given the sizeable 700-input channel, we mitigated extensive parameter expansion by augmenting the neural network's second layer from 128 to 256 neurons. This strategic adjustment significantly improved performance, yielding a 93.55% accuracy rate.

3.4. Ablation study

We delve into the contributions of VAD, Local skip-connection, and Suppressed loss via a comprehensive ablation study (refer to Table 4). Evaluating each method individually on two fully-connected feed-forward SNNs provides the following insights:

- **VAD:** When incorporated, there is a marked enhancement in the accuracy across datasets. Specifically, with the delay module embedded (in the D128-SNN setup), we obtain gains of 14.47% and 18.68% for NTIDIDIGITS and SHD, respectively. Importantly, despite these advancements, the parameters remain nearly unchanged. This is attributed to our adoption of channel-wise delays, implying that the increase in parameters corresponds only to the number of channels in each layer. As an illustration, with the SHD dataset, the integration of VAD results in an increment of N parameters in each layer, with N being set to 128 in our experimental setup.
- **Local skip-connection:** Its standalone application (reflected in the Input-128FC-Local-128FC-Local-11 design) does not bolster accuracy notably. For the SHD dataset, the outcome is even slightly detrimental. However, this method increases the number of trainable parameters. This can be likened to the addition of an extra feedforward layer, resulting in a parameter increment of $N \times N$ for each layer.

Combining VAD and Local skip-connection in the DL128-SNN design yields significant benefits. We clinch state-of-the-art

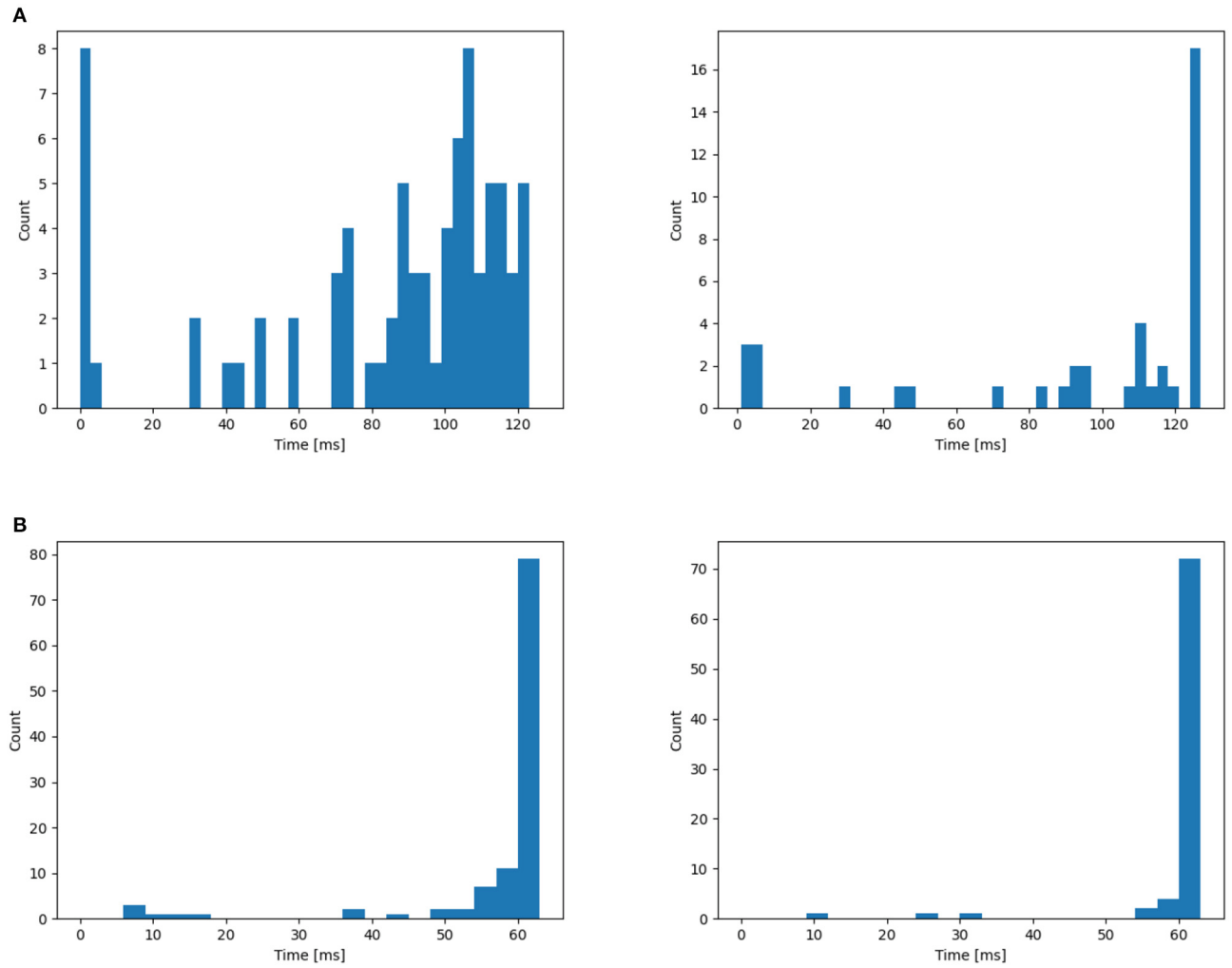


FIGURE 4
Distribution of time delay on **(A)** NTIDIDIGITS, **(B)** SHD. The initial distribution are all 0. From the left to right: First layer, second layer.

accuracy levels for both datasets. This highlights that the enhanced flexibility provided by VAD truly shines when paired with a richer parameter landscape, as provided by the Local skip-connection. Lastly, supplementing the above with the suppressed loss, Dloss, results in stellar performance: 95.22% for NTIDIDIGITS and 92.56% for SHD.

3.5. Axonal delay improves the characterization learning ability

In this section, we begin by offering a visual representation of the axonal delay distribution (refer to Figure 4) for both datasets. Subsequently, we employ an L2 regularizer on the delay to curtail the magnitude of delay values, effectively reducing the number of delayed time steps.

Utilizing the NTIDIDIGITS dataset as an illustrative example, Figure 4A reveals a delay distribution in the first layer that consistently encompasses both long and short delay neurons. This may imply that certain neurons focus on the initial portion of the input, whereas others concentrate on the latter segment of the input features. To understand the dynamics of the VAD, we inspect the cumulative spike count at the input of the network and compare it to the cumulative spike count at the true decision neuron for four different models, as depicted in Figure 5. For illustrative purposes, we select four different English-speaking digit utterances: “1”, “6”, “7”, and “10”. The figures clearly show that the model without delay gradually increases its prediction as the input spikes come in and starts to do so as soon as input spikes start arriving. Conversely, for the other three models equipped with delay modules, the decision to increase spike count in the true neuron is delayed but then increases more quickly and reaches a higher level. This phenomenon arises from the different neurons introducing varying delays to the spikes, thereby providing the terminal neuron with multi-scale information. This may be interpreted as the VAD-enabled network aggregating all information in the spoken word before triggering a decision using all that information simultaneously. Moreover, we can observe that

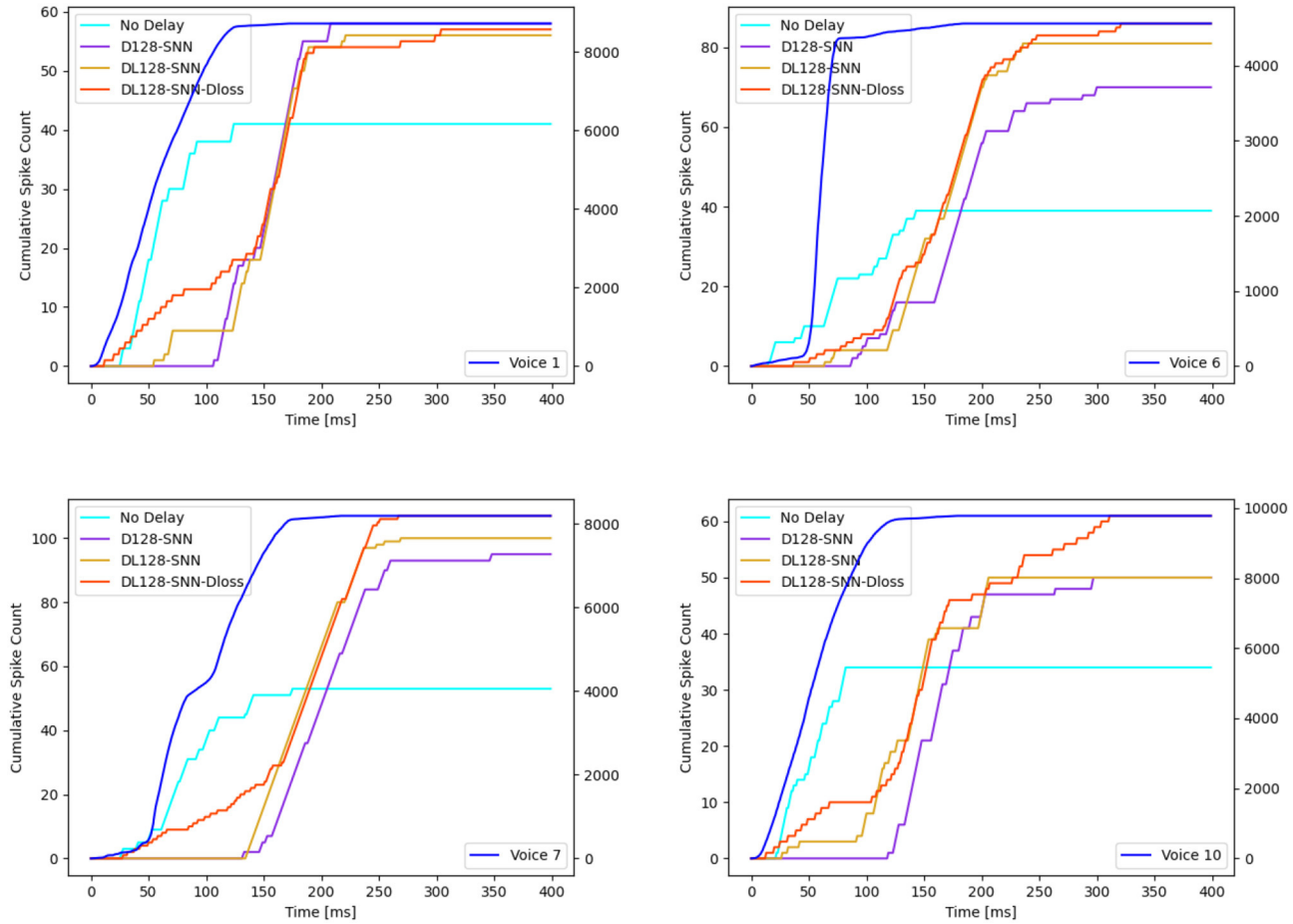


FIGURE 5
Illustration of 4 distinct English examples ("1", "6", "7", and "10"). The cumulative spike count of the input is plotted on the right y-axis (represented by the blue line), while the true neurons' cumulative spike count is on the left y-axis. Four models are showcased:: No delay, D128-SNN, DL128-SNN, and DL128-SNN-Dloss.

the models with delay typically have a total of 60 time step latency, which can be measured after the input is over. This is not only related to the delay itself but also to the choice of loss evaluation. As Shrestha et al. (2022) discussed, the spike-based negative log-likelihood loss results in early classification, even 1400 time steps faster than spike-rate based loss evaluation for NTIDIDIGITS datasets. However, the DL-128-SNN-Dloss generates the highest number of spikes for the true neuron compared to the other models, demonstrating its superior ability to learn characterizations.

Subsequently, the L2 loss is employed to confine the range of delay values to provide a more uniform distribution. This leads to a reduction in delay values for some neurons (see Figure 6), aiming to reduce the total latency and investigate whether shorter delays contribute to a better classification system. This is achieved by applying the L2 regularizer to $\sum_{i=1}^{N} \hat{d}_i$. Nevertheless, as demonstrated in Table 4, the inclusion of the additional L2 loss results in a performance decline. This could indicate that the learned distributions achieved through these architectures may already be optimal within the current delay threshold, denoted as θ_d.

3.6. Local skip-connection as compensation for loss of information in reset mechanism

The positive impact of local skip-connections on the reset mechanism becomes evident when modulating the refractory scale, symbolized as α_r. We conduct a comparative analysis of performance between two distinct configurations: one labeled as VAD, which encompasses solely the delay model, and the other designated as VAD+Local, which additionally incorporates local skip-connections. As shown in Figure 7, the Local skip-connection maintains high performance across a wider range of refractory scales α_r, while the performance with only the VAD module starts to decline with high values. This observation aligns with our earlier conjecture that larger values of α_r may induce information loss, as the neuron's potential struggles to recover efficiently. In contrast, the presence of local connections mitigates this loss by dynamically triggering spiking events among local neurons. Thus, our Local skip-connection diminishes sensitivity to parameter selection, potentially providing more flexibility to train SNNs for

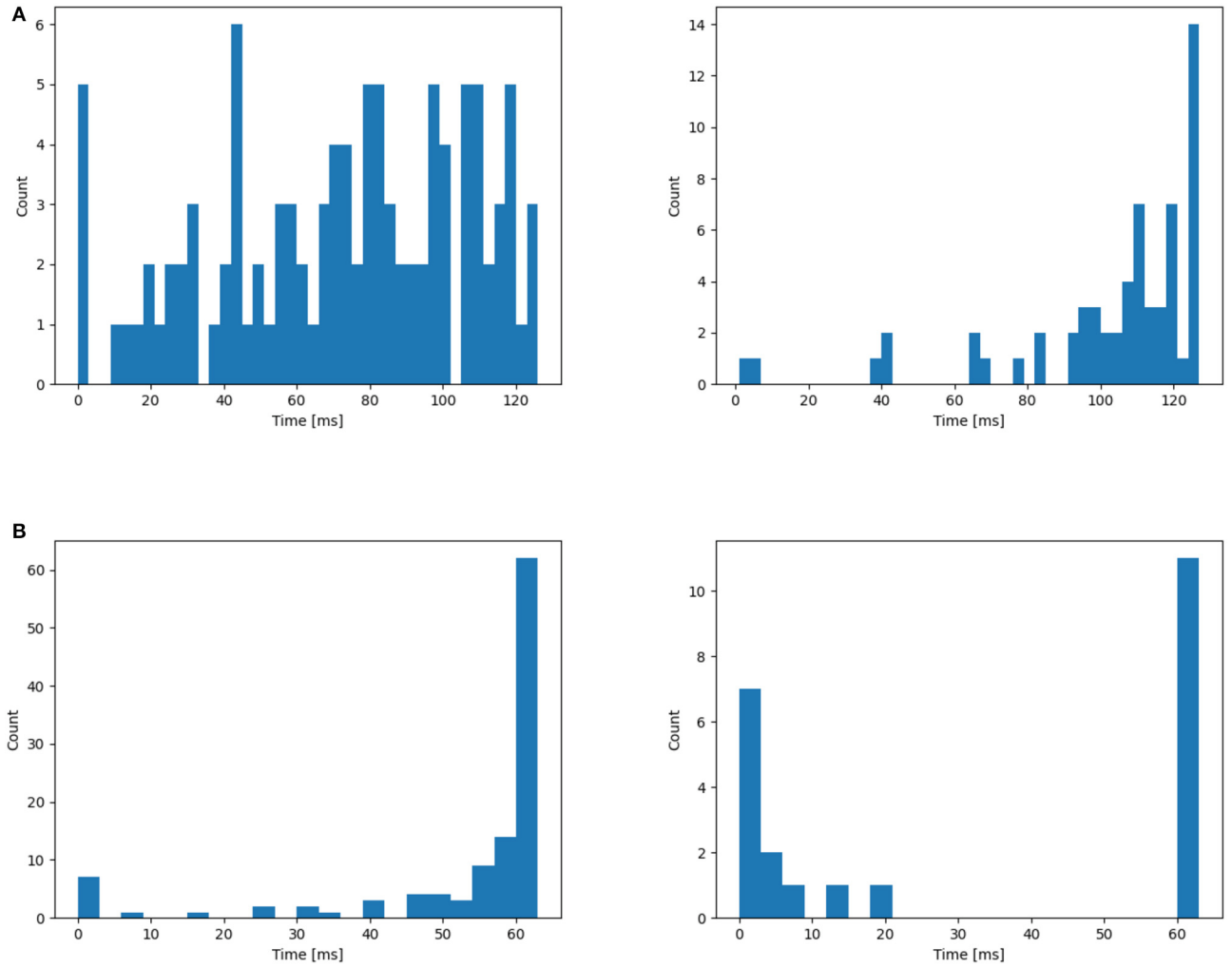


FIGURE 6
Application of the L2 regularizer on the distribution of time delay for **(A)** NTIDIDIGITS, **(B)** SHD. The initial distribution are all 0. From the left to right: First layer, second layer.

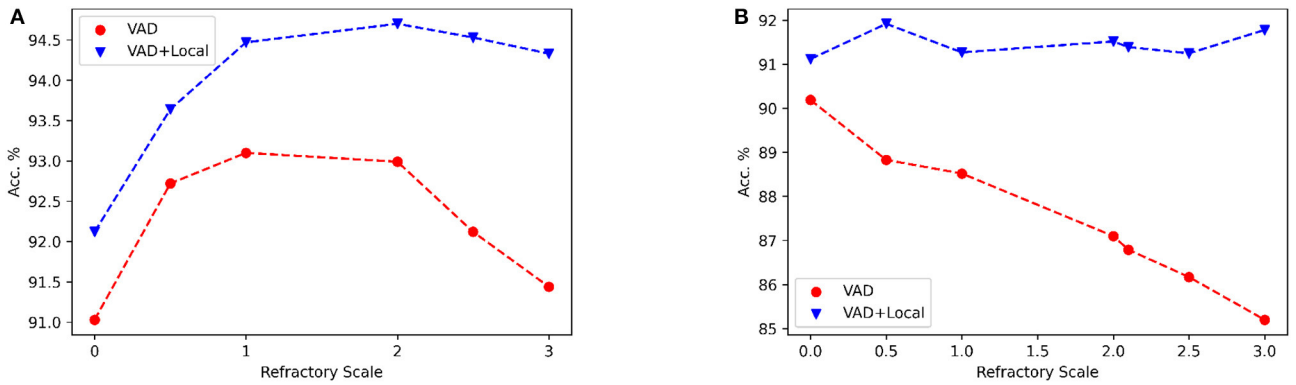


FIGURE 7
The influence of the different refractory scale α_r on accuracy is examined under "VAD" and "VAD+Local" architecture. "VAD" refers to the performance of using only the VAD module, while "VAD+Local" represents the performance using both VAD and local skip-connections. **(A)** NTIDIDIGITS dataset. **(B)** SHD dataset. For this experiment, we use two dense layers with 128 neurons.

varied tasks, indicating that a consistent alpha value can be effective for different tasks.

4. Conclusion

In this study, we introduce several innovative components aimed at enhancing the performance of Spiking Neural Networks (SNNs): the learnable axonal delay module, combined with a local skip connection architecture, and augmented with an auxiliary suppressed loss. The variable axonal delay module plays a pivotal role in aligning spike timing, thereby enhancing the network's capacity for representation. The local skip-connection mechanism compensates for the information loss during the reset process. This enhances network dynamics and reduces the sensitivity to refractory scale tuning, making it more versatile. The inclusion of the suppressed loss works to suppress erroneous neuron firing, facilitating the SNN in making more accurate label distinctions. Importantly, these methods can be seamlessly integrated into the existing framework through the use of backpropagation algorithms.

We demonstrate that the proposed methods boost performance on two benchmark event-based speech datasets with the fewest parameters. Our methods highlight the immense potential of employing them in tandem with a cochlear front-end that encodes features of auditory inputs using spikes, creating a robust bio-inspired system. Our work emphasizes the importance of delving into different dynamic SNN architectures and learning algorithms for tasks involving datasets with rich temporal complexity.

In future work, it will be interesting to investigate the spike count distribution per layer and the total computational cost. Additionally, more exploration could be focused on latency by studying the influence of different loss evaluations and dynamic caps for axonal delays. Since current work mainly focuses on cochlear features with a bio-inspired approach, it would also be intriguing to apply these methods to visual tasks that involve inherent temporal information.

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

PS: Conceptualization, Investigation, Software, Validation, Writing—original draft, Writing—review & editing. YC: Conceptualization, Investigation, Supervision, Writing—review & editing. PD: Supervision, Writing—review & editing, Conceptualization, Investigation. DB: Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Writing—review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was supported in part by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" and the Research Foundation - Flanders under Grant Number G0A0220N (FWO WithMe project). The work of YC was supported in part by the National Key Research and Development Program of China (Grant No. 2021ZD0200300).

Acknowledgments

The authors would express our very great appreciation to Sumit Bam Shrestha for his valuable and constructive suggestions and technical support during the development of this research work.

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.

The author(s) declared that they were an editorial board member of Frontiers, at the time of submission. This had no impact on the peer review process and the final decision.

Publisher's note

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). TrueNorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Design Integr. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Anumula, J., Neil, D., Delbruck, T., and Liu, S.-C. (2018). Feature representations for neuromorphic audio spike streams. *Front. Neurosci.* 12, 23. doi: 10.3389/fnins.2018.00023

Bittar, A., and Garner, P. N. (2022). A surrogate gradient spiking baseline for speech command recognition. *Front. Neurosci.* 16, 865897. doi: 10.3389/fnins.2022.865897

Blouw, P., and Eliasmith, C. (2020). "Event-driven signal processing with neuromorphic computing systems," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE), 8534–8538.

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Carr, C. E., and Konishi, M. (1988). Axonal delay lines for time measurement in the owl's brainstem. *Proc. Natl. Acad. Sci. U.S.A.* 85, 8311–8315.

Cramer, B., Stradmann, Y., Schemmel, J., and Zenke, F. (2020). The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 2744–2757.

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

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021). "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (IEEE), 2661–2671.

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

Hammouamri, I., Khalfaoui-Hassani, I., and Masquelier, T. (2023). Learning delays in spiking neural networks using dilated convolutions with learnable spacings. *arXiv preprint arXiv:2306.17670*.

Hong, C., Wei, X., Wang, J., Deng, B., Yu, H., and Che, Y. (2019). Training spiking neural networks for cognitive tasks: a versatile framework compatible with various temporal codes. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 1285–1296. doi: 10.1109/TNNLS.2019.2919662

Iyer, L. R., Chua, Y., and Li, H. (2021). Is neuromorphic MNIST neuromorphic? Analyzing the discriminative power of neuromorphic datasets in the time domain. *Front. Neurosci.* 15, 608567. doi: 10.3389/fnins.2021.608567

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Leonard, R. G., and Doddington, G. (1993). *Tidigits Speech Corpus*. IEEE: Texas Instruments, Inc.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Patiño-Saucedo, A., Yousefzadeh, A., Tang, G., Corradi, F., Linares-Barranco, B., and Sifalakis, M. (2023). "Empirical study on the efficiency of spiking neural networks with axonal delays, and algorithm-hardware benchmarking," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE), 1–5.

Perez-Nieves, N., Leung, V. C., Dragotti, P. L., and Goodman, D. F. (2021). Neural heterogeneity promotes robust learning. *Nat. Commun.* 12, 1–9. doi: 10.1038/s41467-021-26022-3

Seidl, A. H. (2014). Regulation of conduction time along axons. *Neuroscience* 276, 126–134. doi: 10.1016/j.neuroscience.2013.06.047

Shen, J., Xu, Q., Liu, J. K., Wang, Y., Pan, G., and Tang, H. (2023). ESL-SNNs: an evolutionary structure learning strategy for spiking neural networks. *arXiv preprint arXiv:2306.03693*.

Shrestha, S. B., and Orchard, G. (2018). "SLAYER: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31* (IEEE).

Shrestha, S. B., Zhu, L., and Sun, P. (2022). "Spikemax: spike-based loss methods for classification," in *2022 International Joint Conference on Neural Networks (IJCNN)* (IEEE), 1–7.

Stoelzel, C. R., Bereshpolova, Y., Alonso, J.-M., and Swadlow, H. A. (2017). Axonal conduction delays, brain state, and corticogeniculate communication. *J. Neurosci.* 37, 6342–6358. doi: 10.1523/JNEUROSCI.0444-17.2017

Sun, P., Eqlimi, E., Chua, Y., Devos, P., and Botteldooren, D. (2023a). "Adaptive axonal delays in feedforward spiking neural networks for accurate spoken word recognition," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE), 1–5.

Sun, P., Wu, J., Zhang, M., Devos, P., Botteldooren, D. (2023b). Delayed memory unit: modelling temporal dependency through delay gate. *arXiv preprint arXiv:2310.14982*.

Sun, P., Zhu, L., and Botteldooren, D. (2022). "Axonal delay as a short-term memory for feed forward deep spiking neural networks," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8932–8936.

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015). DL-resume: a delay learning-based remote supervised method for spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3137–3149. doi: 10.1109/TNNLS.2015.2404938

Talidou, A., Frankland, P. W., Mabbott, D., and Lefebvre, J. (2022). Homeostatic coordination and up-regulation of neural activity by activity-dependent myelination. *Nat. Comput. Sci.* 2, 665–676. doi: 10.1038/s43588-022-00315-z

Wang, X., Lin, X., and Dang, X. (2019). A delay learning algorithm based on spike train kernels for spiking neurons. *Front. Neurosci.* 13, 252. doi: 10.3389/fnins.2019.00252

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560.

Wu, J., Chua, Y., and Li, H. (2018a). "A biologically plausible speech recognition framework based on spiking neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)* (IEEE), 1–8.

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2021). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.*

Wu, J., Chua, Y., Zhang, M., Li, H., and Tan, K. C. (2018b). A spiking neural network framework for robust sound classification. *Front. Neurosci.* 12, 836. doi: 10.3389/fnins.2018.00836

Wu, J., Pan, Z., Zhang, M., Das, R. K., Chua, Y., and Li, H. (2019). "Robust sound recognition: a neuromorphic approach," in *Interspeech*, 3667–3668.

Wu, J., Yılmaz, E., Zhang, M., Li, H., and Tan, K. C. (2020). Deep spiking neural networks for large vocabulary automatic speech recognition. *Front. Neurosci.* 14, 199. doi: 10.3389/fnins.2020.00199

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018c). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Xu, Q., Li, Y., Fang, X., Shen, J., Liu, J. K., Tang, H., et al. (2023a). Biologically inspired structure learning with reverse knowledge distillation for spiking neural networks. *arXiv preprint arXiv:2304.09500*.

Xu, Q., Li, Y., Shen, J., Liu, J. K., Tang, H., and Pan, G. (2023b). "Constructing deep spiking neural networks from artificial neural networks with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE), 7886–7895.

Xu, Q., Li, Y., Shen, J., Zhang, P., Liu, J. K., Tang, H., et al. (2022). Hierarchical spiking-based model for efficient image classification with enhanced feature extraction and encoding. *IEEE Trans. Neural Netw. Learn. Syst.* doi: 10.1109/TNNLS.2022.3232106

Xu, Q., Qi, Y., Yu, H., Shen, J., Tang, H., Pan, G., et al. (2018). "CSNN: an augmented spiking based framework with perceptron-inception," in *IJCAI*, 1646.

Xu, Q., Shen, J., Ran, X., Tang, H., Pan, G., and Liu, J. K. (2021). Robust transcoding sensory information with neural spikes. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1935–1946. doi: 10.1109/TNNLS.2021.3107449

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., et al. (2021). "Temporal-wise attention spiking neural networks for event streams classification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (IEEE), 10221–10230.

Yılmaz, E., Gevrek, O. B., Wu, J., Chen, Y., Meng, X., and Li, H. (2020). "Deep convolutional spiking neural networks for keyword spotting," in *Proceedings of Interspeech*, 2557–2561.

Yin, B., Corradi, F., and Bohté, S. M. (2020). "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, 1–8.

Yin, B., Corradi, F., and Bohté, S. M. (2021). Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nat. Mach. Intell.* 3, 905–913. doi: 10.1038/s42256-021-00397-w

Yu, Q., Ma, C., Song, S., Zhang, G., Dang, J., and Tan, K. C. (2022). Constructing accurate and efficient deep spiking neural networks with double-threshold and augmented schemes. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1714–1726. doi: 10.1109/TNNLS.2020.3043415

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2021). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1947–1958. doi: 10.1109/TNNLS.2021.3110991

Zhang, M., Wu, J., Belatreche, A., Pan, Z., Xie, X., Chua, Y., et al. (2020). Supervised learning in spiking neural networks with synaptic delay-weight plasticity. *Neurocomputing* 409, 103–118. doi: 10.1016/j.neucom.2020.03.079

Zhang, M., Wu, J., Chua, Y., Luo, X., Pan, Z., Liu, D., et al. (2019). "MPD-AL: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons," in *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, W., and Li, P. (2019). Spike-train level backpropagation for training deep recurrent spiking neural networks. *arXiv preprint arXiv:1908.06378*.

Zhang, W., and Li, P. (2021). Skip-connected self-recurrent spiking neural networks with joint intrinsic parameter and synaptic weight training. *Neural Comput.* 33, 1886–1913. doi: 10.1162/neco_a_01393

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 10 November 2023
DOI 10.3389/fnins.2023.1261543

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Malu Zhang,
National University of Singapore, Singapore
Man Yao,
Xi'an Jiaotong University, China
Fangwen Yu,
Tsinghua University, China

*CORRESPONDENCE
Yong Song
✉ yongsong@bit.edu.cn
Ya Zhou
✉ zhouya@bit.edu.cn

RECEIVED 20 July 2023
ACCEPTED 23 October 2023
PUBLISHED 10 November 2023

CITATION
Wu X, Song Y, Zhou Y, Jiang Y, Bai Y, Li X and Yang X (2023) STCA-SNN: self-attention-based temporal-channel joint attention for spiking neural networks.
Front. Neurosci. 17:1261543.
doi: 10.3389/fnins.2023.1261543

STCA-SNN: self-attention-based temporal-channel joint attention for spiking neural networks

Xiyan Wu, Yong Song*, Ya Zhou*, Yurong Jiang, Yashuo Bai, Xinyi Li and Xin Yang

School of Optics and Photonics, Beijing Institute of Technology, Beijing, China

Spiking Neural Networks (SNNs) have shown great promise in processing spatio-temporal information compared to Artificial Neural Networks (ANNs). However, there remains a performance gap between SNNs and ANNs, which impedes the practical application of SNNs. With intrinsic event-triggered property and temporal dynamics, SNNs have the potential to effectively extract spatio-temporal features from event streams. To leverage the temporal potential of SNNs, we propose a self-attention-based temporal-channel joint attention SNN (STCA-SNN) with end-to-end training, which infers attention weights along both temporal and channel dimensions concurrently. It models global temporal and channel information correlations with self-attention, enabling the network to learn 'what' and 'when' to attend simultaneously. Our experimental results show that STCA-SNNs achieve better performance on N-MNIST (99.67%), CIFAR10-DVS (81.6%), and N-Caltech 101 (80.88%) compared with the state-of-the-art SNNs. Meanwhile, our ablation study demonstrates that STCA-SNNs improve the accuracy of event stream classification tasks.

KEYWORDS

spiking neural networks, self-attention, temporal-channel, neuromorphic computing, event streams

1. Introduction

As the representatives of mimicking the human brain at the neuronal level, Spiking Neural Networks (SNNs) have gained great attraction for the high biological plausibility, event-driven property, and high energy efficiency (Rieke et al., 1999; Gerstner et al., 2014; Bellec et al., 2018). Using time as an additional input dimension, SNNs record valuable information in a sparse manner and deliver information through spikes only when the membrane potential reaches the firing threshold (Mainen and Sejnowski, 1995). Inspired by biological visual processing mechanisms, Dynamic Vision Sensors (DVS) encode the time, location, and polarity of the brightness changes per pixel into event streams (Lichtsteiner et al., 2008; Posch et al., 2010). With its unique advantages of high event rate, high dynamic range, and fewer resource requirements (Gallego et al., 2020), DVS has broad application prospects in various visual tasks, such as autonomous driving (Cheng et al., 2019), high-speed object tracking (Rebecq et al., 2019), optical flow estimation (Ridwan and Cheng, 2017), and action recognition (Amir et al., 2017). Event-based vision is one of the typical advantage application scenarios of SNNs, providing a platform for demonstrating the capabilities of spiking neurons to process information with spatio-temporal dynamics.

Although the intrinsic time-dependent neuron dynamics endows SNNs with the ability to process spatio-temporal information, there remains a performance gap between SNNs and

ANNs. Recently, ANNs' modules (Hu et al., 2021; Yang et al., 2021; Yao et al., 2021, 2023c) have been integrated into SNNs to improve the performance of SNNs. CSNN (Xu et al., 2018) first validated the application of convolution structure on SNNs, promoting the development of SNNs. Convolution-based SNNs share weights across both temporal and spatial dimensions, following the assumption of spatio-temporal invariance (Huang et al., 2022). This approach can be regarded as a local way of information extraction since convolutional operations can only process a local neighborhood at a time, either in space or time. However, when dealing with sequential data like event streams, capturing long-distance dependencies is of central importance to modeling complex temporal dynamics. Non-local operations (Wang et al., 2018) provided a solution as a building block by computing the response at a position as a weighted sum of the features at all positions. The range of positions can span across space, time, or spacetime, allowing non-local operators to achieve remarkable success in vision attention.

The attention mechanism is inspired by the human ability to selectively find prominent areas in complex scenes (Itti et al., 1998). A popular research direction is to present attention as a lightweight auxiliary unit to improve the representation power of the basic model. In the ANNs domain, Ba et al. (2014) first introduced the term "visual attention" for image classification tasks, utilizing attention to identify relevant regions and locations within the input image. This approach also reduces the computational complexity of the proposed model regarding the size of the input image. SENet (Hu et al., 2018) was introduced to reweight the channel-wise responses of the convolutional features, determining "what" to pay attention to. CBAM (Woo et al., 2018) inferred attention maps sequentially along channel-wise and spatial dimensions for refining the input feature, determining "what" and "where" to pay attention to concurrently. In the SNNs domain, TA-SNN (Yao et al., 2021) first extended the channel-wise attention concept to temporal-wise attention and integrated it into SNNs to determine 'when' to pay attention. MA-SNN (Yao et al., 2023c) extended CBAM to SNNs and proposed a multi-dimensional attention module along temporal-wise, channel-wise, and spatial-wise separately or simultaneously. Recently, TCJA-SNN (Zhu et al., 2022) cooperated temporal-wise and channel-wise attention correlations using the 1-D convolution operation to present the correlation between time-steps and channels. However, the receptive field of TCJA-SNN is a local cross shape that is restricted by its convolution kernels, shown in Figure 1A. Thus long-range dependencies can only be captured when 1-D convolution operation is repeated, which makes multi-hop dependency modeling difficult. On the other hand, self-attention, another vital feature of the human biological system, possesses the ability to capture feature dependencies effectively as an additional non-local operator alongside SE and CBAM. It has sparked a significant wave of interest and achieved remarkable success in various tasks (Vaswani et al., 2017; Dosovitskiy et al., 2020; Liu et al., 2021). Intuitively, there is a compelling interest in investigating the application of self-attention in SNNs to advance deep learning, when considering the biological characteristics of both mechanisms (Yao et al., 2023a,b; Zhou C. et al., 2023; Zhou Z. et al., 2023).

To address the local spatio-temporal receptive field limitation of TCJA, we first adopt self-attention, a non-local operation, to model global temporal and channel information correlations. The self-attention module we employed can capture the global spatio-temporal receptive field, as shown in Figure 1B, allowing for the direct long-range dependencies modeling, which is the highlight of our work. We propose a plug-and-play Self-attention-based Temporal-Channel joint Attention (STCA) module for SNNs with end-to-end training. The STCA-SNNs can learn to focus on different features of the input at each time-step. In other words, the STCA-SNNs can learn 'when' and 'what' to attend concurrently, enhancing the ability of the SNNs to process temporal information. We evaluated the effectiveness of STCA-SNNs across different architectures on three benchmark event stream classification datasets: N-MNIST, CIFAR10-DVS, and N-Caltech 101. Our detailed experiments show that STCA-SNNs achieve competitive accuracy with existing state-of-the-art SNNs.

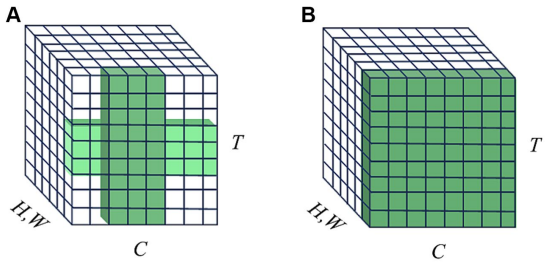


FIGURE 1
Illustration of receptive fields on channel and temporal domains. *T* means the temporal domain, *C* means the channel domain, and *H*, *W* represent the spatial domain. **(A)** TCJA-SNN utilizes two local attention mechanisms with 1-D convolution along temporal-wise and channel-wise, respectively, then fuse them, forming a cross-shaped receptive field. **(B)** STCA-SNN uses self-attention operation to establish temporal-wise and channel-wise correlations, forming a global receptive field.

The main contributions of our work are summarized as follows:

1. We propose STCA-SNNs for event streams that can undertake end-to-end training and inference tasks.
2. The plug-and-play STCA module models global temporal and channel correlations with self-attention, allowing the network to learn 'when' and 'what' to attend simultaneously. This enhances the ability of SNNs to process temporal information.
3. We evaluate the performance of STCA-SNNs on three benchmark event stream classification datasets, N-MNIST, CIFAR10DVS, and N-Caltech 101. Our experimental results demonstrate that STCA-SNNs achieve competitive accuracy compared to existing state-of-the-art SNNs.

2. Related work

2.1. Attention in SNNs

Spiking neural networks benefit from biological plausibility and continuously pursue the combination with brain mechanisms. The attention mechanism draws inspiration from the human ability to selectively identify salient regions within complex scenes and has gained remarkable success in deep learning by allocating attention weights preferentially to the most informative input components. A popular research direction is to present attention as an auxiliary module that can be easily integrated with existing architectures to

boost the representation power of the basic model (Hu et al., 2018; Woo et al., 2018; Guo et al., 2022; Li et al., 2022). Yao et al. (2021) first suggested using an extra plug-and-play temporal-wise attention module for SNNs to bypass a few unnecessary input timesteps. Then they proposed a multi-dimensional attention module along temporal-wise, channel-wise, and spatial-wise separately or simultaneously to optimize membrane potentials, which in turn regulate the spiking response (Yao et al., 2023c). STSC-SNN (Yu et al., 2022) employed temporal convolution and attention mechanisms to improve spatio-temporal receptive fields of synaptic connections. SCTFA-SNN (Cai et al., 2023) computed channel-wise and spatial-wise attention separately to optimize membrane potentials along the temporal dimension. Yao et al. (2023a,b) recently proposed an advanced spatial attention module to harness SNNs' redundancy, which can adaptively optimize their membrane potential distribution by a pair of individual spatial attention sub-modules. TCJA-SNN (Zhu et al., 2022) cooperated temporal-wise joint channel-wise attention correlations using 1-D convolution operation. However, the temporal-channel receptive field of TCJA is a local cross shape that is restricted by its convolution kernels, requiring multiple repeated computations to establish long-range dependencies of features. Therefore, it is computationally inefficient and makes multi-hop dependency modeling difficult.

Among the attention mechanisms, self-attention, as another important feature of the human biological system, possesses the ability to capture feature dependencies. Originally developed for natural language processing (Vaswani et al., 2017), self-attention has been extended to computer vision, where it has achieved significant success in various applications. The self-attention module can also be considered a building block of CNN architectures, which are known for their limited scalability when it comes to large receptive fields (Han et al., 2022). In contrast to the progressive behavior of convolution operation, self-attention can capture long-range dependencies directly by computing interactions between any two positions, regardless of their positional distance. Moreover, it is commonly integrated into the top of the networks to enhance high-level semantic features for vision tasks. Recently, an emerging research direction is to explore the biological characteristics associated with the fusion of self-attention and SNNs (Yao et al., 2023a,b; Zhou C. et al., 2023; Zhou Z. et al., 2023). These efforts primarily revolve around optimizing the computation of self-attention within SNNs by circumventing multiplicative operations, leading to performance degradation. Diverging from these studies, our primary goal is to explore how self-attention can enhance the spatio-temporal information processing capabilities of SNNs.

2.2. Learning algorithms for SNNs

Existing SNN training methods can be roughly divided into three categories: 1) the biologically plausible method, 2) the conversion method, and 3) the gradient-based direct training method. The first one is based on biological plausible local learning rules, like spike timing dependent plasticity (STDP) (Diehl and Cook, 2015; Kheradpisheh et al., 2018) and ReSuMe (Ponulak and Kasinski, 2010), but achieving high performance for deep networks is challenging. The conversion method offers an alternative way to obtain high-performance SNNs by converting a well-trained ANN and mapping its parameters to an SNN with an equivalent architecture, where the firing rate of the SNN acts as ReLU activation (Cao et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Ding et al., 2021; Bu et al., 2022; Wu et al., 2023). Moreover, some works explored post-conversion fine-tuning of converted SNNs to reduce latency and increase accuracy (Rathi et al., 2020; Rathi and Roy, 2021; Wu et al., 2021). However, this method is not suitable for neuromorphic datasets. The gradient-based direct training methods primarily include voltage gradient-based (Zhang et al., 2020), timing gradient-based (Zhang et al., 2021), and activation gradient-based approaches. Among them, the activation gradient-based method demonstrates notable effectiveness when performing challenging tasks. This approach uses surrogate gradients to address the non-differentiable spike activity issue, allowing for error back-propagation through time (BPTT) to interface with gradient descent directly on SNNs for end-to-end training (Neftci et al., 2019; Wu et al., 2019; Yang et al., 2021; Zenke and Vogels, 2021). These efforts have shown strong potential in achieving high performance by exploiting spatio-temporal information. However, further research is required to determine how to make better use of spatio-temporal data and how to efficiently extract spatio-temporal features. This is what we want to contribute.

3. Materials and methods

In this section, we first present the representation of event streams and the adopted spiking neuron model and later propose our STCA module based on this neuron model. Finally, we introduce the training method adopted in this paper.

3.1. Representation of event streams

An event, e, encodes three pieces of information: the pixel location (x, y) of the event, the timestamp t' recording the time when the event is triggered, and the polarity of each single event $p \in \{-1, +1\}$ reflecting an increase or decrease of brightness via $+1/-1$. Formally, a set of events at the timestamp t' can be defined as:

$$\boldsymbol{E}_{\boldsymbol{t}'} = \left\{\left[\boldsymbol{x}_{\boldsymbol{k}}, \boldsymbol{y}_{\boldsymbol{k}}, \boldsymbol{t}', \boldsymbol{p}_{\boldsymbol{k}}\right]\right\}_{\boldsymbol{k}=\boldsymbol{1}}^{\boldsymbol{N}} \tag{1}$$

Assume the spatial resolution is $h \times w$, the event set equals to the spike pattern tensor $S_t \in \mathrm{R}^{2\times \mathrm{h}\times \mathrm{w}}$ at the timestamp t'. However, processing these events one by one can be inefficient due to the limited amount of information contained in a single event. We follow the frame-based representation in SpikingJelly (Fang et al., 2020) that transforms event streams into high-rate frame sequences during preprocessing. Each frame includes many blank (zero) areas, and SNNs can skip the computation of the zero areas in each input frame (Roy et al., 2019), improving overall efficiency.

3.2. Spiking neural models

Spiking neuron in SNNs integrates synaptic inputs from the previous layer and the residual membrane potential into the latest membrane potential. The Parametric Leaky integrate-and-fire (PLIF)

model can learn the synaptic weight and membrane time constant simultaneously, which can enhance the learning capabilities of SNNs (Fang et al., 2021). The subthreshold dynamics of the PLIF neuron is defined as:

$$\tau \frac{dV(t)}{dt} = -\left(V(t) - V_{rest}\right) + X(t) \tag{2}$$

where $V(t)$ indicates the membrane potential of the neuron at time t, τ is the membrane time constant that controls the decay of $V(t)$, $X(t)$ is the input collected from the presynaptic neurons and V_{rest} is the resting potential. When the membrane potential $V(t)$ exceeds the neuron threshold at time t, the neuron will emit a spike, and then the membrane potential goes back to a reset value V_{rest}. We set $V_{rest} = V_{reset} = 0$. The iterative representation of the PLIF model can be described as follows:

$$\begin{cases} \boldsymbol{H}^{t,l} = \boldsymbol{V}^{t-1,l} + \dfrac{1}{\tau}\left(-\left(\boldsymbol{V}^{t-1,l} - \boldsymbol{V}_{reset}\right) + \boldsymbol{X}^{t,l}\right) \\ \boldsymbol{S}^{t,l} = \Theta\left(\boldsymbol{H}^{t,l} - \boldsymbol{V}_{th}\right) \\ \boldsymbol{V}^{t,l} = \left(\mathbf{1} - \boldsymbol{S}^{t,l}\right)\boldsymbol{H}^{t,l} + \boldsymbol{V}_{reset}\boldsymbol{S}^{t,l} \end{cases} \tag{3}$$

where superscripts t and l indicate the time step and layer index. To avoid confusion, we use $H^{t,l}$ and $V^{t,l}$ to represent the membrane potential after neuronal dynamics and after the trigger of a spike in layer l at time-step t, respectively. V_{th} is the firing threshold. $S^{t,l}$ is determined by $\Theta(x)$, the Heaviside step function that outputs 1 if $x \geq 0$ or 0 otherwise. The time constant $\tau = 1/k(a)$, $k(a)$ is a sigmoid function $1/(1+\exp(-a))$ with a trainable parameter a.

3.3. Self-attention-based temporal-channel joint attention module

The processing of temporal information in SNNs is generally attributed to spiking neurons because their dynamics naturally depend on the temporal dimension. However, the LIF neuron and its variants including the PLIF neuron, only sustain very weak temporal linkages. Additionally, event streams are inherently time-dependent therefore, it is necessary to establish spatial–temporal correlations to improve data utilization. The focus of this work is to model temporal-wise and channel-wise attention correlations globally by adopting a self-attention mechanism. We present our idea of attention with a pluggable module termed the Self-attention-based Temporal-Channel joint Attention (STCA), which is depicted in Figure 2.

Formally, we collect intermediate the spatial feature of l-th layer at all time-steps $X^l = [\cdots, X^{t,l}, \cdots] \in \mathrm{R}^{\mathrm{T}\times C\times H\times W}$ as the input of STCA module, where T is time-step, C denotes channels, H and W are height and width of the feature, respectively. The spatial feature $X^{t,l}$ can be extracted from the original input $S^{t,l}$:

$$\boldsymbol{X}^{t,l} = \mathbf{BN}\left(\mathbf{Conv}\left(\mathbf{W}^l, \boldsymbol{S}^{t,l-1}\right)\right) \tag{4}$$

where BN $(\cdot)$ and Conv $(\cdot)$ mean the batch normalization and convolutional operation, W^l is the weight matrix, $S^{t,l-1}$ $(l \neq 1)$ is a spike tensor that only contains 0 and 1, and $X^{t,l} \in R^{C_l \times H \times W_l}$. To simplify the notation, bias terms are omitted. BN is a default operation following the Conv, we also omit it in the rest of this paper. Since each spatial feature $X^{t,l}$ in X^l is time-dependent, our idea of attention is to utilize the temporal correlation of these features. It is well known that each channel of feature maps corresponds to a specific visual pattern. Our STCA module aims to determine 'when' to attend to 'what' are semantic attributes of the given input. For efficiency, STCA only focuses on temporal and channel modeling, the spatial information of the feature is aggregated by using both avg-pooling and max-pooling operations as follows:

$$\boldsymbol{R}^l = \mathbf{AvgPool}\left(\boldsymbol{X}^l\right) + \mathbf{MaxPool}\left(\boldsymbol{X}^l\right) \tag{5}$$

where AvgPool $(\cdot)$ and MaxPool $(\cdot)$ represent the outputs of the avg-pooling and max-pooling layer respectively, $R^l \in \mathrm{R}^{\mathrm{T}\times C}$. The generated different temporal-channel context descriptors, avg-pooled features and max-pooled features, are merged and then fed into a self-attention (SA) block. We follow the convention (Wang et al., 2018) to formulate the SA block, where the input feature in layer l is $R^l \in \mathrm{R}^{\mathrm{T}\times C}$, and the output feature is generated as:

$$\boldsymbol{a}_i^l = \frac{1}{\boldsymbol{C}(\boldsymbol{r}_i)} \sum_{\forall j} \boldsymbol{f}\left(\boldsymbol{r}_i, \boldsymbol{r}_j\right) \boldsymbol{g}\left(\boldsymbol{r}_j\right) \tag{6}$$

where $r_i \in \mathrm{R}^{1\times C}$ and $a_i \in \mathrm{R}^{1\times C}$ indicate the i^{th} position of the input feature R^l and output feature A^l, respectively. Subscript j is the index that enumerates all positions along the temporal domain, i.e., i, $j \in [1,2,\ldots, \mathrm{T}]$, and a pairwise function $f(\cdot)$ computes a representing relationship between i and all j. The function $g(\cdot)$ computes a representation of the input signal at time-step j, and the response is normalized by a factor $C(r_i)$. We use a simple extension of the Gaussian function to compute the similarity in an embedding space, and the function $f(\cdot)$ can be formulated as:

$$\boldsymbol{f}\left(\boldsymbol{r}_i, \boldsymbol{r}_j\right) = \boldsymbol{e}^{\boldsymbol{\theta}(\boldsymbol{r}_i)\varnothing(\boldsymbol{r}_j)^T} \tag{7}$$

where $\theta(\cdot)$ and $\phi(\cdot)$ can be any embedding layers. If we consider the $\theta(\cdot)$, $\phi(\cdot)$, $g(\cdot)$ in the form of linear embedding: $\theta(R^l) = R^l W_\theta$, $\phi(R^l) = R^l W_\phi$, $g(R^l) = R^l W_g$, where $W_\theta \in \mathrm{R}^{C\times C_k}$, $W_\phi \in \mathrm{R}^{C\times C_k}$, $W_g \in \mathrm{R}^{C\times C_k}$, and set the normalization factor as $\boldsymbol{C}(\boldsymbol{r}_i) = \sum_{\forall j} \boldsymbol{f}\left(\boldsymbol{r}_i, \boldsymbol{r}_j\right)$, the Eq. 6 can be rewritten as:

$$\boldsymbol{a}_i^l = \frac{\boldsymbol{e}^{\boldsymbol{r}_i \boldsymbol{w}_{\theta,i} \boldsymbol{w}_{\varnothing,j}^T \boldsymbol{r}_j^T}}{\sum_j \boldsymbol{e}^{\boldsymbol{r}_i \boldsymbol{w}_{\theta,i} \boldsymbol{w}_{\varnothing,j}^T \boldsymbol{r}_j^T}} \boldsymbol{r}_j \boldsymbol{w}_{g,j} \tag{8}$$

where $\boldsymbol{w}_{\theta,i} \in \mathrm{R}^{C\times 1}$ is the i^{th} row of the weight matrix W_θ. For a given index i, $\frac{1}{\boldsymbol{C}(\boldsymbol{r}_i)} \boldsymbol{f}\left(\boldsymbol{r}_i, \boldsymbol{r}_j\right)$ becomes the softmax output along the dimension j. The formulation can be future rewritten as:

$$\boldsymbol{A}^l = \mathbf{softmax}\left(\boldsymbol{R}^l \boldsymbol{W}_\theta \boldsymbol{W}_\varnothing^T \boldsymbol{R}^l\right) \boldsymbol{g}\left(\boldsymbol{R}^l\right) \tag{9}$$

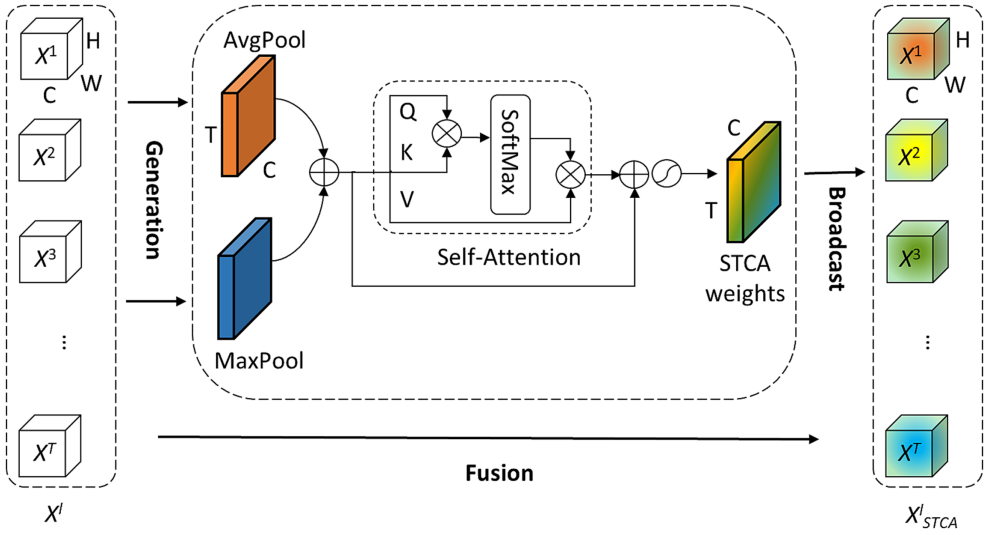


FIGURE 2
Diagram of the STCA module. The STCA module first aggregates spatial information by average-pooling and max-pooling then merges them and feeds it into a self-attention block to establish the correlations in both temporal and channel dimensions.

where $A^l \in \mathrm{R}^{\mathrm{T}\times\mathrm{C}}$ is the output feature of the same size as R^l. Given the query, key, and value representations:

$$\boldsymbol{Q} = \boldsymbol{R^l W^Q}, \boldsymbol{K} = \boldsymbol{R^l W^K}, \boldsymbol{V} = \boldsymbol{R^l W^V} \tag{10}$$

Once $\boldsymbol{W^Q} = \boldsymbol{W_\theta}$, $\boldsymbol{W^K} = \boldsymbol{W_\phi}$, $\boldsymbol{W^V} = \boldsymbol{W_g}$, $W^Q \in \mathrm{R}^{\mathrm{C}\times\mathrm{C}}$, $W^K \in \mathrm{R}^{\mathrm{C}\times\mathrm{C}}$, and $W^V \in \mathrm{R}^{\mathrm{C}\times\mathrm{C}}$, Eq. 9 can be formulated as:

$$\boldsymbol{A^l} = \boldsymbol{softmax}\left(\boldsymbol{QK^T}\right)\boldsymbol{V} \tag{11}$$

In this way, the SA block is constructed. Then we employ a residual connection around the SA block. Finally, the attention process of STCA can be formulated as:

$$\boldsymbol{X^l_{STCA}} = \boldsymbol{f} \odot \boldsymbol{X^l} \tag{12}$$

where $f = \sigma(R^l + A^l) \in R^{\mathrm{T}\times\mathrm{C}}$ is the weight vector of STCA, $\odot$ is element-wise multiplication, σ is the sigmoid function, and $X^l_{STCA} \in \mathrm{R}^{\mathrm{T}\times\mathrm{C}\times\mathrm{H}\times\mathrm{W}}$ denotes the feature extracted by the STCA module along temporal and channel dimensions.

3.4. Training

We integrate the STCA module into networks and utilize the BPTT method to train SNNs. Since the process of neuron firing is non-differentiable, we use the derived ATan surrogate function $\sigma'(x) = \alpha / 2(1 + \pi\alpha x / 2)^2$. For a given input with label n, the neuron that represents class n has the highest excitatory level while other neurons remain silent. So the target output is defined by $Y = [y^{t,i}]$ with $y^{t,i} = 1$ for $i = n$, and $y^{t,i} = 0$ for $i \neq n$. Then the loss function is described by the spike mean squared error:

$$\boldsymbol{L} = \left\| y^i - \frac{1}{T}\sum_{t=1}^{T} o^{t,i} \right\|^2 \tag{13}$$

where $O = [o^{t,i}]$ is the average spiking events of neurons under the voting strategy.

4. Experiments

4.1. Experimental setup

4.1.1. Implementation details

We implement our experiments with the Pytorch package and SpikingJelly framework. All experiments were conducted using the BPTT learning algorithm on 4 NVIDIA RTX 2080 Ti GPUs. We utilized the Adam optimizer (Kingma and Ba, 2015) to accelerate the training process and implemented some standard training techniques of deep learning such as batch normalization and dropout. The corresponding hyper-parameters and SNN hyper-parameters are shown in Table 1. We verify our method on the following DVS benchmarks:

CIFAR10-DVS contains 10 K DVS images of 10 classes recorded with the dynamic vision sensor from the original static CIFAR10 dataset. We apply a 9: 1 train-valid split (i.e., 9 k training images and 1 k validation images). The resolution is 128 × 128, we resize all of them to 48 × 48 in our training and we integrate the event data into 10 frames per sample (Li et al., 2017).

N-Caltech 101 dataset contains 8,831 DVS images converted from the original version of Caltech 101 with a slight change in object classes to avoid confusion. The N-Caltech 101 consists of 100 object classes plus one background class. Similarly, we apply the 9: 1 train-test split as CIFAR10-DVS. We use the SpikingJelly (Fang et al., 2020) package to process the data and integrate them into 14 frames per sample (Orchard et al., 2015).

TABLE 1 Hyper-parameter setting.

Hyperparameter	N-MNIST	CIFAR10-DVS	N-Caltech 101
Max Epoch	500	1,000	500
Automatic mixed precision	×	×	✓
Batch size	64	32	8
Learning rate	1e-3	1e-3	1e-3
Time step	10	10	14
V_{th}	1.0	1.0	1.0
τ_0	2.0	2.0	2.0
head	4	4	4

TABLE 2 The network structures with STCA for different datasets.

Dataset	Network structure
N-MNIST	Input-128C3-Neuron-MP2-128C3-Neuron-STCA-MP2-0.5DP-2048FC-Neuron-0.5DP-100FC-Neuron-Voting
CIFAR10-DVS	Input-64C3-Neuron-128C3-Neuron-AP2-256C3-Neuron-256C3-Neuron-STCA-AP2-512C3-Neuron -512C3-Neuron-STCA-AP2-512C3-Neuron-512C3-Neuron-AP2-10FC-Neuron
N-Caltech 101	64C3-Neuron-MP2-128C3-Neuron-MP2-256C3-Neuron-STCA-MP2-256C3-Neuron-STCA-MP2-512C3-Neuron-0.8DP-1024FC-Neuron-0.5DP-101FC-Neuron

*x*C*y*/MP*y*/AP*y* denotes the Conv2D/MaxPooling/Avgpooling layer with output channel = *x*, and kernel size = $y \times y$, *n*FC denotes the fully connected layer with output feature = *n*, MP*y* is the spiking dropout layer with dropout ratio *m*. BN follows behind all *x*C*y*.

TABLE 3 Accuracy performance comparison between the proposed method and the SOTA methods on different datasets.

Method	Binary spikes	N-MNIST		CIFAR10-DVS		N-Caltech 101	
		T	Acc. (%)	T	Acc. (%)	T	Acc. (%)
tdBN (Yang et al., 2021)	✓	–	–	10	67.8	–	–
Rollout (Kugele et al., 2020)	✓	32	99.57	48	66.97	–	–
LIAF-Net (Wu et al., 2019)	×	20	99.13	10	70.4	–	–
ConvSNN (Samadzadeh et al., 2023)	✓	-	99.6	-	69.2	–	–
PLIF (Fang et al., 2021)	✓	10	99.61	20	74.80	–	–
TA-SNN (Yao et al., 2021)	×	–	–	10	72.0	–	–
SALT (Kim and Panda, 2021)	✓	–	–	20	67.1	20	55.0
STSC-SNN (Yu et al., 2022)	✓	10	99.64	10	81.4[a]	–	–
TCJA-SNN (Zhu et al., 2022)	✓	–	–	10	80.7[a]	14	78.5
This work	✓	10	99.67	10	81.6[a]	14	80.88

[a]With data augmentation.

The neuromorphic MNIST dataset is a converted dataset from the original static MNIST dataset (Orchard et al., 2015). It contains 50 K training images and 10 K validation images. We integrate the event data into 10 frames per sample using SpikingJelly (Fang et al., 2020) package.

4.1.2. Networks

The network structures with STCA for different datasets are provided in Table 2 and the network architectures we use have been proven to perform quite well on each dataset. Specifically, for the CIFAR10-DVS dataset, we adopt a VGG11-like architecture. To mitigate the apparent overfitting on the CIFAR10-DVS dataset, we adopt the neuromorphic data augmentation, including horizontal Flipping and Mixup in each frame, which is also used in Zhu et al. (2022) for training the same dataset. For the N-Caltech 101 dataset, we adopt the same architecture with Zhu et al. (2022) and N-MNIST refers to PLIF Fang et al. (2021). The voting layers are implemented using average pooling for classification robustness.

4.2. Comparison with existing state-of-the-art works

Table 3 displays the accuracy performance of the proposed STCA-SNNs compared to other competing methods on three neuromorphic datasets, N-MNIST, CIFAR10-DVS, and N-Caltech 101. We mainly include direct training results of SNNs with signal transmission via

TABLE 4 Accuracy of vanilla SNN, TA-SNN, and STCA-SNN models on different datasets.

Model	N-MNIST	CIFAR10-DVS	N-Caltech 101
Vanilla SNN	99.64	80.7	79.40
TA-SNN	99.64	81.3	80.76
STCA-SNN	99.67	81.6	80.88

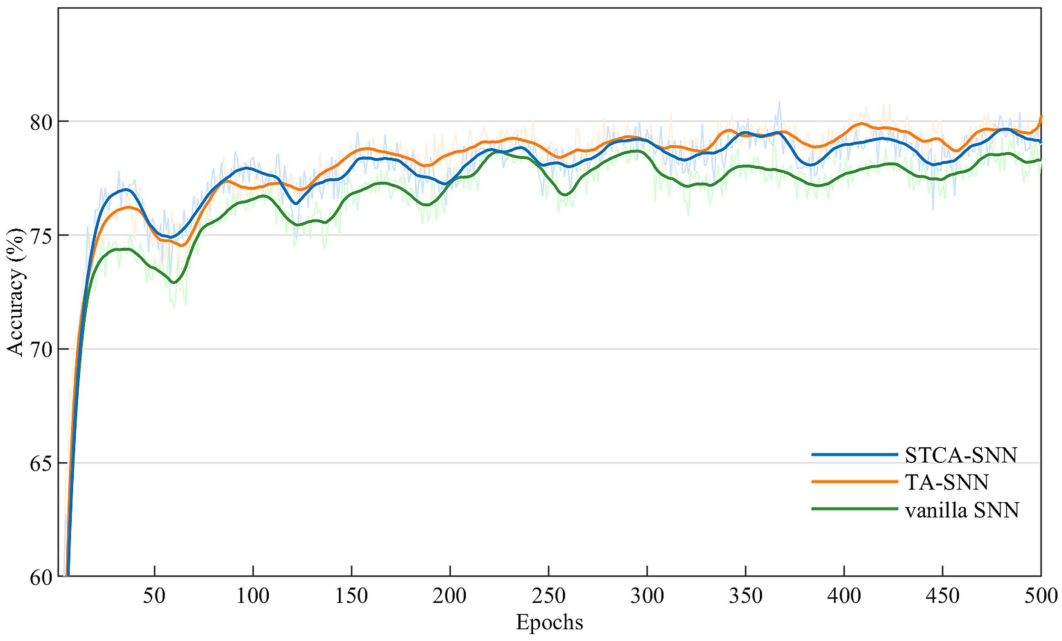


FIGURE 3
Convergence of compared SNN methods on N-Caltech101 dataset.

binary spike. Among them, some works (Wu et al., 2019; Yao et al., 2021) replace binary spikes with floating-point spikes and maintain the same forward pipeline as SNNs to obtain enhanced classification accuracy. STCA-SNNs achieve better performance than existing state-of-the-art SNNs on all datasets. We first compare our method on the CIFAR10-DVS dataset. We continue to utilize MSE the loss function and the same network architecture as TCJA-SNN (Zhu et al., 2022) and STSC-SNN (Yu et al., 2022) to preserve the consistency of this work, and our method reaches 81.6% top-1 accuracy, improving the accuracy by 0.9% over TCJA-SNN (Zhu et al., 2022). We also compare our method on N-Caltech 101dataset. Under the same condition as TCJA-SNN (Zhu et al., 2022) with MSE the loss function, we get a 2.38% increase over it and outperform the comparable result. Finally, we test our algorithm on the N-MNIST dataset. As shown in Table 3, most comparison works get over 99% accuracy. We use the same architecture as PLIF. Our STCA-SNN reaches the best accuracy of 99.67%.

4.3. Ablation study

4.3.1. Ablation study

We performed ablation experiments based on the PLIF neuron model to evaluate the effectiveness of the STCA module. For each dataset, we trained three types of SNNs: STCA-SNNs, TA-SNNs with temporal-wise attention module (Yao et al., 2023c), and vanilla SNNs (PLIF-SNN) without any attention module. The SE attention employed by TA-SNNs in the temporal dimension and the Self-attention employed in this work are both non-local operators, thus, we compared the performance of these two classic non-local operators under the same experiment conditions. We followed the learning process described in section 4.1 for all ablation experiments, and the attention locations were identical for both TA-SNNs and STCA-SNNs. Table 4 shows that all STCA-SNNs outperformed vanilla SNNs on three event stream classification datasets, suggesting that the benefits of the STCA module are not limited to a specific dataset or architecture. Furthermore, Figure 3 illustrates the accuracy performance trend of vanilla SNN, TA-SNN, and our proposed STCA-SNN over 1,000 epochs on the N-Caltech101 dataset. As the training epoch increased, our proposed STCA-SNN demonstrated comparable performance with TA-SNN. This indicates that our STCA module can enhance the representation ability of SNNs.

4.3.2. Discuss of pooling operations

To investigate the influence of the avg-pooling and max-pooling operation, we conducted several ablation studies. As is well known, avg-pooling can capture the degree information of target objects,

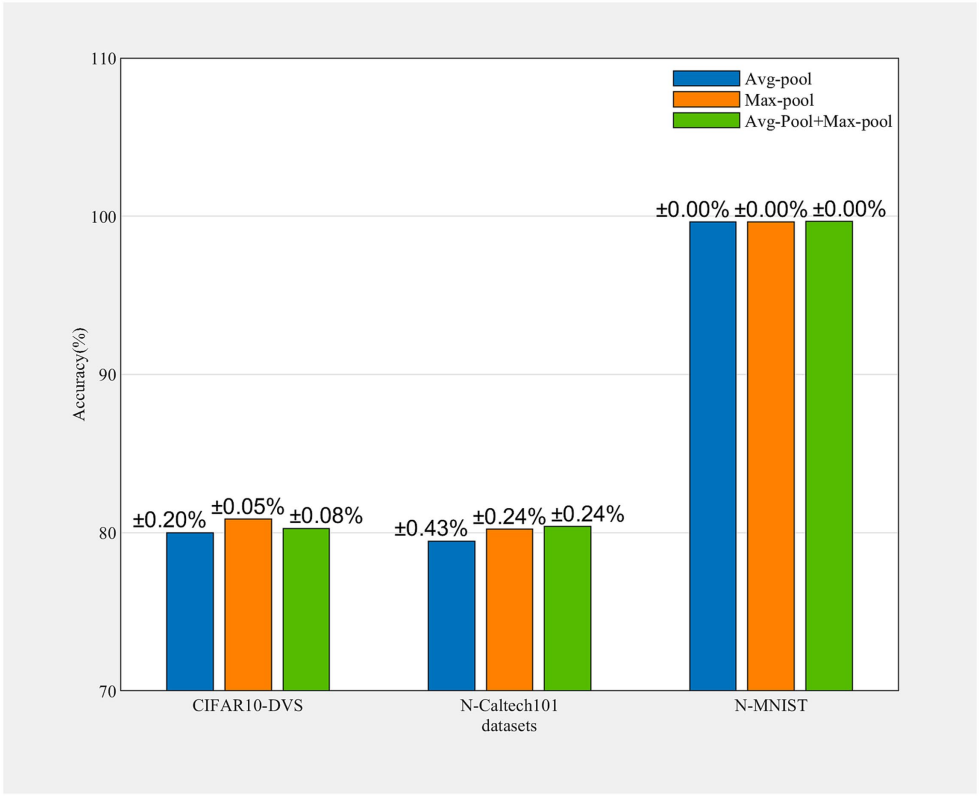


FIGURE 4
Accuracy of different datasets obtained by avg-pooling, max-pooling, and a combination of both. Each experiment is run 3 times.

while max-pooling can extract discriminative features of objects. As shown in Figure 4, the max-pooling operation contributes significantly to performance enhancement. Each experiment is run 3 times. Notably, the fusion of both pooling operations exhibits improved performance across all datasets examined, which means avg-pooling encoded global information can effectively compensate for the discriminative information encoded by max-pooling.

5. Conclusion

In this work, we propose the STCA-SNNs to enhance the temporal information processing capabilities of SNNs. The STCA module captures temporal dependencies across channels globally using self-attention, enabling the network to learn 'when' to attend to 'what'. We verified the performance of STCA-SNNs on various neuromorphic datasets across different architectures. The experimental results show that STCA-SNNs achieve competitive accuracy on N-MNIST, CIFAR10-DVS, and N-Caltech 101 datasets.

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 authors.

Author contributions

XW: Conceptualization, Investigation, Methodology, Software, Visualization, Writing – original draft. YS: Funding acquisition, Supervision, Writing – review & editing. YZ: Supervision, Writing – review & editing. YJ: Supervision, Writing – review & editing. YB: Formal analysis, Validation, Writing – review & editing. XL: Formal analysis, Software, Validation, Visualization, Writing – review & editing. XY: Writing – review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was supported by the National Natural Science Foundation of China General Program (No. 82272130) and the National Natural Science Foundation of China Key Program (No. U22A20103).

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

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. Proceedings of the IEEE conference on computer vision and pattern recognition.

Ba, J., Mnih, V., and Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. In ICLR.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. 32nd conference on neural information processing systems.

Bu, T., Ding, J., Yu, Z., and Huang, T. (2022). Optimized potential initialization for low-latency spiking neural networks, the thirty-sixth AAAI conference on artificial intelligence (AAAI).

Cai, W., Sun, H., Liu, R., Cui, Y., Wang, J., Xia, Y., et al. (2023). A spatial-channel-temporal-fused attention for spiking neural networks. IEEE transactions on Neural Networks and Learning Systems. arXiv:2209.10837.

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Cheng, W., Luo, H., Yang, W., Yu, L., Chen, S., and Li, W. (2019). Det: a high-resolution dvs dataset for lane extraction. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pp. 1666–1675.

Diehl, P., 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

Ding, J., Yu, Z., Tian, Y., and Huang, T. (2021). Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks. International joint conference on artificial intelligence.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). "An image is worth 16x16 words: transformers for image recognition at scale" in *International conference on learning representations (ICLR)*

Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., et al. (2020). Spikingjelly. Available at: https://github.com/fangwei123456/spikingjelly.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. (2021). Incorporating learnable membrane time constant to enhance learning of spiking neural networks. Proceedings of the IEEE/CVF international conference on computer vision, 2661–2671.

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2020). Event-based vision: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 44:1. doi: 10.1109/TPAMI.2020.3008413

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal dynamics: From single neurons to networks and models of cognition*, Cambridge University Press, Cambridge, MA.

Guo, M. H., Xu, T. X., Liu, J. J., Liu, Z. N., Jiang, P. T., Mu, T. J., et al. (2022). Attention mechanisms in computer vision: a survey. *Comput. Visual Media* 8, 331–368. doi: 10.1007/s41095-022-0271-y

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., et al. (2022). A survey on vision transformer. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 87–110. doi: 10.1109/TPAMI.2022.3152247

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. Proceedings of the IEEE conference on computer vision and pattern recognition.

Hu, Y., Tang, H., and Pan, G. (2021). Spiking deep residual networks. *IEEETrans. Neural Netw. Learn. Syst.* 34, 5200–5205. doi: 10.1109/TNNLS.2021.3119238

Huang, Z., Zhang, S., Pan, L., Qing, Z., Tang, M., Liu, Z., et al. (2022). *TAda! oman. In ICLR.*

Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 1254–1259. doi: 10.1109/34.730558

Kheradpisheh, S., Mohammad, G., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kim, Y., and Panda, P. (2021). Optimizing deeper spiking neural networks for dynamic vision sensing. *Neural Netw.* 144, 686–698. doi: 10.1016/j.neunet.2021.09.022

Kingma, D. P., and Ba, J. L. (2015). Adam: a method for stochastic optimization. ICLR 2015: International conference on learning representations.

Kugele, A., Pfeil, T., Pfeiffer, M., and Chicca, E. (2020). Efficient processing of spatio-temporal data streams with spiking neural networks. *Front. Neurosci.* 14:439. doi: 10.3389/fnins.2020.00439

Li, G., Fang, Q., Zha, L., Gao, X., and Zheng, N. (2022). HAM: hybrid attention module in deep convolutional neural networks for image classification. *Pattern Recogn.* 129:108785. doi: 10.1016/j.patcog.2022.108785

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci.* 11:309. doi: 10.3389/fnins.2017.00309

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

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., et al. (2021). Swin transformer: hierarchical vision transformer using shifted windows. Proceedings of the IEEE/CVF International Conference on Computer Vision.

Mainen, Z. F., and Sejnowski, T. (1995). J, reliability of spike timing in neocortical neurons. *Science* 268, 1503–1506. doi: 10.1126/science.7770778

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

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

Ponulak, F., and Kasinski, A. (2010). Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Posch, C., Matolin, D., and Wohlgenannt, R. (2010). A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE J. Solid State Circuits* 46, 259–275. doi: 10.1109/JSSC.2010.2085952

Rathi, N., and Roy, K. (2021). DIET-SNN: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Trans. Neural Networks Learn. Syst.* 34, 3174–3182. doi: 10.1109/TNNLS.2021.3111897

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. International conference on learning representations.

Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. (2019). High speed and high dynamic range video with an event camera. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 1964–1980. doi: 10.48550/arXiv.1906.07165

Ridwan, I., and Cheng, H., *An event-based optical flow algorithm for dynamic vision sensors* (2017) University of Lethbridge Lethbridge

Rieke, F., Warland, D., Van Steveninck, R. D. R., and Bialek, W. (1999). *Spikes: Exploring the neural code*. MIT Press, Cambridge, MA.

Roy, K., Jaiswal, A., and Panda, A. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*. 575, 607–617. doi: 10.1038/s41586-019-1677-2

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

Samadzadeh, A., Far, F. S. T., Javadi, A., Nickabadi, A., and Chehreghani, M. H. (2023). Convolutional spiking neural networks for spatio-temporal feature extraction. *Neural Processing Letters*. 1–7.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., et al. (2017). Attention is all you need. *Adv. Neural Inf. Proces. Syst.* 30, 5998–6008. doi: 10.48550/arXiv.1706.03762

Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition.

Woo, S., Park, J., Lee, J., and Kweon, I. (2018). Cbam: convolutional block attention module. Proceedings of the European conference on computer vision (ECCV).

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better, in Association for the Advancement of artificial intelligence (AAAI).

Wu, J., Xu, C., Han, X., Zhou, D., Zhang, M., Li, H., et al. (2021). Progressive tandem learning for pattern recognition with deep spiking neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 7824–7840. doi: 10.1109/TPAMI.2021.3114196

Wu, X., Zhao, Y., Song, Y., Jiang, Y., Bai, Y., Li, X., et al. (2023). Dynamic threshold integrate and fire neuron model for low latency spiking neural networks. *Neurocomputing* 544:126247. doi: 10.1016/j.neucom.2023.126247

Xu, Q., Qi, Y., Yu, H., Shen, J., Tang, H., Pan, G., et al. (2018). Csnn: an augmented spiking based framework with perceptron-inception. International Joint Conference on Artificial Intelligence (Stockholm).

Yang, Z., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. *Neural Evol. Comput.* 35, 11062–11070. doi: 10.1609/aaai.v35i12.17320

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., et al. (2021). Temporal-wise attention spiking neural networks for event streams classification. Proceedings of the IEEE/CVF international conference on computer vision (ICCV).

Yao, M., Hu, J., Zhao, G., Wang, Y., Zhang, Z., Xu, B., et al. (2023a). Inherent redundancy in spiking neural networks. Proceeding of the IEEE/CVF international conference on computer vision (ICCV). arXiv preprint arXiv:2308.08227.

Yao, M., Hu, J., Zhou, Z., Yuan, L., Tian, Y., Xu, B., et al. (2023b). Spike-driven Transformer. Advances in Neural Information Processing Systems (NeurIPS). arXiv preprint arXiv:2307.01694.

Yao, M., Zhao, R., Zhang, H., Hu, Y., Deng, L., Tian, Y., et al (2023c). Attention spiking neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 9393–9410. doi: 10.1109/TPAMI.2023.3241201

Yu, C., Gu, Z., Li, D., Wang, G., Wang, A., and Li, E. (2022). STSC-SNN: Spatio-temporal synaptic connection with temporal convolution and attention for spiking neural networks. *Front. Neurosci.* 16:1079357. doi: 10.3389/fnins.2022.1079357

Zenke, F., and Vogels, T. P. (2021). The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural Comput.* 33, 899–925. doi: 10.1162/neco_a_01367

Zhang, M., Luo, X., Chen, Y., Wu, J., Belatreche, A., Pan, Z., et al. (2020). An efficient threshold-driven aggregate-label learning algorithm for multimodal information processing. *IEEE J. Sel. Top Signal Process* 14, 592–602. doi: 10.1109/JSTSP.2020.2983547

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2021). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Trans. Neural Netw. Learn Syst.* 33, 1947–1958. doi: 10.1109/TNNLS.2021.3110991

Zhou, C., Yu, L., Zhou, Z., Ma, Z., Zhang, H., Zhou, H., et al. (2023). Spikingformer: Spike-driven residual learning for transformer-based spiking neural network. arXiv preprint arXiv:2304.

Zhou, Z., Zhu, Y., He, C., Wang, Y., Yan, S., Tian, Y., et al. (2023). Spikformer: When spiking neural network meets transformer. ICLR, 2023. arXiv preprint arXiv:2209.15425.

Zhu, R., Zhao, Q., Zhang, T., Deng, H., Duan, Y., Zhang, M., et al. (2022). TCJA-SNN: Temporal-Channel joint attention for spiking neural networks. arXiv:2206.10177.

frontiers | Frontiers in Computational Neuroscience

TYPE Original Research
PUBLISHED 21 November 2023
DOI 10.3389/fncom.2023.1250908

OPEN ACCESS

EDITED BY
Huajin Tang,
Zhejiang University, China

REVIEWED BY
Jean-Philippe Thivierge,
University of Ottawa, Canada
Sadique Sheik,
SynSense, Switzerland

*CORRESPONDENCE
Bernard Girau
✉ bernard.girau@loria.fr

RECEIVED 30 June 2023
ACCEPTED 27 October 2023
PUBLISHED 21 November 2023

CITATION
Fois A and Girau B (2023) Enhanced representation learning with temporal coding in sparsely spiking neural networks.
Front. Comput. Neurosci. 17:1250908.
doi: 10.3389/fncom.2023.1250908

Enhanced representation learning with temporal coding in sparsely spiking neural networks

Adrien Fois and Bernard Girau*

Université de Lorraine, Centre National de la Recherche Scientifique, Laboratoire lorrain de Recherche en Informatique et ses Applications, Nancy, France

Current representation learning methods in Spiking Neural Networks (SNNs) rely on rate-based encoding, resulting in high spike counts, increased energy consumption, and slower information transmission. In contrast, our proposed method, Weight-Temporally Coded Representation Learning (W-TCRL), utilizes temporally coded inputs, leading to lower spike counts and improved efficiency. To address the challenge of extracting representations from a temporal code with low reconstruction error, we introduce a novel Spike-Timing-Dependent Plasticity (STDP) rule. This rule enables stable learning of relative latencies within the synaptic weight distribution and is locally implemented in space and time, making it compatible with neuromorphic processors. We evaluate the performance of W-TCRL on the MNIST and natural image datasets for image reconstruction tasks. Our results demonstrate relative improvements of 53% for MNIST and 75% for natural images in terms of reconstruction error compared to the SNN state of the art. Additionally, our method achieves significantly higher sparsity, up to 900 times greater, when compared to related work. These findings emphasize the efficacy of W-TCRL in leveraging temporal coding for enhanced representation learning in Spiking Neural Networks.

KEYWORDS

spiking neural networks, temporal code, spike-timing-dependent plasticity, representation learning, visual representations, latency-coding, sparsity, unsupervised learning

1 Introduction

Spiking Neural Networks (SNNs) have been gaining recognition due to their application in supervised (Kheradpisheh and Masquelier, 2020; Lee et al., 2020), reinforcement (Mozafari et al., 2018; Patel et al., 2019), and unsupervised learning tasks (Diehl and Cook, 2015; Kheradpisheh et al., 2018). Compared to Artificial Neural Networks (ANNs) that output continuous values synchronously, SNNs transmit binary outputs sparsely and asynchronously as spikes. This event-based information transmission scheme reduces communication channels, significantly lowers energy requirements, and offers potential energy efficiency gains of up to 1,000 times with neuromorphic processors (Furber, 2016) compared to traditional processors. The massively parallel architecture of neuromorphic processors, where memory (synapses) and computational units (neurons) are co-located, further contributes to this energy efficiency. By utilizing local event-based learning rules, such as Spike-timing-dependent plasticity (STDP), which leverage information from presynaptic and postsynaptic terminals, the principles of parallelism and locality can be fully exploited at the algorithmic level.

Extracting representations from event-based and asynchronous data streams using local rules compatible with neuromorphic processors holds great promise. In this context, King et al. (2013) proposed a spatially local rule learning receptive fields resembling Gabor filters.

Burbank (2015) developed a method that reproduces the behavior of an autoencoder with an STDP rule. This STDP rule learns encoding and decoding weights with a Hebbian and anti-Hebbian rule, respectively. The combination of these two rules allows the approximation of the cost function of an autoencoder. More recently, Tavanaei et al. (2018) proposed an STDP rule, integrating a vector quantization module and a regularization module and provides the best performance in terms of reconstruction error.

However, all these methods operate with spike rate encoding, where information is encoded in the number of emitted spikes. Spike rate encoding incurs high energy costs, and leads to slow information transmission. In contrast, temporal codes, which transmit information based on spike times rather than spike rates, offer a more economical alternative. Temporal encoding using relative latencies achieves significantly higher energy efficiency, reduces the number of spikes required, and enables faster data transmission rates compared to rate coding. This makes temporal codes particularly relevant for neuromorphic computing and learning (Guo et al., 2021).

In this paper, we introduce a two-layer SNN that leverages temporal codes, specifically population-based latency coding, where each neuron fires a single spike. Our Weight-Temporally Coded Representation Learning (W-TCRL) model improves the reconstruction performance while increasing the sparsity of the activity in the network. We propose a novel STDP rule that adjusts synaptic weights based on spike times, operating locally in both space and time, thus facilitating a future implementation on neuromorphic processors (Davies et al., 2018). To the best of our knowledge, this is the first presented method for learning representations from a temporal encoding of inputs using an STDP-like rule aimed at achieving low reconstruction error. We evaluate our spiking model using the MNIST dataset and a natural image dataset.[1] Furthermore, we propose a generic parameterization of the model to address the common adaptability issue with data of varying dimensions.

2 Materials and methods

In this section, the various architectural and algorithmic components of the neural model are illustrated, which will be utilized for representation learning. The model is assessed on an image reconstruction task, with a focus on its new STDP rule that targets synaptic weights.

2.1 Spiking neural network

As depicted in Figure 1, our SNN architecture consists of two fully interconnected layers: the first layer encodes input data into relative spike latencies, generating a single spike per neuron, while the second layer extracts representations from the resulting spiking activity.

1 The code is released: https://github.com/afois/W-TCRL.

2.1.1 Neuron and synapse model

The network contains only LIF neurons with current-based (CuBa) synapses. This widely-used model captures the basic behavior of a biological neuron while maintaining low computational cost and ease of analysis. The dynamic of a LIF neuron is given by:

$$\tau_m \frac{dV(t)}{dt} = -V(t) + I(t) + g\eta(t)$$

Neurons face a Gaussian white noise process η, scaled by g; g = 0 (no noise) for most tests. However, we test the robustness of the network on MNIST dataset in Section 3.3 by varying g. We add the output $s(t)$ and the resetting of the membrane potential $V(t)$ when the firing threshold V_θ is reached, $V(t)$ is then maintained at 0 during a refractory period T_{refrac}:

$$\text{if } V(t) < V_\theta, \quad \text{then } s(t) = 0$$

$$\text{if } V(t) \geq V_\theta, \quad \text{then } \begin{cases} s(t) = 1 \\ V(u) = 0 \; \forall u \in]t, t + T_{\text{refrac}}] \end{cases}$$

Between the two layers, synaptic transmission is modeled by an exponential synapse model. If at least one presynaptic neuron i fires, the indicator function $s_i(t)$ takes the value 1 (0 otherwise) and the input to the postsynaptic neuron $I_j(t)$ changes instantaneously by an amount equal to the synaptic weight w_{ji}:

$$I_j(t) \leftarrow I_j(t) + \sum_{i=1}^{n} w_{ji} s_i(t)$$

otherwise $I_j(t)$ exponentially decays over time with a time constant τ_f:

$$\tau_f \frac{dI_j(t)}{dt} = -I_j(t)$$

2.1.2 Synaptic traces

Synapses connect the n presynaptic neurons of the encoding layer to the m postsynaptic neurons of the representation layer. Each synapse has access to a local state variable $x_i(t)$ (with $i = 1, 2, \ldots n$) that tracks recent presynaptic spikes. This variable is known as presynaptic trace. It is commonly used for efficient and local implementations of STDP rules, both in simulations (Pfister and Gerstner, 2006) and neuromorphic processors (Davies et al., 2018).

Similarly, a postsynaptic trace $y_j(t)$ (with $j = 1, 2, \ldots m$) accessible to the synapses tracks recent postsynaptic spikes. When a pre (post)-synaptic spike occurs, $x_i(t)$ ($y_j(t)$) jumps to 1 and then decays exponentially to 0 with a time constant τ_x (τ_y):

$$\begin{aligned} x_i(t) &\leftarrow 1, && \text{if } s_i(t) = 1 \\ \tau_x \frac{dx_i(t)}{dt} &= -x_i(t), && \text{otherwise} \end{aligned} \tag{1}$$

$$\begin{aligned} y_j(t) &\leftarrow 1, && \text{if } s_j(t) = 1 \\ \tau_y \frac{dy_j(t)}{dt} &= -y_j(t), && \text{otherwise} \end{aligned} \tag{2}$$

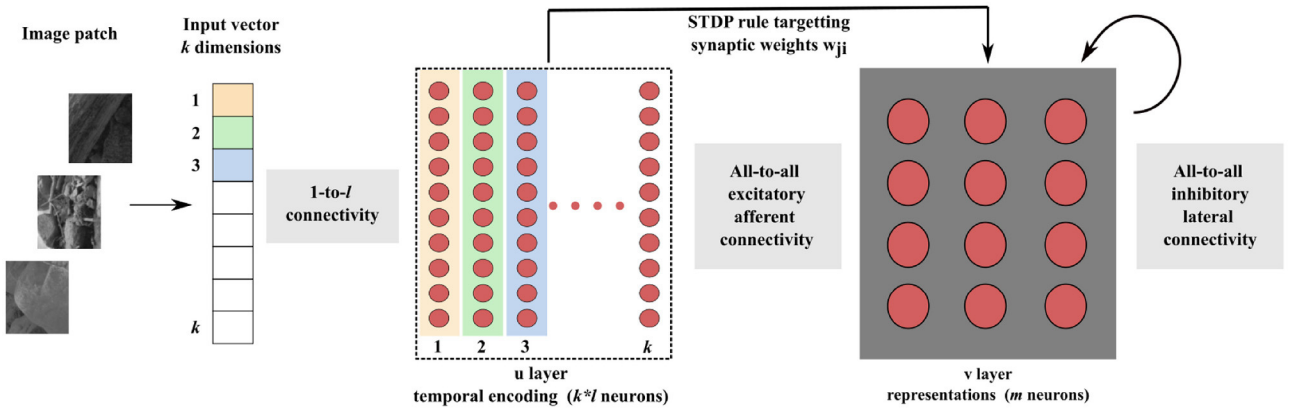


FIGURE 1
Block diagram of the SNN architecture. Consider an input vector of k dimensions. Each dimension of the input vector is encoded by l neurons, each of them firing a single spike. The sparse activity of the $k * l$ neurons is transmitted to the representation layer. Learning of the synaptic weights w_{ji} takes place between these two layers, with our new STDP rule. Inhibitory lateral connections are present in the representation layer to put neurons into competition, thereby generating competitive learning.

where $s(t)$ is an indicator function returning 1 when a neuron fires a spike at time t, 0 otherwise.

2.2 Encoding input data in relative spike latencies

Let $\mathbf{a} \in [0,1]^k$ denote a normalized k-dimensional real-valued input vector that we aim to encode into relative spike latencies (or timing) through population coding (Ebitz and Hayden, 2021). This can be achieved by distributing each dimension of the input vector over the population activity of l neurons (Figure 2). We used a population of $l = 10$ neurons to collectively represent one dimension. Each neuron $i \in \{1,2,...,l\}$ of the population has an associated gaussian receptive field in a circular space in range $[0,1]$, characterized by its preferential value (or center) μ_i and by its width (or standard deviation) σ. The centers μ_i are uniformly spread between 0.05 and 0.95. The standard deviation $\sigma = 0.6$ is the same for all neurons. This broad tuning ensures that any neuron receives high enough activation levels to fire in response to any input value. The resulting receptive fields overlap, covering the entire input space.

A gaussian function $G_{\mu,\sigma} : \mathbf{a} \rightarrow \mathbf{A}$ is then used to transform the input vector $\mathbf{a} \in [0,1]^k$ into a vector of activations levels $\mathbf{A} \in [0,1]^{k \times l}$ that feed the $n = k \times l$ LIF neurons of the encoding layer.

To illustrate the encoding process, consider a single entry of the input vector $a \in \mathbf{a}$. The ith neuron—with $i \in \{1,2,...,l\}$—whose gaussian receptive field center μ_i is the closest to the input value a, gets the highest activation level A_i and will thus fire first. The other neurons fire later. The higher the distance between their centers and the input value, the lower the activation levels they get, and therefore the later they fire. The temporal separation of spikes is accentuated by the non-linear integration of the received activation levels by LIF neurons. With this mechanism, a specific input value is encoded in a specific spike pattern in the spatio-temporal domain. This population-based coding does not lie within the times to first spikes, but explicitly within the relative latencies of the spikes emitted by each neuron.

Note that in the scenario of continuous input presentation, this circuit will disrupt the temporal representation of the analog inputs. Inputs need to be somewhat reset to achieve spike time reproducibility. However, it is straightforward to replace this circuit with others such as the one proposed by Rumbell et al. (2014). This method enables continuous input presentation to the network by producing oscillatory firing through an inhibitory mechanism. Thus, it generates an internal time reference through oscillations. Both encoding methods utilize a population of Gaussian receptive fields and yield identical spiking patterns. Rumbell's method offers greater flexibility at the cost of a more complex encoding circuit.

2.3 W-TCRL: STDP rule for learning representations in weights

Non-weight-dependent Spike-Timing-Dependent Plasticity (NSTDP) rules can produce stable receptive fields; however, they are bistable, leading to synaptic weights saturating at minimum and maximum values (Song et al., 2000). Hence, the potential range of learning parameters, the synaptic weights in this case, is not fully utilized under NSTDP rules. This limitation significantly impedes the expressive capacity of synaptic weights, diminishing their capability to finely represent external states, such as environmental states.

On the other hand, Weight-dependent Spike-Timing-Dependent Plasticity (WSTDP) rules rectify this issue by generating unimodal and continuous distributions of synaptic weights, thereby leveraging the entire range of weight variation. However, these rules fall short in generating stable receptive fields (Billings and van Rossum, 2009).

Our STDP rule provides the dual advantages of stability and the complete utilization of the synaptic weight range for representation storage. The interplay between our STDP rule and the unimodal temporal code results in a stable and unimodal

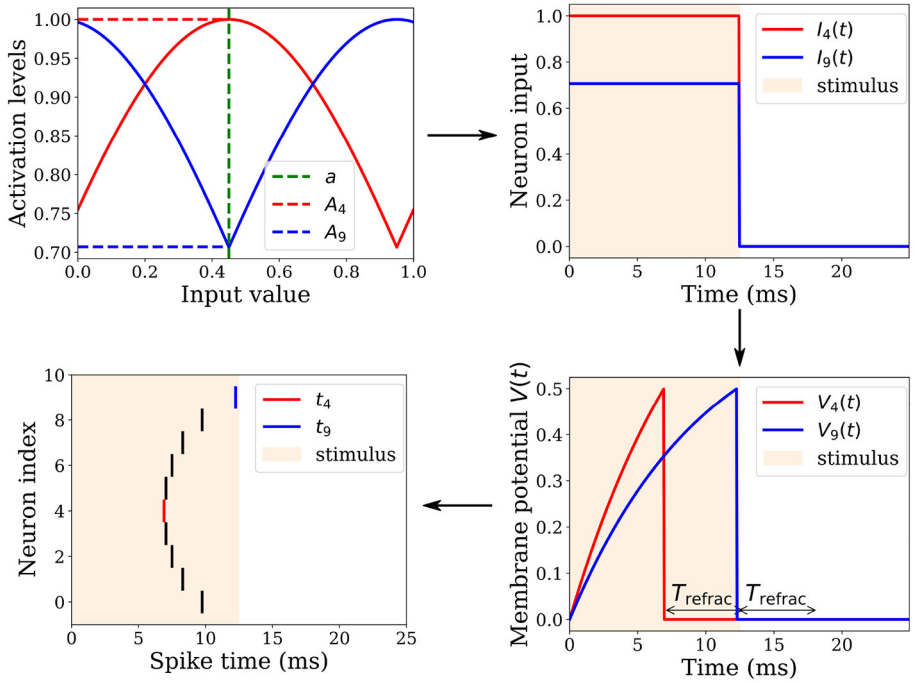


FIGURE 2
Spatio-temporal spike pattern encoding an input value of $a = 0.45$. **Top left**: The input is received by a population of $l = 10$ neurons, each having a Gaussian tuning curve centered on a preferential value μ_i. For instance, Neuron 4 and 9 possess $\mu_4 = 0.45$ and $\mu_9 = 0.95$, respectively. Their tuning curves and corresponding activation levels A_4 and A_9 for the input value are depicted in red and blue, respectively. **Top right**: These activation levels remain constant over a 12.5 ms interval, followed by an equal duration of quiescence. **Bottom right**: The neuronal inputs are integrated into membrane potentials V_4 and V_9. With a higher activation level, V_4 grows faster than V_9, enabling it to reach its firing threshold V_θ sooner. Upon crossing this threshold, the neuron fires a spike, resets its membrane potential, and enters a refractory period T_{refrac}. **Bottom left**: A neuron fires a spike when it crosses its threshold V_θ, thereby distributing an input value across a neuronal population to produce a specific spatio-temporal spike pattern.

distribution of synaptic weights (see Figure 4). Here, the magnitude of a synaptic weight is function of the temporal distance between a presynaptic and a postsynaptic spike. In the encoding layer u, faster firing neurons carry a more accurate representation of the input, correspondingly leading to a higher value of the synaptic weight. This, in turn, boosts the neuron's causal influence over the emission of a postsynaptic spike.

Our novel STDP rule, derived from a vector quantization criterion, operates both spatially and temporally. Spatially, it focuses on a single pair of presynaptic and postsynaptic neurons i and j, and temporally, it operates within a defined temporal window.

The change in synaptic weight Δw_{ji} is calculated based on the presynaptic trace $x_i(t) \in]0, 1]$ and the postsynaptic trace $y_j(t) \in]0, 1]$. The synaptic weight $w_{ji} \in [0, 1]$ is then updated by this change $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$. Our new STDP rule is defined as follows (see Figure 3):

$$\Delta w_{ji} = \begin{cases} +\alpha^+\big(1 - x_i(t) - w_{ji} + w_{\text{offset}}\big), & \text{if } s_j(t) = 1 \text{ and } x_i(t) > \epsilon \\ -\alpha^-\big(1 - y_j(t)\big), & \text{if } s_i(t) = 1 \text{ and } y_j(t) > \epsilon \end{cases} \tag{3}$$

where α^+ and α^- are the learning rates. This STDP rule works in combination with the introduction of hard bounds that restrict synaptic weights to the range $0 \leq w_{ji} \leq 1$:

$$w_{ji} \leftarrow \min\big(1, \max(0, w_{ji} + \Delta w_{ji})\big)$$

2.3.1 First case of synaptic weight adaptation

The first adaptation case (Figure 3A) is (1) triggered event-wise by a postsynaptic spike indicated by $s_j(t = t_{\text{post}}) = 1$ and (2) operates locally in time since the presynaptic trace $x_i(t)$ is sampled at time $t = t_{\text{post}}$ and must be greater than ϵ, corresponding to a temporal window since the time of the presynaptic spike t_{pre}. Thus, postsynaptic spikes arriving outside this temporal window do not trigger the rule. This rule addresses the case of causal interactions, i.e., presynaptic spikes emitted before or at the time of the postsynaptic spike, i.e., $\Delta_t \geq 0$.

This first adaptation case integrates a vector quantization module aimed at learning the distribution of relative spike times in the distribution of synaptic weights, and thus minimizing the reconstruction error because the neural encoding is temporal. We can observe, by analyzing the equilibrium solution $\Delta_{w_{ji}} = 0$, that the weight w_{ji} converges to a value dependent on $x_i(t)$ and thus on the time interval $\Delta_t = t_{\text{post}} - t_{\text{pre}}$ between pre- and

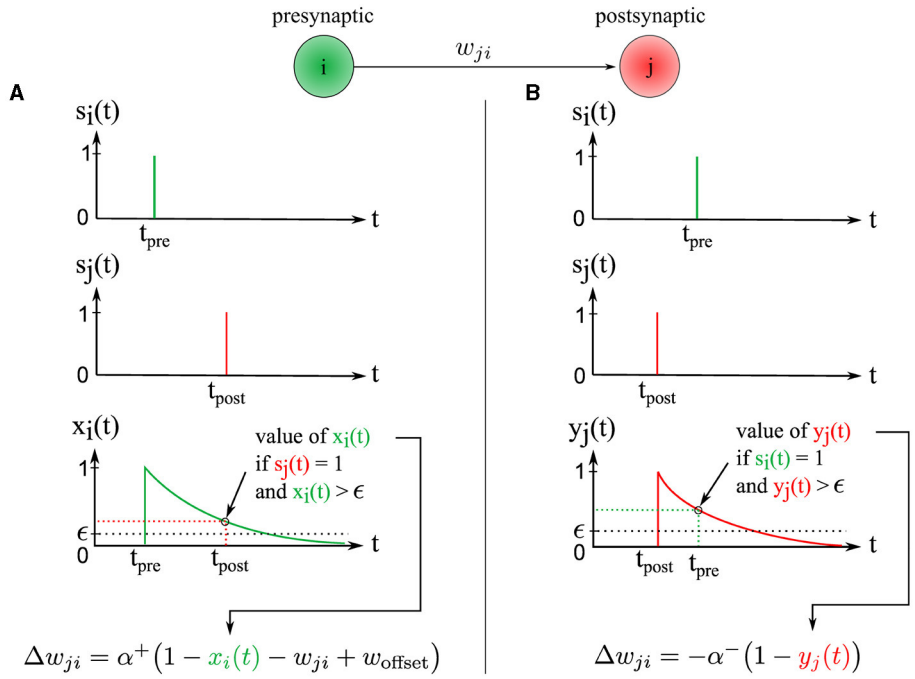


FIGURE 3
Illustration of the two synaptic weight adaptation cases under our novel STDP rule, demonstrating spatial and temporal locality. **(A)** The left column presents the first case, triggered in a causal context where the presynaptic spike precedes the postsynaptic one. **(B)** Conversely, the right column illustrates the second case, triggered in an anti-causal context when the postsynaptic spike precedes the presynaptic one. In both adaptation cases, the adjustment magnitude Δw_{ji} is influenced by the presynaptic and postsynaptic traces $x_i(t)$ and $y_j(t)$, respectively. These traces are equivalent to exponential decay kernels applied to Δ_t, where $\Delta_t = t_{\text{post}} - t_{\text{pre}}$.

postsynaptic spikes:

$$\begin{aligned} &\Delta w_{ji} = 0 \\ \Leftrightarrow\ &0 = 1 - x_i(t) - w_{ji} + w_{\text{offset}} \\ \Leftrightarrow\ &w_{ji} = 1 - x_i(t) + w_{\text{offset}} \\ \Leftrightarrow\ &w_{ji} = 1 - \exp\left(-\frac{t - t_{\text{pre}}}{\tau_x}\right) + w_{\text{offset}} \end{aligned}$$

As this adaptation case is triggered by a postsynaptic spike indicated by $s_j(t = t_{\text{post}}) = 1$, we finally obtain:

$$\Leftrightarrow\ w_{ji} = 1 - \exp\left(-\frac{\Delta_t}{\tau_x}\right) + w_{\text{offset}} \tag{4}$$

Consider a spike pattern repeatedly presented to a postsynaptic neuron for illustration. The synaptic weight w_{ji} will converge to a value that depends on the presynaptic trace $x_i(t)$ and therefore on Δ_t. Recall that the trace $x_i(t)$ jumps to 1 upon the emission of a presynaptic spike, and then decays exponentially over time. Thus, the larger the value of $\Delta_t \to \infty$, the smaller the value of $x_i(t) \to 0$. The first presynaptic spikes carry most of the information about the input, which can be interpreted as a hidden variable. These first spikes are reflected by a large Δ_t and thus a small $x_i(t) \in]0, 1]$. Therefore we introduce the term $1 - x_i(t)$ in the rule, which induces high synaptic weight values w_{ji} for the earliest firing presynaptic neurons, thereby amplifying the causal impact of the first presynaptic spikes on the postsynaptic neuron. Consequently, the first presynaptic neuron that fires induces the highest weight value w_{ji}, while the last firing neuron induces the lowest weight value w_{ji}. This process unfolds for each presynaptic-postsynaptic neuron pair meeting the conditions of the first adaptation case. Thus, the order and latency of the spikes are represented in the resulting synaptic weight distribution w_{ji} (see Figure 4). By contrast, the traditional STDP rule (Song et al., 2000) maximally strengthens the latest spike ($t_{pre} \approx t_{post}$) for causal interactions and typically leads to weight saturation, where all weights saturate to 1.0 for instance.

We still need to address the role of the last term w_{offset}. When $\Delta_t \to 0$, this is reflected by $x_i(t) \to 1$ and induces $1 - x_i(t) \to 0$ and thus $w_{ji} \to 0$. To avoid $w_{ji} \to 0$ when a presynaptic spike is emitted at a time $t_{\text{pre}} \approx t_{\text{post}}$, i.e., when $\Delta_t \to 0$, we add a positive offset $w_{\text{offset}} > 0$ to the rule.

2.3.2 Second case of synaptic weight adaptation

The second adaptation case (Figure 3B) is triggered by a presynaptic spike indicated by $s_i(t = t_{\text{pre}}) = 1$, and operates locally in time because the postsynaptic trace $y_j(t)$ is sampled at time $t = t_{\text{pre}}$ and must be greater than ϵ, corresponding to a temporal window from the postsynaptic spike time t_{post}. Thus, presynaptic spikes falling outside of this temporal window do not trigger the rule. This rule handles the case of anti-causal interactions, i.e., presynaptic spikes emitted after or at the time of the postsynaptic spike, i.e., $\Delta_t \leq 0$.

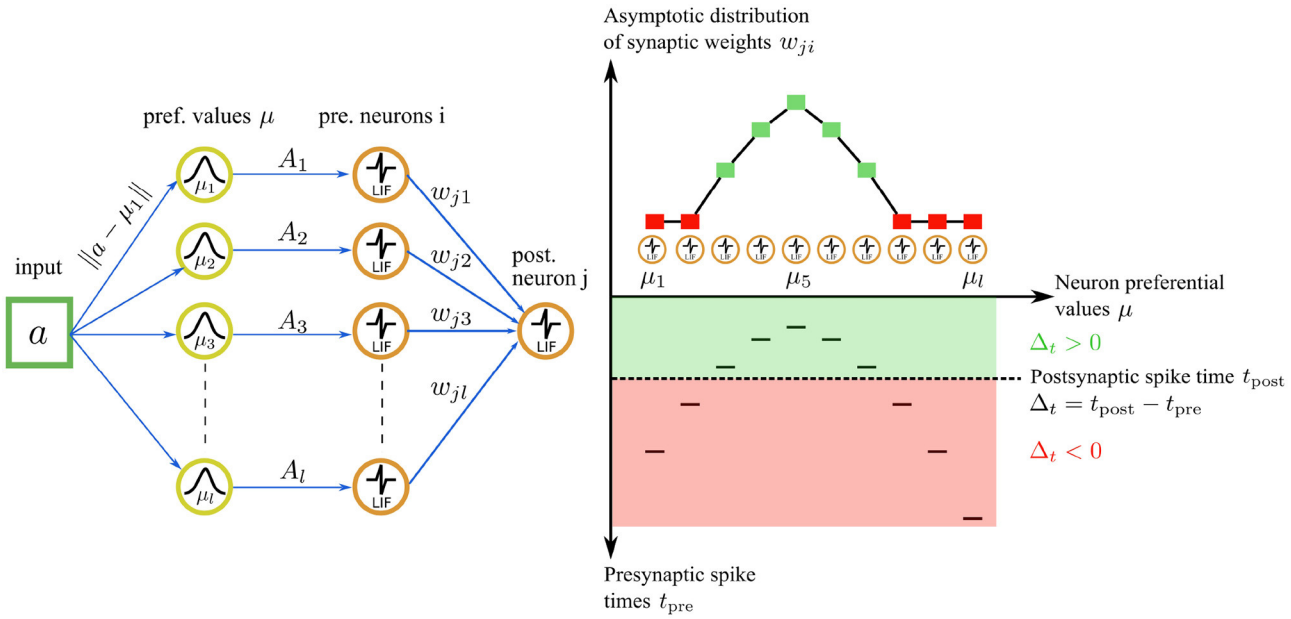


FIGURE 4
Illustration of the asymptotic weight distribution learned with our STDP rule for an input value of $a = 0.45$. **Left**: The input value is encoded by a population of $l = 10$ neurons in the encoding layer u. **Right bottom**: The neuron with the nearest preferential value μ_i to the input ($\mu_5 = 0.45$ in this case) fires a spike first, followed by the other neurons in the population. The spiking pattern is transmitted through synaptic connections to a postsynaptic neuron j in the representation layer. The postsynaptic neuron j integrates the received inputs and fires a spike at time t_{post} (dashed line). The sign of the time difference Δ_t between a postsynaptic and a presynaptic spike delimits two learning regimes (green and red areas) for the STDP rule. **Right top**: for $\Delta_t \geq 0$, the first adaptation case is triggered, resulting in an attractive fixed point weight. The asymptotic weight value w_{ji} increases as Δ_t increases (green squares). For $\Delta_t \leq 0$, the second adaptation case is triggered, leading to weight depression. The weight value w_{ji} decreases as Δ_t increases, tending asymptotically toward 0 (red squares).

We can reformulate this adaptation case to explicitly show the dependence on $\Delta_t = t_{\text{post}} - t_{\text{pre}}$:

$$\Delta w_{ji} \propto 1 - y_j(t)$$
$$\Delta w_{ji} \propto 1 - \exp\left(-\frac{t - t_{\text{post}}}{\tau_y}\right)$$

Since this adaptation case is triggered by a presynaptic spike indicated by $s_i(t = t_{\text{pre}}) = 1$, we have:

$$\Delta w_{ji} \propto 1 - \exp\left(-\frac{t_{\text{pre}} - t_{\text{post}}}{\tau_y}\right)$$
$$\Delta w_{ji} \propto 1 - \exp\left(+\frac{\Delta_t}{\tau_y}\right)$$

Δw_{ji} is proportional to $1 - \exp(\frac{\Delta_t}{\tau_y})$, which is multiplied by $-\alpha^-$. Therefore, this second adaptation case depresses the synaptic weights. The postsynaptic trace $y_j(t)$ behaves similarly to the presynaptic trace $x_i(t)$, where it jumps to 1 upon emission of a postsynaptic spike and then decays exponentially over time. Thus, the larger the value of $\Delta_t \to \infty$, the smaller the value of $y_j(t) \to 0$. Late-arriving presynaptic spikes convey minimal information about the input. That is why we introduce $\Delta w_{ji} \propto 1 - y_j(t)$ into the rule: the later a presynaptic spike arrives relative to a postsynaptic spike, the greater the depression of w_{ji}. Thus, considering the asymptotic behavior of this synaptic depression case, the weight $w_{ij} \to 0$ (see Figure 4), as the lower bound of the weight is fixed at 0 using a hard bound. This asymptotical behavior mirrors the traditional STDP rule (Song et al., 2000) for anti-causal interactions. However, the learning dynamics are reversed : the latest spike ($t_{pre} \gg t_{post}$) induces maximal weight depression, while close anti-causal interactions ($t_{pre} \approx t_{post}$) result in minimal weight depression.

2.3.3 Winner-Take-All circuit

Through afferent synaptic weight adjustments, postsynaptic neurons progressively adapt to distinct spatiotemporal patterns (see Figure 4). Additionally, akin to biological neurons, they emit spikes before receiving all presynaptic inputs, thus boosting the SNN's processing speed.

Remember that the first spikes reduce uncertainty about the input the most, which can be interpreted as a hidden variable. This feature is harnessed by a temporal Winner-Take-All circuit. In this circuit, the first neuron crossing its firing threshold $V_j(t) > V_\theta$ in response to a spatiotemporal input pattern is identified as the best pattern representative, termed the Spiking Best Matching Unit (SBMU). The SBMU strongly inhibits the other neurons in the representation layer, preventing them from firing and thus learning the current prototype.

This self-organizing competitive learning scheme is implemented by all-to-all lateral inhibitory connections. The circuit induces highly sparse activity in the representation layer and forces neurons to learn uncorrelated code vectors, two

important ingredients for efficient learning (Bengio et al., 2013; Falez et al., 2019).

However, for a given input vector, a pure WTA circuit only updates the code vector of a single neuron: the SBMU. Our aim is to speed up learning by recruiting more than one neuron (k-WTA) during the presentation of an input vector.

To achieve this goal, we introduce a homeostatic mechanism that increases the value of the lateral synaptic weights in the representation layer. Initially, lateral inhibition is minimal with lateral weights initialized at $w_{ji} = -c_{min}$. As learning progresses, the homeostatic mechanism raises these weights toward the target value of $w_{ji} \approx -c_{max}$. With appropriate values of $c_{\min}$ and $c_{\max}$, the circuit can dynamically transition from a k-WTA to a WTA behavior.

The homeostatic mechanism progressively involves fewer neurons during the learning phase, enabling gradual decorrelation and specialization of the code vectors that neurons learn. The mechanism is described by the following equation:

$$\tau_w \frac{dw_{ji}}{dt} = -c_{\max} - w_{ji} \tag{5}$$

where τ_w is the time constant of the homeostatic mechanism. We set τ_w to one-third of the total learning phase duration to ensure $w_{ji} \approx -c_{max}$ is reached by the end of the learning phase. Given an encoding time window of T and P input vectors, the time constant is then given by $\tau_w = 1/3 \times T \times P$.

3 Experimental protocol and results

3.1 Metrics for performance evaluation

We report experiments of representation learning with visual data in this section. A first natural objective is to evaluate the quality of visual representations learned by neurons in the representation layer v. For this, we use the Root Mean Square (RMS) reconstruction error between an input vector and the code vector of the associated SBMU, comparing the original and reconstructed image patches.

The Structural Similarity Index Measure (SSIM) is another measure that gauges the structural similarity between two images, rather than pixel-by-pixel difference as done by the Peak Signal-to-Noise Ratio (PSNR) measure based on RMS error. However, SSIM is not used in works related to ours, making it difficult to compare our performances. Furthermore, studies (Horé and Ziou, 2010; Dosselmann and Yang, 2011) have revealed analytical and statistical links between PSNR (based on RMS) and SSIM, indicating that their differences essentially stem from their sensitivity to image degradation. More generally, there is currently no satisfactory visual quality measure that fully captures human perception. Hence, in addition to the RMS reconstruction error used in our quantitative analysis, we provide a qualitative evaluation that visually examines the learned representations and the resulting reconstructed images.

Related works (Burbank, 2015; Tavanaei et al., 2018) use a correlation coefficient-based similarity measure for evaluating visual representation quality. However, we forgo this measure due to its high sensitivity to outliers (Jenkin et al., 1991; Yen and Johnston, 1996), interpretational challenges (Lee Rodgers and Nicewander, 1988), and technical limitations, such as undefined coefficients for image patches with uniform intensity—a common occurrence in the MNIST database, particularly with pure white patches.

Beyond the quality of learned representations, we also evaluate additional SNN characteristics, such as the sparsity of activity in the representation layer and the Euclidean incoherence in the self-organized election process of the SBMU—a new measure that we introduce subsequently. The RMS reconstruction error, sparsity, and incoherence provide insights into the accuracy, efficiency, and self-organization capability of the SNN.

3.1.1 Mean squared error

We use the Root Mean Squared (RMS) error to quantify the difference between an input image patch $\mathbf{a}_p$ and a reconstructed patch $\hat{a}_p$ (6):

$$RMS = \frac{1}{P}\sum_{p=1}^{P}\sqrt{\frac{1}{k}\sum_{i=1}^{k}(a_{i,p} - \hat{a}_{i,p})^2} \tag{6}$$

Here, k is the number of pixels in a patch and P is the number of patches, and $a_{i,p}$ is the ith pixel of a_p.

Recall that each dimension $i \in 1, 2, ..., k$ of an input patch is distributed among l encoding neurons in the encoding layer. Each of the $z \in 1, 2, ..., l$ encoding neurons has an associated preferential value $\mu_z \in \mu_1, \mu_2, ..., \mu_l$. The encoding layer is fully connected to the representation layer in an all-to-all relationship. Each postsynaptic neuron in the representation layer is thus connected to $k * l$ presynaptic neurons in the encoding layer.

Each dimension $i \in 1, 2, ..., k$ of the reconstructed patch $\hat{a}_{i,p}$—corresponding in our case to the intensity of a pixel—is decoded from the distribution of l synaptic weights $w_1^i, w_2^i, ..., w_l^i$ associated with it. The decoding method is based on the circular mean of neurons' preferred directions, weighted by their weight values, as exposed below (see Figure 5). The underlying principle is that the neural code exhibits unimodality (convexity) due to the presence of circular Gaussian receptive fields. This distribution of spike timings is mirrored in a unimodal synaptic weight distribution in the learned weights, a result of our STDP rule (see Figure 4). Therefore, we employed a circular weighted mean to decode the empirical mean. Interestingly, this decoding method has been successfully employed to decode the direction of arm movement from neuronal populations activity in the primate motor cortex (Georgopoulos et al., 1986).

The first step of the decoding process is to transform each preferential value $\mu_z \in \mu_1, \mu_2, ..., \mu_l$ of the presynaptic neurons into an angle θ_z (preferred direction):

$$\theta_z = 2\pi\mu_z$$

We then calculate the weighted mean of the Cartesian coordinates $\overline{x}_i$ and $\overline{y}_i$ from the angles θ_z (angular coordinates) and

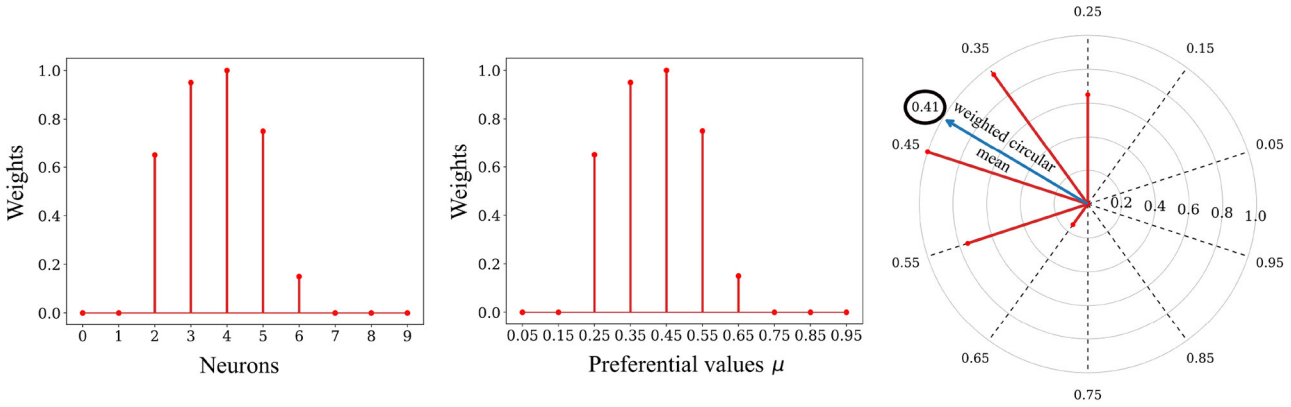


FIGURE 5
Decoding process. **Left**: The relative spike timing relationships of the spike patterns are stored in the weights. **Middle**: We can replace the index of a neuron with its associated preferential value μ. **Right**: The decoding process involves using a circular mean of the encoding neuron's preferential values, weighted by the weights values, to map the stored representation back to the input space. In this example the value 0.41 was decoded from the weights.

the synaptic weights w_z^i associated with pixel i of the SBMU (radial coordinates).

$$\text{with } w_z^i = w_{\text{sbmu},(i-1)l+z}$$

$$\overline{x_i} = \frac{1}{\Delta_i} \sum_{z=1}^{l} w_z^i \cos(\theta_z)$$

$$\overline{y_i} = \frac{1}{\Delta_i} \sum_{z=1}^{l} w_z^i \sin(\theta_z)$$

$$\Delta_i = \sum_{z=1}^{l} w_z^i$$

We calculate a new angle $\overline{\theta_i}$ from the Cartesian coordinates $\overline{x}_i$ and $\overline{y}_i$. This angle $\overline{\theta_i}$ corresponds to the weighted circular mean:

$$\overline{\theta_i} = \text{atan2}\left(-\overline{y_i}, -\overline{x_i}\right) + \pi$$

We end with an inverse transformation, transforming the angle $\overline{\theta_i} \in [0, 2\pi]$ into the normalized intensity of a pixel in the reconstructed patch $\hat{a}_{i,p} \in [0, 1]$:

$$\hat{a}_{i,p} = \frac{\overline{\theta_i}}{2\pi}$$

3.1.2 Sparsity

Sparsity is not evaluated on the encoding layer but on the output layer, i.e., the representation layer. Sparsity corresponds to the percentage of active neurons for an input vector during the encoding time window T:

$$\text{Sparsity} = \frac{1}{m} N_{\text{imp}} \tag{7}$$

where m is the number of neurons, and N_{imp} is the number of spikes emitted by the m neurons in response to an input vector over the time window T. A low value indicates a large number of inactive neurons and therefore a high sparsity.

3.1.3 Coherence in the election of the SBMU

The Best Matching Unit (BMU) in traditional Vector Quantization models like the Self-Organizing Map (Kohonen, 2013), is selected each iteration using global network information. However, in our SNN, the Spiking Best Matching Unit (SBMU) selection is a dynamic and self-organized process influenced by neuron competition to represent the current input.

To assess the coherence of this self-organized SBMU selection compared to a Euclidean distance-based selection, we introduce a measure of incoherence. This measure evaluates if the SBMU is among the best representatives minimizing the Euclidean distance between the reconstructed patch from the SBMU and the input.

Firstly, we define a function d which, for a given input vector, calculates the normalized Euclidean distance between the input vector and the associated code vector of a neuron indexed $j \in 1, \ldots, m$ in the representation layer:

$$d : 1, \ldots, m \rightarrow [0, 1]$$

Then, all distances $d(j)$ are computed, and pairs $\left[j, d(j)\right]$ for $j = 1, \ldots, m$ are sorted in increasing distance order, resulting in an ordered list $\left[j_p, d(j_p)\right]$, where $p = 1, \ldots, m$, and j_p denotes a pair index in the list:

$$\forall p < m \quad d(j_{p+1}) \geq d(j_p) \quad \text{and} \quad \{j_p | p = 1, \cdots, m\} = \{1, \cdots, m\}$$

Next, we assess if the index of the selected SBMU lies within the top x% of the ordered list. If yes, we increment the coherent SBMU count (sc), otherwise the paradoxical SBMU count (sp) is incremented. This check is repeated for each input vector to classify the elected SBMU as coherent or paradoxical. The top x% of the list sets the tolerance threshold for distinguishing between a coherent and paradoxical SBMU.

$$\text{if sbmu} \in j_p \mid p \in 1, \ldots, \lceil mx \rceil, \quad \text{then } sc \leftarrow sc + 1$$
$$\text{else } sp \leftarrow sp + 1$$

TABLE 1 W-TCRL parameters used in all simulations.

Neuronal parameters						
dt	τ_m^u	τ_m^v	V_θ^u	V_θ^v	T_{refrac}^u	T_{refrac}^v
0.1 ms	10.0 ms	1.4 ms	0.5	0.25 kl	6 ms	6 ms
Synaptic parameters						
τ_f (u to v)	τ_f (v to v)	τ_x	τ_y			
2.8 ms	0.3 ms	1.3 ms	4.3 ms			
STDP rule parameters						
α^+	α^-	w_{offset}	ϵ			
0.004	0.024	0.2	0.1			
Homeostatic mechanism parameters						
τ_w	$c_{\min}$	$c_{\max}$				
1/3 TP	9 V_θ^v	91 V_θ^v				

Lastly, the incoherence measure is defined, with sc and sp denoting the total number of coherent and paradoxical SBMU, respectively:

$$\text{Incoherence} = 1 - \frac{sc}{sc + sp}$$

High incoherence may for instance potentially lead to a high root mean square (RMS) reconstruction error.

3.2 Parameters of W-TCRL

A common challenge in training spiking neural networks (SNNs) is their limited adaptability to varying data dimensionalities, often requiring extensive hyperparameter tuning.

To address this issue, we propose a generic parametrization approach for the W-TCRL model that can handle arbitrary data dimensionalities, denoted as k. Only three parameters in the representation layer, namely the neuron firing threshold V_θ^v, the initial lateral inhibition level $c_{\min}$, and the final inhibition level $c_{\max}$, depend on the data dimensionality. For simplicity, we assume a linear relationship between the data dimensionality k and these three parameters.

In our parametrization, each encoder neuron emits a single spike to represent a continuous input, and the total number of spikes in the encoding layer is proportional to the data dimensionality k, given by $k \times l$, where l represents the number of neurons used to represent each dimension of the input space. As spikes are weighted by synaptic weights within the range of $[0, 1]$, a neuron can only fire if its firing threshold V_θ^v is set to $c \times k \times l$, where c is a coefficient ranging from 0 to 1.

Additionally, to maintain a Winner-Take-All (WTA) behavior in the representation layer, it is crucial to calibrate the lateral inhibition level based on the data dimensionality. This ensures that the inhibition level is sufficiently high to prevent other neurons from reaching their firing threshold V_θ^v.

We determined the coefficients of the linear equations relating these three parameters to the input data dimensionality k and the number of encoder neurons l using the Bayesian optimization method known as Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011). Our objective was to minimize the root mean square (RMS) reconstruction error on the MNIST and natural image datasets, considering input vectors with varying dimensions. The aim was to obtain generic and robust coefficients capable of handling diverse input data distributions and dimensionalities.

By incorporating the optimized coefficient values into the linear equations, the values of the three hyperparameters can be determined as follows:

$$V_\theta^v = 0.25kl \tag{8}$$

$$c_{\min} = 9V_\theta^v \tag{9}$$

$$c_{\max} = 91V_\theta^v \tag{10}$$

The parameters of W-TCRL used in all simulations are given in Table 1.

3.3 Results for MNIST

Experiments were run with varying network sizes with $m \in \{16, 32, 64, 128, 256\}$ neurons in the representation layer and therefore m code vectors submitted to learning. Each experiment was evaluated with 30 independent simulations. For a fair comparison with the state of the art set by Tavanaei et al. (2018), we used the same experimental protocol, including the lack of cross-validation.

Synaptic weights between the encoding layer u and the representation layer v are randomly initialized in the interval $[0.6, 0.8]$. The inputs are normalized within the interval $[0.15, 0.85]$. We introduce a margin due to the projection of linear input data onto a circular space (receptive fields) where extreme values become equivalent (2π is equivalent to 0 radians). In addition, MNIST is essentially composed of extreme values (black and white pixels).

For training and testing, subsets of 15,000 and 1,000 handwritten digits were respectively used. These subsets provided

5×5 pixel patches extracted from 28×28 pixel digits as SNN inputs. The dimension-dependent parameters V_θ^v, $c_{\min}$, $c_{\max}$ were automatically determined for $5 \times 5 = 25$ dimensions by Equations (8), (9), (10).

In the training phase, 60,000 image patches were utilized, while the test phase held the lateral inhibition in the representation layer at its maximum ($c_{\max}$), with plasticity disabled.

Performance was assessed based on the three aforementioned metrics: the RMS reconstruction error between an input vector and the code vector of the SBMU, the sparsity of the representation layer activity, and the incoherence in the self-organized process of SBMU election.

3.3.1 Learning phase

Figure 6 shows that the different performance measures consistently enhance with the progression of learning iterations, and also with an increasing number of neurons $m \in \{16, 32, 64, 128, 256\}$ in the representation layer v. The RMS reconstruction error decreases, indicating that W-TCRL learns to compress the distribution of input data, reaching a relatively stable plateau after 8,000 iterations with an RMS error of about 0.08 for $m \in \{64, 128, 256\}$ neurons. The sparsity also decreases, indicating that the percentage of neurons firing in the representation layer v decreases in response to an input vector. This is attributed to heightened lateral inhibition through homeostasis and increased neuronal specialization. Lastly, Euclidean incoherence in the SBMU election process reduces through training iterations for a decision threshold x at 5 and 10%, signaling fewer paradoxical SBMUs and improved neuronal receptive field selectivity for specific input regions. This shows that the first firing neuron is, on average, an accurate representative of the input spike pattern.

Figure 6 shows that the code vectors in the representation layer gradually become selective to

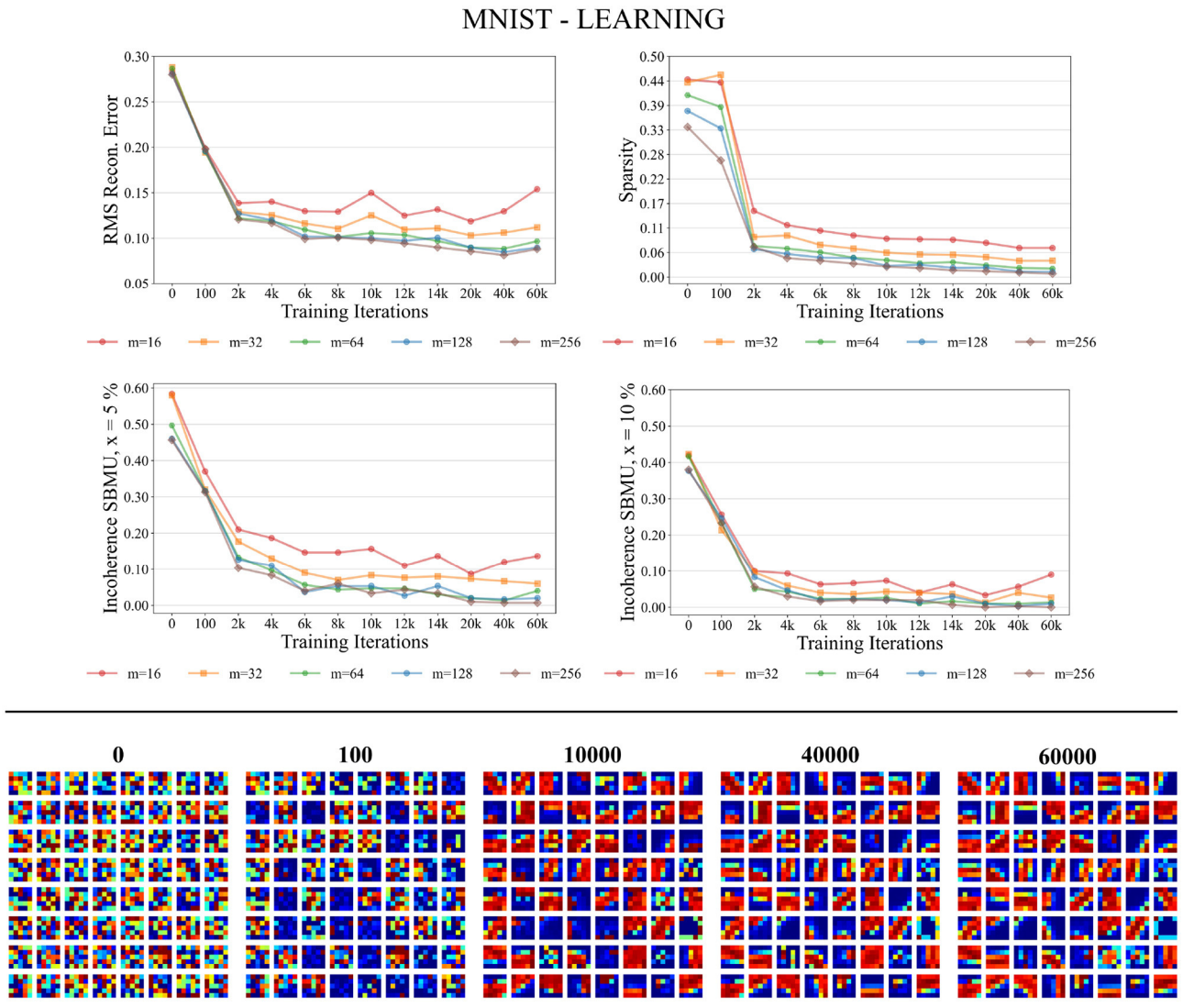


FIGURE 6
Performance for MNIST during the training phase. **Top**: Average model performance for different network size through training iterations in terms of RMS reconstruction error, sparsity, and incoherence for a decision threshold x at 5 and 10%. **Bottom**: Decoded code vectors for $m = 64$ neurons after 0, 100, ..., 60,000 training iterations. The blue-red gradient represent minimum and maximum values, respectively.

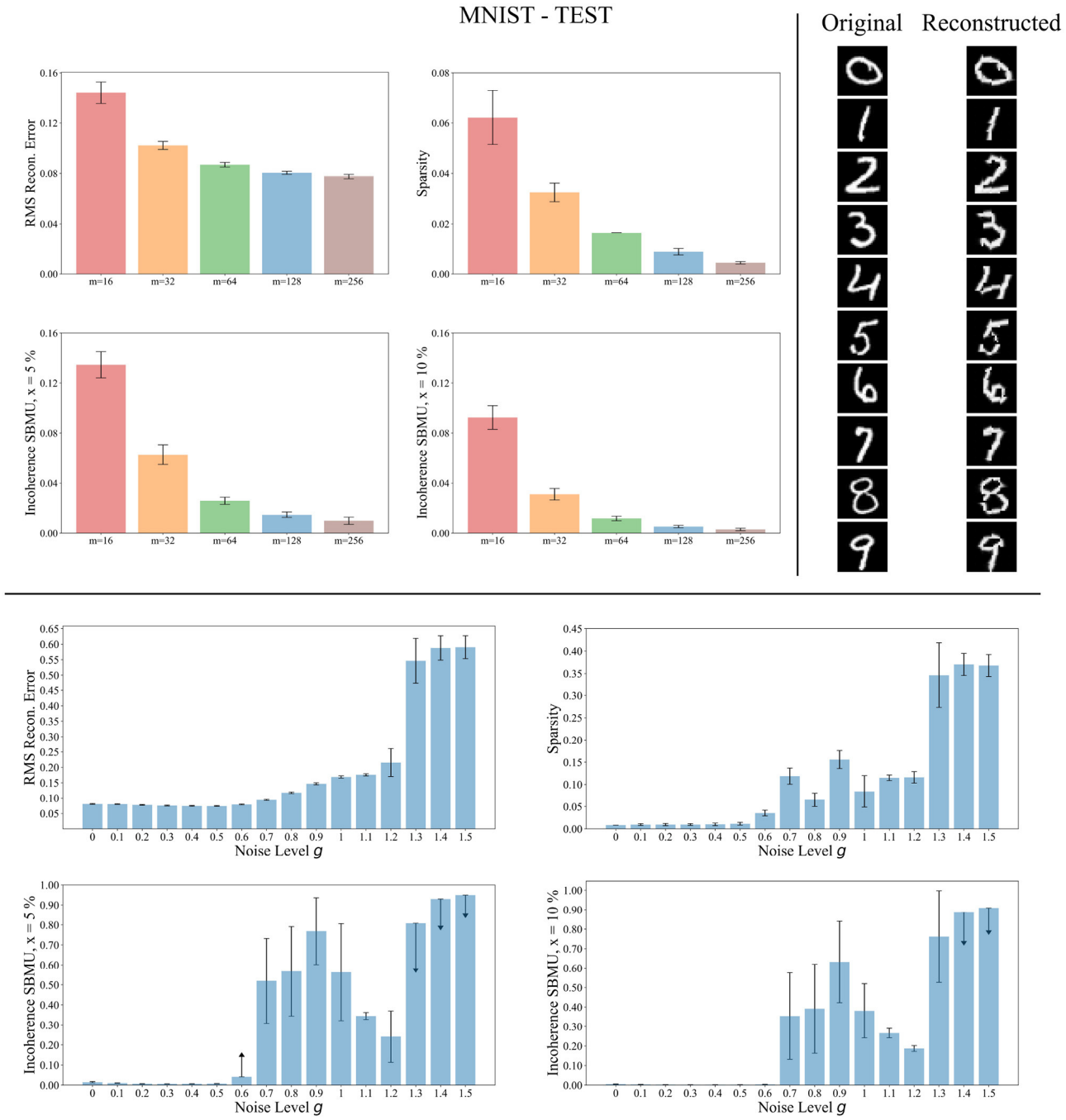


FIGURE 7
Performance for MNIST during the testing phase. **Top left**: Average model performance for different network size in terms of RMS reconstruction error, sparsity and incoherence for a decision threshold x at 5 % and 10 %. Error bars show standard error of the mean for 30 independent runs. **Top right**: Set of original images presented as input to the SNN and reconstructed using the code vectors of neurons in the representation layer, with $m = 128$ neurons. **Bottom**: Average model performance for different noise levels g with $m = 128$ neurons. Error bars show standard error of the mean for 30 independent runs.

various visual orientations over the course of the learning iterations.

3.3.2 Testing phase

We now evaluate the performance of W-TCRL on the test dataset for MNIST. The achieved performances all increase with the capacity of the SNN (see Figure 7). The average sparsity for $m = 256$ neurons reaches a value of 0.004, meaning that only 0.4% of the neurons in the representation layer are active on average. Self-organized SBMU selection is achieved with up to 99.0% of coherent SBMU for a decision threshold of 5% and $m = 256$ neurons. The best RMS reconstruction error reaches the value of 0.08 (see Table 2). This represents a relative improvement of 53%

TABLE 2 Average results on 30 independent runs obtained on MNIST and natural images datasets for a given number of neurons $m \in \{16, 32, 64, 128, 256\}$.

Dataset	MNIST					Natural images				
Neurons m	16	32	64	128	256	16	32	64	128	256
RMS	0.144	0.102	0.087	0.080	0.078	0.164	0.138	0.061	0.056	0.056
Sparsity	0.062	0.032	0.016	0.009	0.004	0.255	0.234	0.141	0.076	0.043
Incoherence 5 %	0.134	0.063	0.026	0.015	0.010	0.516	0.473	0.361	0.275	0.224
Incoherence 10 %	0.092	0.031	0.012	0.005	0.003	0.378	0.344	0.271	0.176	0.064

TABLE 3 Best RMS reconstruction errors reported by Tavanaei et al. (2018) are compared to our model W-TCRL.

Dataset	Tavanaei	W-TCRL (our model)
MNIST	0.17	**0.08**
Natural images	0.24	**0.06**

Bold values represent the performances of our model.

compared to the SNN state-of-the-art (see Table 3) for learning representations using the MNIST dataset.

The reconstruction of images from the self-organized elected code vectors found by W-TCRL produces well-reconstructed handwritten digits, comparable to the original images (see Figure 7). The quality of the reconstructed digits demonstrates the effectiveness of our SNN architecture in capturing and representing the visual information accurately.

We also tested the robustness of our network's ability to learn under noise (see Figure 7). During the training phase, we varied the white noise scaling factor g in the range $[0, 1.5]$, resulting in spike jitter up to around 5 ms in spikes outputted by the encoding layer. The performance remains consistent across the different metrics for values of g up to 0.5, demonstrating that noise doesn't prevent neurons in the representation layer from learning statistical correlations from input spikes. However, with $g > 1.2$, the noise becomes too strong. It destroys the temporal information contained in the spike pattern. Consequently, the neurons fail to extract meaningful representations (high RMS reconstruction error), tend to fire simultaneously (low sparsity), and lose their selectivity (high incoherence).

3.4 Results for natural images

As for MNIST, experiments for the natural images dataset were conducted for several network sizes with $m \in \{16, 32, 64, 128, 256\}$ neurons in the representation layer and thus m code vectors subject to learning. Each experiment was evaluated with 30 independent simulations.

The synaptic weights between the encoding layer u and the representation layer v are randomly initialized in the interval $[0.6, 0.8]$. Input normalization was performed to scale inputs within the interval $[0.05, 0.95]$. 16×16 pixel patches, extracted from natural images of 512×512 pixels, are provided as input vectors to the SNN. The dimension-dependent parameters V_θ^v, $c_{\min}$, $c_{\max}$ were automatically determined for $16 \times 16 = 256$ dimensions using Equations (8), (9), (10). A total of 60,000 image patches are provided to the SNN during the training phase. In the test phase, lateral inhibition in the representation layer v was set to its maximum value, $c_{\max}$, and plasticity was disabled.

Similarly to the previous section, we evaluated the performance of W-TCRL using the same three metrics: the root mean squared (RMS) reconstruction error, the sparsity of the representation layer activity, and the Euclidean incoherence of the SBMU.

3.4.1 Training phase

As depicted in Figure 8 the performance measures for the natural images dataset in the training process consistently improve as the number of neurons in the representation layer increases. When the network capacity is too small, specifically with 16 and 32 neurons in the representation layer, the RMS reconstruction error is significantly degraded. In the case of higher network capacities, specifically with 64, 128, and 256 neurons in the representation layer, the RMS reconstruction error exhibits a consistent decrease throughout the training process. After approximately 6,000 iterations, the RMS error reaches a stable plateau, converging to a value of approximately 0.06. The sparsity of the representation layer activity decreases over the course of training, reaching a plateau after 6,000 iterations. The euclidian incoherence of the SBMU election decreases when using a decision threshold of 5% and 10%, but the euclidian incoherence exhibits more fluctuations compared to the results obtained for the MNIST dataset. This observation can be attributed to the high dimensionality of the input data (256 dimensions), which causes the Euclidean distances to become concentrated and thus difficult to distinguish. This hypothesis is further supported by a low RMS reconstruction error and its low fluctuation.

Figure 8 demonstrates that the code vectors in the representation layer exhibit an increasing selectivity toward visual patterns as the number of training iterations increases.

3.4.2 Testing phase

We now evaluate the performance of W-TCRL on the test dataset for natural images. The obtained performance increases as the capacity of the SNN increases, i.e., the number of neurons $m \in \{16, 32, 64, 128, 256\}$ in the representation layer

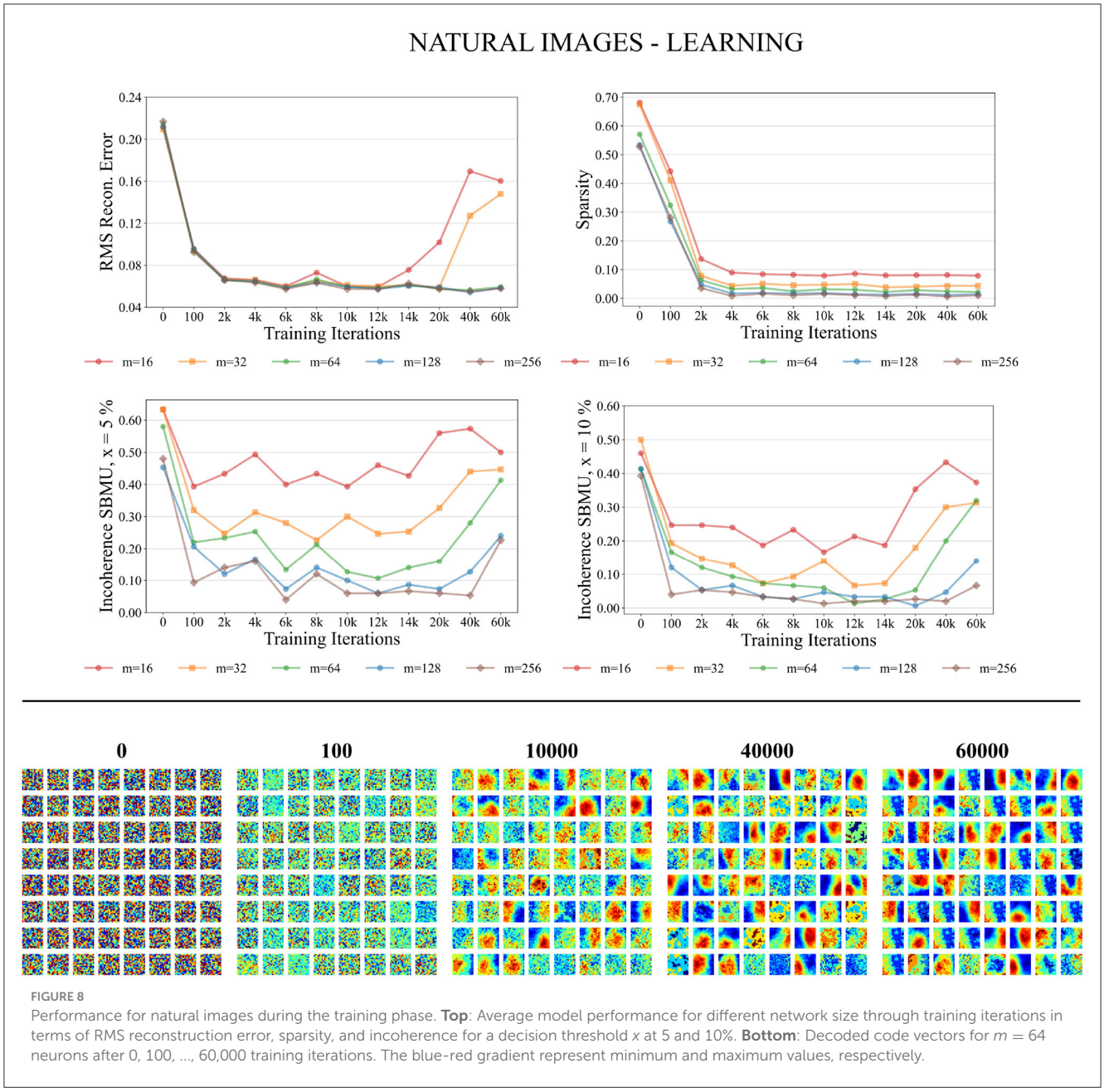


FIGURE 8
Performance for natural images during the training phase. **Top**: Average model performance for different network size through training iterations in terms of RMS reconstruction error, sparsity, and incoherence for a decision threshold x at 5 and 10%. **Bottom**: Decoded code vectors for $m = 64$ neurons after 0, 100, ..., 60,000 training iterations. The blue-red gradient represent minimum and maximum values, respectively.

(see Figure 9). The average sparsity for $m = 256$ neurons reaches a value of 0.04 indicating that only 4% of the neurons in the representation layer are active on average. The self-organized election of the SBMU is achieved with up to 95.6% of coherent SBMU for a decision threshold of 10% and $m = 256$ neurons. The RMS reconstruction error reaches a plateau at 0.06 for $m \in \{64, 128, 256\}$ neurons (see Table 2). This represents a relative improvement of 75% compared to the SNN state of the art (see Table 3) for this dataset of natural images.

Figure 9 demonstrates that the self-organized elected code vectors of W-TCRL yield reconstructed images that closely resemble the original images (though the visual quality inherently suffers from large sizes of patches, regardless of the model used for representation learning).

4 Discussion

In contrast to rate-based encoding methods, which require up to 255 spikes for an 8-bit pixel value (King et al., 2013; Burbank, 2015; Tavanaei et al., 2018), our approach employs latency-based encoding with population coding (Ebitz and Hayden, 2021). This method uses 10 neurons per input coordinate, each emitting a single spike, resulting in a sparse and efficient encoding with only 10 spikes.

Our encoding method diverges from the widely used Time-To-First-Spike (TTFS) coding, which necessitates knowledge of stimulus onset timing. This information is available to the experimenter but not the neurons, which makes it biologically implausible. In contrast, our use of population coding embeds information within the spike pattern itself, specifically in the

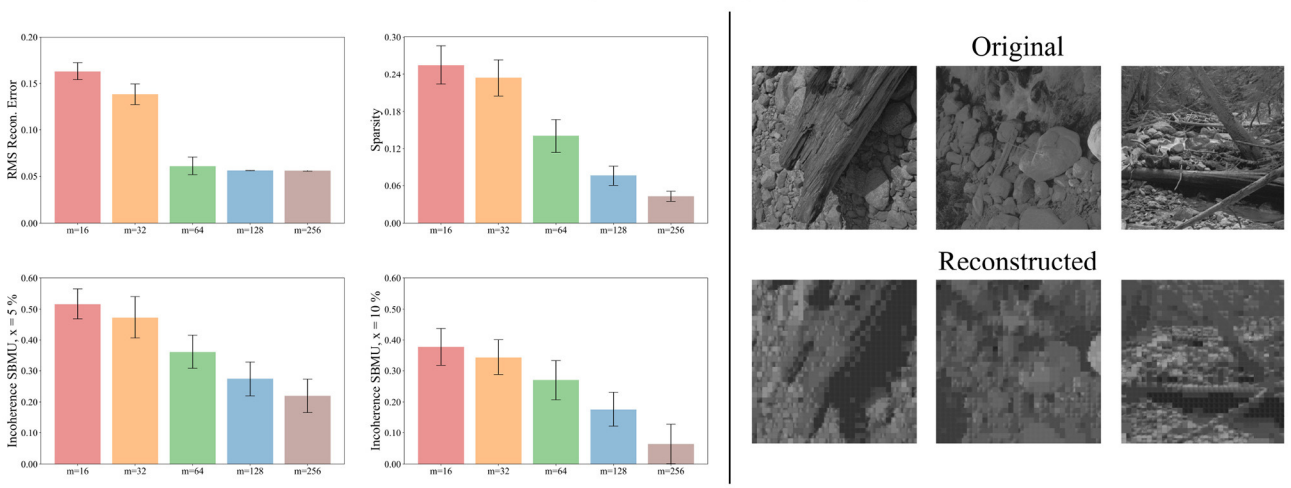


FIGURE 9
Performance for natural images during the testing phase. **Left**: Average model performance for different network size in terms of RMS reconstruction error, sparsity and incoherence for a decision threshold x at 5 and 10%. Error bars show standard error of the mean for 30 independent runs. **Right**: Set of original images presented as input to the SNN and reconstructed using the code vectors of neurons in the representation layer, with $m = 128$ neurons.

timing relative to other neurons in the population. This allows downstream neurons to process these spike patterns in an event-based manner. More generally, population-based latency coding can be viewed as an extension of TTFS. It provides a more flexible representation by distributing information across multiple dimensions (population of neurons) rather than one (single neuron). This enhances flexibility and transmission speed with minimal extra spikes. Although population-based latency coding does not require an external time reference, it does need an internal one, similar to rate coding which needs a temporal window for averaging spikes. As Zhou et al. (2012) noted, the brain may generate several internal time references, including large-scale oscillations (Hopfield, 1995), local field potentials (Eggermont, 1998), and population onsets (Stecker and Middlebrooks, 2003; Chase and Young, 2007).

One limitation of latency-coding is its sensitivity to noise (Guo et al., 2021). This issue is not exclusive to this coding scheme. All neural codes relying on precise spike timing are vulnerable to noise and jitter. A potential solution is to pool information from a neuron population, as we have done, instead of relying on a single neuron as in TTFS. Our experimental results demonstrate that the combination of population-based latency coding with our new STDP rule makes the model quite robust in learning from noisy input spike patterns. The correlation among a neuron population can establish an error-correcting code that mitigates noise effects. This is because the spike timing jitter in relation to other neurons could be less than the jitter referenced to the onset of the stimulus (Zhou et al., 2012).

The extraction of representations from a temporal code lacked event-based local learning rules aimed at achieving low reconstruction error. Our work bridges this gap by introducing a novel STDP rule that extracts spatio-temporal centroids. This simple rule enables stable learning of the distribution of relative latencies within the synaptic weight distribution. Unlike backpropagation algorithms, which require costly global information transport and significant memory for sequence history storage, our STDP rule operates on an event-based and local level in both time and space. The event-based and local operations of our STDP rule align well with the principles of asynchrony and locality inherent in neuromorphic processors (Davies et al., 2018). This makes our STDP rule an ideal fit for such implementations.

Early presynaptic spikes, carrying the most input information, trigger the first (causal) case of weight adaptation on postsynaptic spikes, leading to synaptic weights $w_{ji} > 0$. The learned weight w_{ji} is a function of the latency $\Delta_t = t_{post} - t_{pre}$ between the pre and postsynaptic spikes. Earlier spikes result in higher weights and have a greater impact on postsynaptic spike emission. This differs from the classical STDP rule, which maximizes weight strengthening for late-arriving spikes ($t_{pre} \approx t_{post}$) and usually leads to weight saturation rather than converging to a stable value. Later presynaptic spikes, containing less information, trigger the second (anti-causal) case of weight adaptation, causing synaptic depression ($w_{ji} \to 0$). Through these adaptations, a postsynaptic neuron learns a code vector or filter in its afferent synaptic weights. This facilitates rapid decision-making before receiving all presynaptic spikes, thereby accelerating the SNN's processing speed and mirroring biological neurons.

Rank-order TTFS coding (Thorpe and Gautrais, 1998) follow a similar approach to build detectors, where the weights decrease linearly with the spike rank. The synaptic weights are M (maximum) for the first spike, then M-1, ...,1. This approach focuses on the spike order rather than their precise timings, contrary to the basis of our approach. An accompanying modulation function emphasizes the importance of earliest spikes. The conjunction of a set of weights and a modulation function allows to build accurate detectors of spike patterns (Bonilla et al., 2022). This modulation function concept can be related to other TTFS coding works that utilize precise spike timing for supervised learning

tasks (Rueckauer and Liu, 2018; Zhang et al., 2022; Sakemi et al., 2023). These studies enhance the impact of the earliest spike by linearly increasing the postsynaptic potential (PSP) over time. Our model naturally achieves a similar effect through the CuBa LIF model. The PSP shape of the CuBa LIF is determined by the ratio τ_f/τ_m of the synapse's exponential decay time constant to that of the neuron (Göltz et al., 2021). By selecting $\tau_f > \tau_m$ as done in our experiments, the earliest spikes have a greater impact on the postsynaptic neuron response due to their influence being integrated over a longer period.

To implement competitive learning among neurons in the representation layer, our SNN incorporates lateral inhibitory connections. During the learning phase, a mechanism is employed to initially recruit multiple neurons, which accelerates the convergence of the SNN. Subsequently, the number of recruited neurons gradually decreases to promote specialization and decorrelation of the learned code vectors. This behavior is achieved through a homeostatic mechanism that increases the magnitudes of inhibitory lateral weights. As a result, the circuit transitions dynamically from a k-winners-take-all (k-WTA) to a winner-take-all (WTA) behavior, thereby increasing the sparsity of activity in the representation layer.

While representation learning in ANNs is a mature field, achieving good performance with SNNs remains challenging. Our W-TCRL model, based on temporal coding and a novel STDP rule, outperforms the state-of-the-art of SNNs using a rate-based coding approach by Tavanaei et al. (2018) in terms of RMS reconstruction error. Our method exhibits a relative improvement of 53% for MNIST and 75% for natural images. The reconstructed images exhibit a good visual quality compared to the original images.

The representation layer exhibits a highly sparse activity during inference, a desirable feature (Frenkel, 2021). For instance, on the MNIST dataset with $m = 256$ neurons, the average sparsity is 0.004, indicating that only 0.4% of neurons in the representation layer fire. Notably, neurons in the representation layer fire only once, distinguishing our approach from rate coding methods. In the work of Tavanaei et al. (2018), sparsity was averaged over a time window of $T = 40$ time steps for a given number of neurons m. The reported best sparsity of 9% implies that, on average, 9% of neurons fire at each time step within the encoding window T. In contrast, in our temporal coding-based SNN, a sparsity of 0.4% means that 0.4% of neurons fire a single time during the encoding window T. Therefore, based on the best reported sparsity by Tavanaei et al. (2018), a neuron fires an average of $0.09*40 = 3.6$ spikes during the temporal window T. In contrast, based on our best sparsity results, a neuron emits an average of 0.004 spikes during the temporal window T. Thus, our method achieves up to 900 times fewer spikes emitted per neuron in the representation layer on average, providing substantial benefits in terms of energy consumption and bandwidth in spike communication protocols such as AER, while improving the RMS reconstruction error. This empirical evidence confirms the benefits of using temporal codes compared to rate coding.

Lastly, the self-organized selection of the Spiking Best Matching Unit (SBMU) in our SNN achieves up to 99.0% coherent SBMUs (for MNIST with $m = 256$ neurons and a decision threshold x set to 5%). The high coherence of the SBMUs indicates that the neurons in the representation layer have developed a high level of selectivity for specific regions within the spatio-temporal input space. This selectivity results in an effective clustering of the spatio-temporal input space.

Future research should prioritize implementing our spiking representation learning model on neuromorphic processors (Davies et al., 2018) to enable real-time, low-power, and high-throughput processing, taking advantage of their parallelism and efficiency. Additionally, exploring the unsupervised extraction of hierarchical representations from sensory data by stacking multiple layers using our STDP rule can uncover more complex and abstract features in self-organizing SNN architectures. This unsupervised approach is exemplified by the HOTS architecture (Lagorce et al., 2017), which employs a three-layer stack for unsupervised feature extraction, with the final layer's output fed to a simple histogram classifier that achieves near 100 % accuracy on DVS datasets.

Data availability statement

Publicly available datasets were analyzed in this study. These data can be found at: MNIST: http://yann.lecun.com/exdb/mnist/; Natural Images: http://www.rctn.org/bruno/sparsenet/.

Author contributions

AF was the originator of the W-TCRL model. AF and BG developed its formalized expression and its analysis. AF carried out all simulations, and wrote the manuscript with contributions of BG. Both authors contributed to the article and approved the submitted version.

Funding

This work has been supported by ANR project SOMA ANR-17-CE24-0036.

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

Bengio, Y., Courville, A. C., and Vincent, P. (2013). Representation learning: a review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 1798–1828. doi: 10.1109/TPAMI.2013.50

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems 24*, eds J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger (Granada: Curran Associates Inc.), 2546–2554.

Billings, G., and van Rossum, M. C. W. (2009). Memory retention and spike-timing-dependent plasticity. *J. Neurophysiol.* 101, 2775–2788. doi: 10.1152/jn.91007.2008

Bonilla, L., Gautrais, J., Thorpe, S., and Masquelier, T. (2022). Analyzing time-to-first-spike coding schemes: a theoretical approach. *Front. Neurosci.* 16, 971937. doi: 10.3389/fnins.2022.971937

Burbank, K. S. (2015). Mirrored STDP implements autoencoder learning in a network of spiking neurons. *PLoS Comput. Biol.* 11, e1004566. doi: 10.1371/journal.pcbi.1004566

Chase, S. M., and Young, E. D. (2007). First-spike latency information in single neurons increases when referenced to population onset. *Proc. Natl. Acad. Sci. U.S.A.* 104, 5175–5180. doi: 10.1073/pnas.0610368104

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

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

Dosselmann, R., and Yang, X. D. (2011). A comprehensive assessment of the structural similarity index. *Signal Image Video Process.* 5, 81–91. doi: 10.1007/s11760-009-0144-1

Ebitz, R. B., and Hayden, B. Y. (2021). The population doctrine in cognitive neuroscience. *Neuron* 109, 3055–3068. doi: 10.1016/j.neuron.2021.07.011

Eggermont, J. J. (1998). Azimuth coding in primary auditory cortex of the cat. II. Relative latency and interspike interval representation. *J. Neurophysiol.* 80, 2151–2161.

Falez, P., Tirilly, P., Bilasco, I. M., Devienne, P., and Boulet, P. (2019). Unsupervised visual feature learning with spike-timing-dependent plasticity: how far are we from traditional feature learning approaches? *Pattern Recogn.* 93, 418–429. doi: 10.1016/j.patcog.2019.04.016

Frenkel, C. (2021). Sparsity provides a competitive advantage. *Nat. Mach. Intell.* 3, 742–743. doi: 10.1038/s42256-021-00387-y

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

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science* 233, 1416–1419.

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., et al. (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nat. Mach. Intell.* 3, 823–835. doi: 10.1038/s42256-021-00388-x

Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2021). Neural coding in spiking neural networks: a comparative study for robust neuromorphic systems. *Front. Neurosci.* 15, 638474. doi: 10.3389/fnins.2021.638474

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature* 376, 33–36.

Horé, A., and Ziou, D. (2010). "Image quality metrics: PSNR vs. SSIM," in *20th International Conference on Pattern Recognition, ICPR 2010* (Istanbul: IEEE), 2366–2369.

Jenkin, M. R. M., Jepson, A. D., and Tsotsos, J. K. (1991). Techniques for disparity measurement. *CVGIP Image Understand.* 53, 14–30.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kheradpisheh, S. R., and Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* 30, 2050027:1–2050027:16. doi: 10.1142/S0129065720500276

King, P. D., Zylberberg, J., and DeWeese, M. R. (2013). Inhibitory interneurons decorrelate excitatory cells to drive sparse code formation in a spiking model of V1. *J. Neurosci.* 33, 5475–5485. doi: 10.1523/JNEUROSCI.4188-12.2013

Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Netw.* 37, 52–65. doi: 10.1016/j.neunet.2012.09.018

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). HOTS: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lee Rodgers, J., and Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *Am. Stat.* 42, 59–66.

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., and Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Front. Neurosci.* 14, 119. doi: 10.3389/fnins.2020.00119

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 6178–6190. doi: 10.1109/TNNLS.2018.2826721

Patel, D., Hazan, H., Saunders, D. J., Siegelmann, H. T., and Kozma, R. (2019). Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game. *Neural Netw.* 120, 108–115. doi: 10.1016/j.neunet.2019.08.009

Pfister, J.-P., and Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *J. Neurosci.* 26, 9673–9682. doi: 10.1523/JNEUROSCI.1425-06.2006

Rueckauer, B., and Liu, S.-C. (2018). "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (Florence: IEEE), 1–5.

Rumbell, T., Denham, S. L., and Wennekers, T. (2014). A spiking self-organizing map combining STDP, oscillations, and continuous learning. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 894–907. doi: 10.1109/TNNLS.2013.2283140

Sakemi, Y., Morino, K., Morie, T., and Aihara, K. (2023). A Supervised learning algorithm for multilayer spiking neural networks based on temporal coding toward energy-efficient VLSI processor design. *IEEE Trans. Neural Netw. Learn. Syst.* 34, 394–408. doi: 10.1109/TNNLS.2021.3095068

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

Stecker, G. C., and Middlebrooks, J. C. (2003). Distributed coding of sound locations in the auditory cortex. *Biol. Cybernet.* 89, 341–349. doi: 10.1007/s00422-003-0439-1

Tavanaei, A., Masquelier, T., and Maida, A. (2018). Representation learning using event-based STDP. *Neural Netw.* 105, 294–303. doi: 10.1016/j.neunet.2018.05.018

Thorpe, S. J., and Gautrais, J. (1998). "Rank order coding," in *Computational Neuroscience*, ed J. M. Bower (Boston, MA: Springer US), 113–118.

Yen, E. K., and Johnston, R. G. (1996). *The Ineffectiveness of the Correlation Coefficient for Image Comparisons*. Technical Report LA- UR-96-2474, Los Alamos, NM.

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2022). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1947–1958. doi: 10.1109/TNNLS.2021.3110991

Zhou, Y., Mesik, L., Sun, Y. J., Liang, F., Xiao, Z., Tao, H. W., et al. (2012). Generation of spike latency tuning by thalamocortical circuits in auditory cortex. *J. Neurosci.* 32, 9969–9980. doi: 10.1523/JNEUROSCI.1384-12.2012

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 05 January 2024
DOI 10.3389/fnins.2023.1291051

OPEN ACCESS

EDITED BY
Priyadarshini Panda,
Yale University, United States

REVIEWED BY
Garrick Orchard,
Facebook Reality Labs Research, United States
Yujie Wu,
Tsinghua University, China
Qi Xu,
Dalian University of Technology, China

*CORRESPONDENCE
Ying Fang
✉ fy20@fjnu.edu.cn
Qifeng Li
✉ liqf@nercita.org.cn

RECEIVED 08 September 2023
ACCEPTED 06 December 2023
PUBLISHED 05 January 2024

CITATION
Li H, Wan B, Fang Y, Li Q, Liu JK and An L (2024) An FPGA implementation of Bayesian inference with spiking neural networks.
Front. Neurosci. 17:1291051.
doi: 10.3389/fnins.2023.1291051

An FPGA implementation of Bayesian inference with spiking neural networks

Haoran Li[1], Bo Wan[2,3], Ying Fang[4,5]*, Qifeng Li[6]*, Jian K. Liu[7] and Lingling An[1,2]

[1]Guangzhou Institute of Technology, Xidian University, Guangzhou, China, [2]School of Computer Science and Technology, Xidian University, Xi'an, China, [3]Key Laboratory of Smart Human Computer Interaction and Wearable Technology of Shaanxi Province, Xi'an, China, [4]College of Computer and Cyber Security, Fujian Normal University, Fuzhou, China, [5]Digital Fujian Internet-of-Thing Laboratory of Environmental Monitoring, Fujian Normal University, Fuzhou, China, [6]Research Center of Information Technology, Beijing Academy of Agriculture and Forestry Sciences, National Engineering Research Center for Information Technology in Agriculture, Beijing, China, [7]School of Computer Science, University of Birmingham, Birmingham, United Kingdom

Spiking neural networks (SNNs), as brain-inspired neural network models based on spikes, have the advantage of processing information with low complexity and efficient energy consumption. Currently, there is a growing trend to design hardware accelerators for dedicated SNNs to overcome the limitation of running under the traditional von Neumann architecture. Probabilistic sampling is an effective modeling approach for implementing SNNs to simulate the brain to achieve Bayesian inference. However, sampling consumes considerable time. It is highly demanding for specific hardware implementation of SNN sampling models to accelerate inference operations. Hereby, we design a hardware accelerator based on FPGA to speed up the execution of SNN algorithms by parallelization. We use streaming pipelining and array partitioning operations to achieve model operation acceleration with the least possible resource consumption, and combine the Python productivity for Zynq (PYNQ) framework to implement the model migration to the FPGA while increasing the speed of model operations. We verify the functionality and performance of the hardware architecture on the Xilinx Zynq ZCU104. The experimental results show that the hardware accelerator of the SNN sampling model proposed can significantly improve the computing speed while ensuring the accuracy of inference. In addition, Bayesian inference for spiking neural networks through the PYNQ framework can fully optimize the high performance and low power consumption of FPGAs in embedded applications.Taken together, our proposed FPGA implementation of Bayesian inference with SNNs has great potential for a wide range of applications, it can be ideal for implementing complex probabilistic model inference in embedded systems.

KEYWORDS

spiking neural networks, probabilistic graphical models, Bayesian inference, importance sampling, FPGA

1 Introduction

Neuroscience research plays an increasingly important role in accelerating and inspiring the development of artificial intelligence (Demis et al., 2017; Zador et al., 2022). Spikes are the fundamental information units in the neural systems of the brain (Bialek et al., 1999; Yu et al., 2020), which also play an important role in information transcoding and representation in artificial systems (Zhang et al., 2020; Gallego et al., 2022; Xu et al., 2022). Spiking neural

networks (SNNs) utilize spikes as brain-inspired models are proposed as a new generation of computational framework (Maass, 1997). SNNs have received extensive attention and can utilize many properties of artificial neural networks for deep learning in various tasks (Kim et al., 2018; Shen et al., 2021; Yang et al., 2022).

Numerous neuroscience experiments (Ernst and Banks, 2002; Körding and Wolpert, 2004) have shown that the cognitive and perceptual processes of the brain can be expressed as a probabilistic reasoning process based on Bayesian reasoning. From the macroscopic perspective, Bayesian models have explained how the brain processes uncertain information and have been successfully applied in various fields of brain science (Shi et al., 2013; Chandrasekaran, 2017; Alais and Burr, 2019). In contrast, recent studies focus on implementing SNNs using probabilistic graphical models (PGMs) at the micro level (Yu et al., 2018a,b, 2019; Fang et al., 2019). However, the realization of PGMs is considerably slow due to the sampling process. Since probabilistic sampling on SNNs involves massive probabilistic computations that can consume a lot of time and many computationally intensive operations are involved in processing the data in the neural network, the inference speed will be even slower with the scale of the problem. In some practical application scenarios such as medical diagnosis, environmental monitoring, intelligent monitoring, etc., these problems lead to poor real-time application, which causes a series of problems. Therefore, we want to do some acceleration and improvements to meet the demand for speed in real applications. At present, there are dedicated hardware designs for SNNs (Cai et al., 2018; Liu et al., 2019; Fang et al., 2020; Han et al., 2020; Zhu et al., 2022), and for PGMs based on conventional artificial neural networks (Cai et al., 2018; Liu et al., 2020; Fan et al., 2021; Ferianc et al., 2021). Yet, there are few studies for hardware platforms to implement PGM-based SNNs. Therefore, it is highly demanding and meaningful for hardware acceleration of PGM-based SNNs, not only for simulation speed-up but for neuromorphic computing implementation (Christensen et al., 2022).

In this study, we address this question by utilizing FPGA hardware to implement a recently developed PGM-badsed SNN model, named the sampling-tree model (STM) (Yu et al., 2019). The STM is an implementation of spiking neural circuits for Bayesian inference using importance sampling. In particular, The STM is a typical probabilistic graphical model based on a hierarchical tree structure with a deep hierarchical structure of layer-on-layer iteration and uses a multi-sampling mode based on sampling coupled with population probability coding. Each node in the model contains a large number of spiking neurons that represent samples. The STM process information based on spikes, where spiking neurons integrate input spikes over time and fire a spike when their membrane potential crosses a threshold. With these properties, the STM is a typical example of PGM-based SNN for Bayesian inference. The software implementation of sampling-based SNN is very time-consuming, and actual tasks are limited by the model running speed on CPU. Therefore, to fulfill our requirements for the running speed of the model, it is necessary to choose a hardware platform for designing a hardware accelerator.

Here we need to consider which hardware platform is chosen to better implement the design of the accelerator.

ASIC-based design implementations: Compared with general integrated circuits, ASIC has the advantages of smaller size, lower power consumption, improved reliability, improved performance,and enhanced confidentiality. ASICs can also reduce costs compared to general-purpose integrated circuits in mass production. Ma et al. (2017) designed a highly-configurable neuromorphic hardware coprocessor based on SNN implemented with digital logic, called Darwin neural processing unit (NPU), which was fabricated as ASIC in SMIS's 180 nm process for resource-constrained embedded scenarios. Tung et al. (2023) proposed a design scheme for a spiking neural network ASIC chip and developed a built-in-self-calibration (BSIC) architecture based on the chip to realize the network to perform high-precision inference under a specified range of process parameter variations. Wang et al. (2023) proposed an ASIC learning engine consisting of a memristor and an analog computing module for implementing trace-based online learning in a spiking neural network, which significantly reduces energy consumption compared to existing ASIC products of the same type. However, ASIC requires a long development cycle and is risky. Once there is a problem, the whole piece will be discarded. Consequently, we do not consider the use of ASIC for design here.

FPGA-based design implementations: FPGA has a shorter development cycle compared to ASIC, is flexible in use, can be used repeatedly, and has abundant resources.

Ferianc et al. (2021) proposed an FPGA-based hardware design to accelerate Bayesian recurrent neural networks (RNNs), it can achieve up to 10 times speedup compared with GPU implementation. Wang (2022) implemented a hardware accelerator on FPGA for the training and inference process of Bayesian belief propagation neural network (BCPNN), and the computing speed of the accelerator can improve the CPU counterpart by two orders of magnitude. However, RNN and BCPNN in the above two designs are essentially traditional neural network architectures, which are different from the hardware implementation of the SNN architecture and cannot be directly applied to our SNN implementation.

In addition, Fan et al. (2021) proposed a novel FPGA-based hardware architecture to accelerate BNNs inferred through Monte Carlo, it can achieve up to nine times better compute efficiency compared with other state-of-the-art BNN accelerators. Awano and Hashimoto (2023) proposed a Bayesian neural network hardware accumulator called B2N2, i.e., Bernoulli random number-based Bayesian neural network accumulator, which reduces resource consumption by 50% compared to the same type of FPGA implementation. For the above two designs, the hardware architecture proposed by Fan and Awano cannot be used for the acceleration of the STM, because the variational inference model and the Monte Carlo inference model are not suitable for importance sampling, but STM needs to be sampled through importance sampling. In other words, the hardware architecture is different due to the different models, so we cannot use these two hardware architectures to accelerate STM on the FPGA.

In summary, many previous designs were implemented on FPGAs because ASIC is less flexible and complex than FPGAs (Ju et al., 2020). GPUs often perform very well on applications that benefit from parallelism, and are currently the most widely

used platform for implementing neural networks. However, GPUs are not able to handle spike communication well in real-time, while the high energy consumption of GPUs leads to limitations in some embedded scenarios. Therefore, we chose the FPGA as a compromise solution, which provides reasonable cost, low power consumption, and flexibility for our design. Furthermore, for some FPGA-based design implementations, due to the limitations of the traditional ANN neural network architecture (Que et al., 2022) and some inference models are not suitable for sampling (Fan et al., 2022), we also need to design a hardware implementation suitable for importance sampling (Shi and Griffiths, 2009). Based on the above design reference and our previous work that the STM of a neural network model for Bayesian inference, we finally chose FPGA to complete the design of the STM accelerator, and also complete the neural network model construction of Bayesian inference on FPGA with the help of PYNQ framework to achieve the acceleration of STM. The overall design idea is as follows. Firstly, optimize the model inference part of the algorithm to make full use of FPGA resources to improve program parallelism, thus reducing the computing delay, and complete the design of custom hardware IP cores. Secondly, the designed IP core is connected to the whole hardware system, and the overall hardware module control is realized according to the preset algorithm flow through the PYNQ framework.

The main contributions of this work are as follows:

- We are the first work targeting acceleration of STM on the FPGA board, and the inference results of the STM implemented on the FPGA are similar to the inference results implemented by the CPU;
- We implemented the acceleration of the STM on a Xilinx Zynq ZCU104 FPGA board, and we also found that the acceleration on the FPGA increases with the problem size, such as the number of model layers, the number of neurons, and other factors;
- We demonstrate that the neural circuits we implemented on the FPGA board can be used to solve practical cognitive problems, such as the integration of multisensory, it can also efficiently perform complex Bayesian reasoning tasks in embedded scenarios.

2 Related work

2.1 Bayesian inference with importance sampling

Existing neural networks using variational-based inference methods such as belief propagation (BP) (Yedidia et al., 2005) and Monte Carlo (MC) (Nagata and Watanabe, 2008) can obtain accurate inference results in some Bayesian models. However, most Bayesian models in the real world are more complex. When using BP (George and Hawkins, 2009) or MCMC (Buesing et al., 2011) to implement Bayesian model inference, each or each group of neurons generally has to implement a different and complex computation in these neural networks. In addition, since spiking neural networks require multiple iterations to obtain optimal Bayesian inference results, they are more complicated to implement. Therefore, STM employs the tree structure of Bayesian networks to convert global inference into local inference through network decomposition. Importance sampling is introduced to perform local inference, which ensures that each group of neurons works simply, making the model suitable for large-scale distributed computing.

Unlike the traditional method of sampling from a distribution of interest, we use importance sampling to implement Bayesian inference for spiking neural networks, which is a method of sampling from a simple distribution to achieve the estimation of a certain function value. When given the variable y, the conditional expectation of a function $f(x)$ is estimated by importance sampling as:

$$\begin{aligned} E(f(x)|y) &= \sum_x f(x)P(x|y) = \frac{\sum_x f(x)P(y|x)P(x)}{\sum_x P(y|x)P(x)} \\ &= \frac{E(f(x)P(y|x))_{P(x)}}{E(P(y|x))_{P(x)}} \approx \sum_{x^i} f(x^i)\frac{P(y|x^i)}{\sum_{x^i} P(y|x^i)}, x^i \sim P(x). \end{aligned} \tag{1}$$

where x^i follows the distribution $P(x)$. This equation transforms the conditional expectation $E(f(x)|y)$ into a weighted combination of normalized conditional probabilities $P(y|x^i)/\sum_{x^i} P(y|x^i)$. Importance sampling can be used to draw a large number of samples from a simple prior, and skillfully convert the posterior distribution into the ratio of likelihood, thereby estimating the expectation of the posterior distribution.

2.2 Sampling-tree model with spiking neural network

To build a general-purpose neural network for large-scale Bayesian models, the STM was proposed in the previous work (Yu et al., 2019), as shown in Figure 1. As a spiking neural network model for Bayesian inference, STM is also a probabilistic graph model with an overall hierarchical structure. Each node in the graph has a large number of neurons as sample data.

The STM is used to explain how Bayesian inference algorithms can be implemented through neural networks in the brain, building large-scale Bayesian models for SNN. In contrast to other Bayesian inference methods, the STM focuses on multiple sets of neurons to achieve probabilistic inference in PGM with multiple nodes and edges. Performing neural sampling on deep tree-structured neural circuits can transform global inference problems into local inference tasks and achieve approximate inference. Furthermore, since the STM does not have neural circuits specifically designed for a specific task, it can be generalized to solve other inference problems. In summary, the STM is a general neural network model that can be used for distributed large-scale Bayesian inference.

In this model, the root node of the Bayesian network is the problem or reason that needs to be inferred in our experiment, the leaf node represents the information or evidence we receive from the outside world, and the branch nodes are the intermediate variable of the reasoning problem. From the macroscopic perspective, the STM is a probabilistic graphical

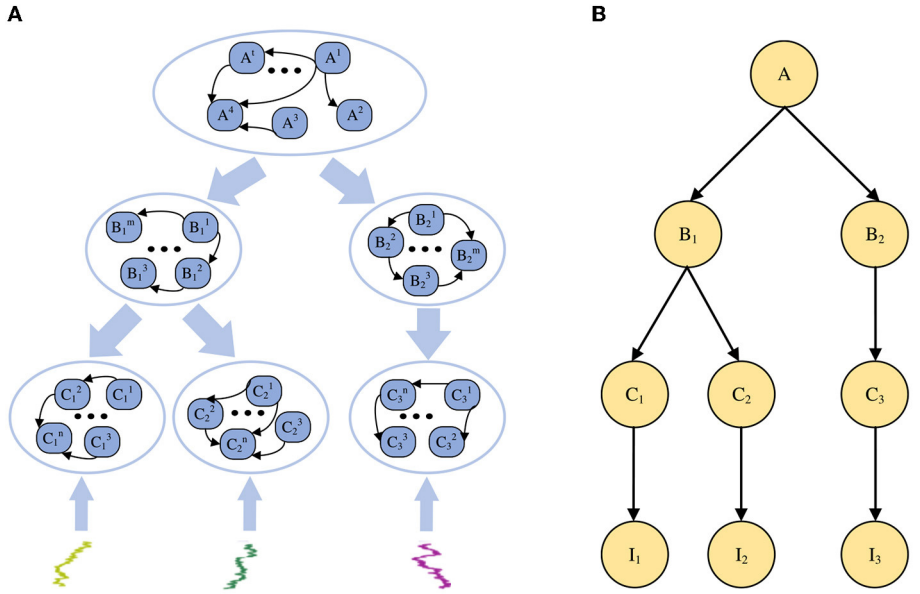


FIGURE 1
Sampling-tree model. **(A)** An example of the STM in spiking neural networks. **(B)** A tree-structured Bayesian network corresponding to the STM in **(A)**.

model with a hierarchical tree structure. From the neuron level, each node in the model contains a group of spiking neurons, and multiple connections between these neurons. Each spiking neuron is regarded as a sample from a special distribution, and the information transmission or probability calculation in the model is achieved through the connections between neurons.

2.3 Hardware implementation using PYNQ framework

PYNQ provides a Jupyter-based framework and Python API for designing programmable logic circuits using the Xilinx adaptive computing platform instead of using ASIC-style design tools. PYNQ consists of three layers: application layer, software layer, and hardware layer. The overall framework is shown in Figure 2. There have been many works implementing neural network acceleration on FPGAs with the help of the PYNQ framework before this.

Tzanos et al. (2019) implemented the acceleration of the Naive Bayesian neural network algorithm on the Xilinx PYNQ-Z1 board. The hardware accelerator was evaluated on Naive Bayes-based machine learning applications. Ju et al. (2020) proposed a hardware architecture to enable efficient implementation of SNNs and validate it on the Xilinx ZCU102. However, this design directly mapped each different computing stage to a hardware layer. Although this approach can improve the parallelism of the program, this direct mapping method would consume a great deal of the hardware resources or even exceed them. Awano and Hashimoto (2020) proposed an efficient inference algorithm for BNN, named BYNQNet, and its FPGA implementation. The Monte Carlo inference method that this design was based on belongs to variational inference, which is very complicated in implementing

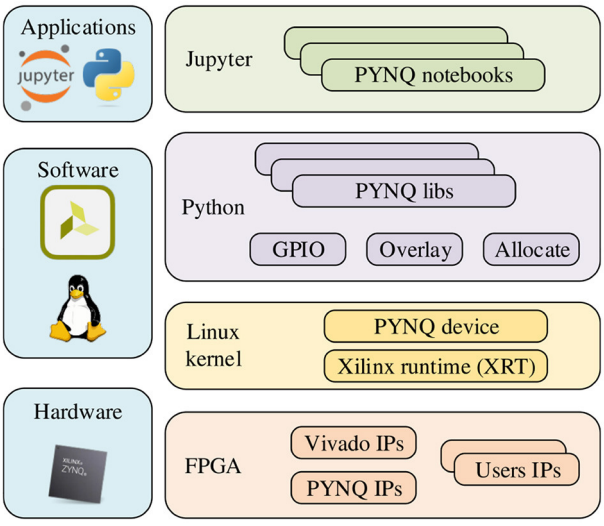


FIGURE 2
Overall framework of using PYNQ to develop Zynq.

larger-scale impulsive neural network models, and the Monte Carlo inference method is not suitable for sampling models.

In our work, we focus on ensuring the inference accuracy of the STM on FPGAs while improving performance. Since the PYNQ framework provides a Python environment that integrates hardware Overlay for easy porting. And with the PYNQ framework, we can implement hardware execution in parallel while creating high-performance embedded applications, and execute more complex analysis algorithms through Python programs, the performance of which can be close to desktop workstations. It also has the advantages of high integration, small size, and

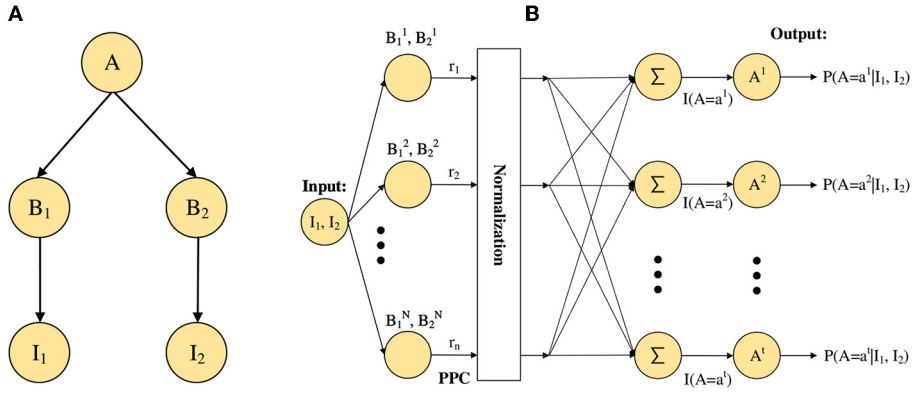


FIGURE 3
The example of Bayesian network. **(A)** A simple Bayesian neural network model. **(B)** The neural network architecture of the STM for the basic network as in **(A)**.

low power consumption. When using the PYNQ framework, the tight coupling between PS (Processing System, i.e., ARM processor) and PL (Programmable Logic, i.e. FPGA part) can achieve better responsiveness, higher reconfigurability, and richer interface functions than traditional methods. The simplicity and efficiency of the Python language and the acceleration provided by programmable logic are also fully utilized. Finally, Xilinx has simplified and improved the design of Zynq-based products on the PYNQ framework by combining a hybrid library that implements acceleration within Python and programmable logic. This is a significant advantage over traditional SoC approaches that cannot use programmable logic. Therefore, we implement the Bayesian neural network inference algorithm on Xilinx ZCU104 with the help of the PYNQ framework.

3 System analysis

In this section, we first summarize the basis of our work on implementing probabilistic inference algorithms for the brain through neural networks. We then analyze the difficulties of accelerating the probabilistic inference algorithm for running neural network models and briefly describe how we address these difficulties.

3.1 Neural network implementation

In this subsection, we take the neural network shown in Figure 3A as an example, and we consider the following two aspects in the implementation of the neural network: First, for the stimulus encoding problem, it is important to know how to accomplish the activities of neurons from stimulus input. Second, for the estimation of posterior probability, it is also necessary to consider how the activities of neurons realize the estimation of posterior probability because our final inference result requires the expectation over posterior distribution.

For the first problem, we convert stimulus input information into the activities of neurons through probabilistic population codes (PPCs) (Ma et al., 2006, 2014). According to PPCs, the activities of these neurons encoding stimuli inputs, I_1, I_2, and others, can be obtained neuronal activity of the root node A. For the second problem, we divide it into two steps, one is the calculation of the posterior probability, and the other is the neural implementation of the posterior probability. Based on importance sampling, we can estimate the posterior probability by the ratio approximation of the likelihood function, as shown in Eq. (2).

$$P(B_1 = B_1^i, B_2 = B_2^i | I_1, I_2) = \frac{P(I_1, I_2 | B_1^i, B_2^i) \cdot P(B_1^i, B_2^i)}{\int P(I_1, I_2 | B_1, B_2) \cdot P(B_1, B_2) \mathrm{d}B_1, B_2} \approx \frac{P(I_1, I_2 | B_1^i, B_2^i)}{\sum_i P(I_1, I_2 | B_1^i, B_2^i)}. \tag{2}$$

Then, for the neural implementation of posterior probability, Shi and Griffiths (2009) have shown that divisive normalization $E(r_i / \sum_i r_i)$ is commonly found in the cerebral cortex by neuroscience experiments, and Eq. (3) has been proved, where r_i is the firing rate of the i^{th} neuron.

$$E(r_i / \sum_i r_i) = \frac{P(I_1, I_2 | B_1^i, B_2^i)}{\sum_i P(I_1, I_2 | B_1^i, B_2^i)}. \tag{3}$$

Next, we will describe the processes and mechanisms of probabilistic inference implemented in the neural network (adapted from Fang et al. 2019). First, for the process of probabilistic inference, the neural network processes external stimulus inputs I_1 and I_2 together in a bottom-up manner, as shown in Figure 3B. Second for the process of generation, which is to generate sampling neurons and the opposite of the inference process. Based on the generative model in Figure 3A, we can get sampling neurons B_1^i and B_2^i from $P(B_1)$ and $P(B_2)$, respectively. In other words, we can get that the sampling neurons follow $B_1, B_2 \sim N(0, \sigma^2)$.

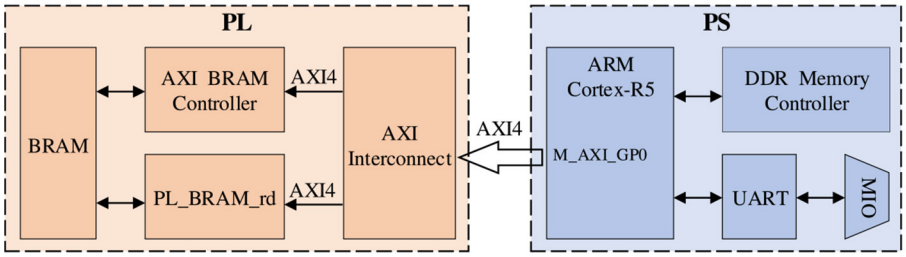


FIGURE 4
Data interaction architecture between PS and PL, here we use m_axi interface for data transmission.

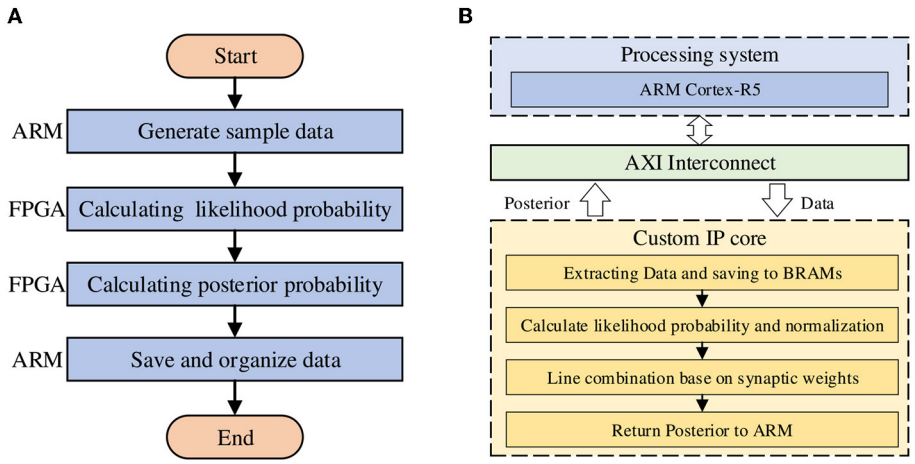


FIGURE 5
The design idea and overall computing architecture. **(A)** The program flow of the model on the ZCU104 board. **(B)** The hardware architecture of the model.

3.2 Difficulties in designing the accelerator

In this work, the communication settings between PS and PL should be considered first in the design of the accelerator. Since the design requires frequent data interactions during operation, the selection of a suitable data interface can ensure the stability of data transmission while improving the time required for data transmission. The second is the design in the PL part, the design of this part is mainly to complete the work of the FPGA, which usually needs to achieve the purpose of acceleration by reducing the Latency of the design.

For the communication setting between PS and PL, since the BRAM in PL part is not enough to store a large amount of data and parameters, it is necessary to exchange data frequently between the PL and PS parts. Therefore, in order to achieve high-speed read/write operations for large-scale data, we use the m_axi interface to realize it. Figure 4 shows the data interaction architecture between PS and PL. The m_axi interface has independent read-and-write channels, supports burst transfer mode, and the potential performance can reach 17GB/s, which fully meets our data scale and transfer speed requirements.

Furthermore, for the design of the PL part, since each node in the model contains a large number of neurons, it will take up a lot of resources, and clocks in the process of encoding, summing, multiplying, and normalizing neurons, in which loops may also be nested. Although pipelines can be added to the loops to improve the parallelism of the model operation, the optimization is not satisfactory due to the large number of bases. Therefore, we propose a highly parallelized structure by introducing an array division method to divide the array into blocks, which can further unroll the loop and make each loop execute independently to improve the degree of program parallelization. In short, it is a method of exchanging space for time.

4 Software and hardware optimizations

The design idea and overall architecture of this work are shown in Figure 5, which consists of ARM, AXI interface, and custom IP core designed by Vivado HLS. In the IP core part, we mainly use the structure of the streaming pipeline to reduce Latency and thus improve the operation speed. As mentioned in the previous section, we use the AXI master interface provided by Xilinx for data transmission between PS and PL, and the prior distribution and sample data that are ready to participate

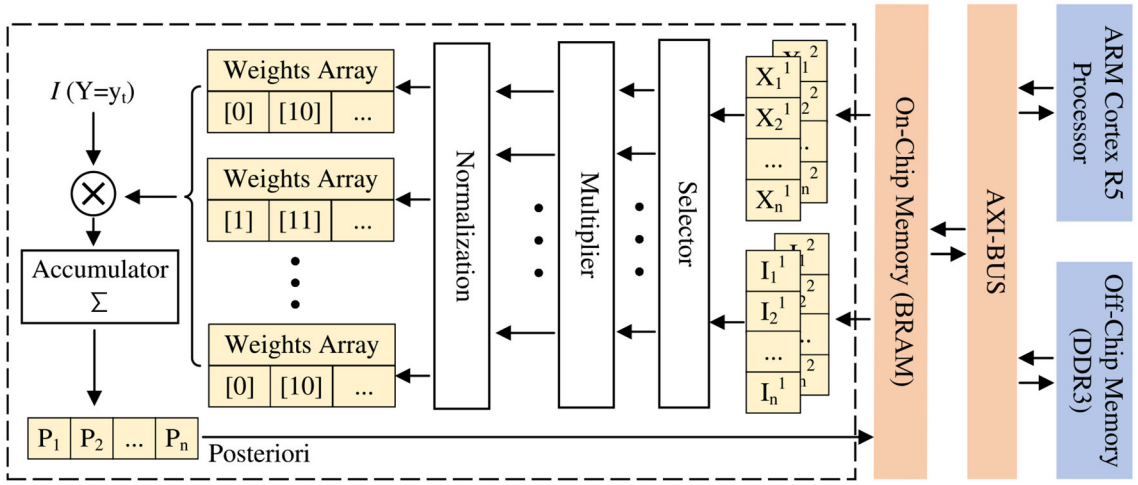


FIGURE 6
Design optimization ideas consisting of on-chip BRAM and processing elements (PE) using array division.

in inference will be allocated and stored in the on-chip BRAM. When the operation is finished, the result will also be returned to the off-chip DDR memory through the AXI master interface for subsequent processing.

In our work, we use the Vivado HLS tool provided by Xilinx to complete the design of the hardware IP core. This tool allows the synthesis of digital hardware directly using the high-level description developed in C/C++. With this tool we can convert C/C++ designs into RTL implementations for deployment on the FPGA, thereby significantly reducing the time required for FPGA development using traditional RTL descriptions. Therefore, the hardware architecture of the STM accelerator is designed by the programming language C++.

4.1 IP-core optimization

As mentioned in the last section, while adding the PIPELINE directive to the loop, we also use the method of array division to further improve the parallelism of the operation.

Here we take the sum of arrays as an example to illustrate how to improve parallelism. Under normal circumstances, the summation of an array is to iterate through each element of the array and accumulate them in turn. But even if we use the pipeline structure here, the accumulated value needs to be continuously read and written during the accumulation process. To prevent the emergence of dirty data, which leads to a time gap between the two loops, thus slowing down the speed of operation. In contrast, after we divide the original large-scale array into 10 blocks through array division, the subscripts of the array elements are accumulated every 10. In this way, the two adjacent loops in the accumulation process do not read and write to the same memory, thereby eliminating the time interval that would normally occur, to achieve the degree of parallelization of accumulation, as shown in Figure 6.

Finally, adding all blocks is the result of the array summation. The purpose of the manual expansion is to avoid memory access

TABLE 1 Comparison of resource consumption and Latency between the normal and the case using array division.

	BRAM	DSP	FF	LUT	Latency
Normal	14	172	25,934	38,817	11,170
Array division	14	179	28,142	43,849	6,698

bottlenecks and increase the degree of parallelism while using DSP as much as possible. Table 1 is based on the Bayesian network model shown in Figure 3A. In the case of setting 1,000 neurons in each node, the resource consumption and latency of not using array segmentation and using array segmentation are compared. It can be seen that the resource consumption increases slightly with array segmentation, but the Latency decreases significantly.

In addition, to further reduce resource utilization and improve performance, we use a bit-width of 32 bits for each operation through a simple quantization of floating-point operations. This kind of quantization has a relatively low negative impact on accuracy and can improve the performance of each IP core without reducing the parameters and input accuracy. At the same time, to alleviate the problem of the maximum frequency increase caused by reusing the same hardware components, especially BRAM resources, we added input and output registers to each BRAM instance to meet the 10 ns clock cycle of each IP core. Algorithm 1 shows the pseudocode of the IP core design. By default, all nested loops are executed sequentially. During this process, Vivado HLS provides different pragmas to affect scheduling and resource allocation.

4.2 Interface signal control

When we compile the PL-side custom core, we need to set up the top-level file containing the form parameters and return values. These parameters are mapped to the hardware circuitry to generate

Require: Get sample data and prior distribution $b1$, $b2$, $b3$, a.
Ensure: Posterior probability $post$.
1. Calculate the likelihood distribution based on sample data and prior probabilities.
for i in $NumA$ **do** {Likelihood loop1}
for j in $NumB$ **do** {Likelihood loop2}
$la \leftarrow b1, b2, b3, a$
end for
end for
2. Summation by array division.
3. Calculate the Posterior probability $post$ based on Eq. (2).
for i in $NumA$ **do** {Posterior loop1}
for j in $NumB$ **do** {Posterior loop2}
$post \leftarrow la, sum(la)$
end for
end for
4. Return calculation result.

Algorithm 1. IP-core design in pseudo-code.

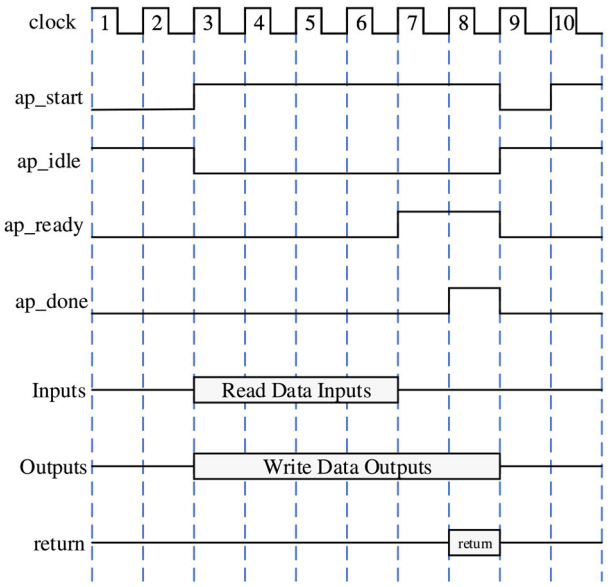


FIGURE 7
Timing diagram of ap_ctrl_hs four handshake signal functions. We mainly use ap_start interface to send read data commands to the FPGA, and detect ap_dong interface in real-time to determine whether the FPGA has completed the work.

interface signals, which can be controlled to not only help set better constraints but also to better control the input and output data flow according to the port timing. In addition, control logic needs to be extracted to form a state machine, so some handshake signals such as ap_start and ap_done will be formed.

Common interface constraints can be divided into Block-Level Protocols and Port-Level Protocols. Here we mainly use the ap_ctrl_hs signal in Block-Level Protocols, which contains four handshake signals ap_start, ap_idle, ap_ready, and ap_done. The ap_start signal is active high and indicates when the design starts working. The ap_idle signal is active low and indicates whether the design is idle. The ap_ready signal indicates whether the design is currently ready to receive new inputs. The ap_done signal indicates when the data on the output signal line is valid. The specific functional timing diagram is shown in Figure 7.

According to the timing diagram, we only need to pull the ap_start signal high and the design will automatically read or write data through the AXI bus while performing the inference operation. When the ap_done signal is read high, it means that the design has been completed, and the valid operation result can be obtained by reading the memory allocated for return.

4.3 Hardware–software streaming architecture

After the IP core has been designed, it is added to the Zynq block design to create the complete hardware architecture, as shown in Figure 8. The axi_interconnection module ensures communication between the IP core, PS system, and AXI interface. The axi_intc module controls the communication interruption of the interface.

Following the initialization of the design, the PS part will be used to implement the bitstream loading of the SNN. It also allows the PS to pass the values of external stimuli and SNN synaptic strengths to the PL part at runtime, which implements the specific neural network model. The main interface is used to connect the PL and PS parts of the SoC to ensure high-performance communication and data exchange between the IP-core and the PS in the streaming architecture. At the same time, the interlayer pipeline inside each IP-core is highly customized to build a Co-design with reset and GPIO. Both external stimulus values and synaptic strength values are stored in the cache of the BRAM in the PL part to improve the data reading speed for STM inference.

5 Simulations

We use the Intel i7-10700 and i5-12500, two of the more capable CPUs currently available, as benchmarks to compare the performance of model inference implemented on FPGAs. We test the performance and accuracy of the STM on the FPGA board for Bayesian inference on two brain perception problems: causal inference and multisensory integration. The evaluation metrics include inference effectiveness and processing speed on the model. In terms of inference effectiveness, causal inference is evaluated by the error rate varies with sample size, and multisensory integration is evaluated by comparison of the inference results and theoretical value.

5.1 Causal inference

Causal inference is the process by which the brain infers the causal effect between cause and outcomes when it receives

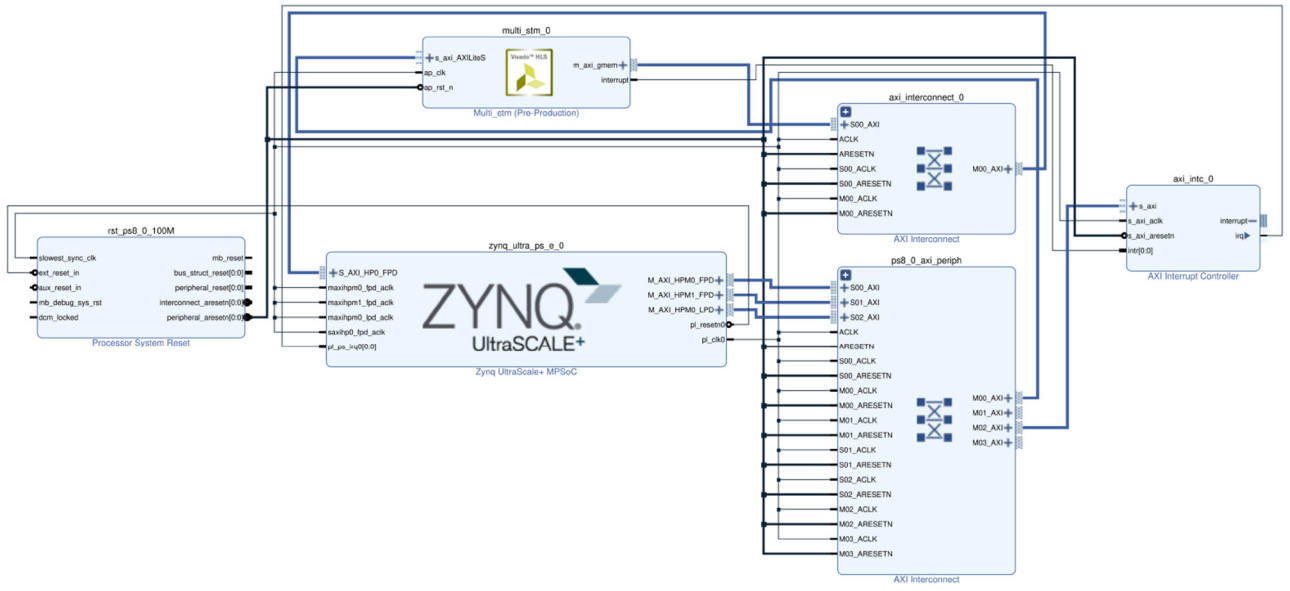


FIGURE 8
Hardware streaming architecture block design targeting the Soc with the m_axi interface between the PL and PS.

external information (Shams and Beierholm, 2010). The core problem of causal inference is to calculate the probability of the cause, which can be expressed as the expectation value defined on the posterior distribution. The calculation of the posterior probability is converted into the calculation of the prior probability and the likelihood probability through importance sampling, to realize the simulation of the causal inference process in the brain. In this experiment, we verify the accuracy and efficiency of Bayesian inference in the STM on the Xilinx ZCU104 FPGA board because probabilistic sampling on SNNs involves a large number of probabilistic calculations that can consume a lot of time, and the processing of the data in the inference process involves many computation-intensive operations, and the CPU is not able to handle these tasks very quickly.

In this paper, the validity of the model is verified from the accuracy of inference when different samples are input, and the STM is modeled by the Bayesian network shown in Figure 9A. Where B_1, B_2, B_3, and B_4 represent the input stimulus in causal inference and A denotes the cause. The tuning curve of each spiking neuron can be represented as the state of the variable. We suppose that the prior and conditional distributions are known, the distributions of these spiking neurons follow the prior distribution $P(B_1, B_2, B_3, B_4)$, and the tuning curve of the neuron i is proportional to the likelihood distribution $P(B_1^i, B_2^i, B_3^i, B_4^i|A)$. We can then normalize the output of Poisson spiking neurons through shunt inhibition and synaptic inhibition. Here we use y_i to denote the individual firing rate of the spiking neuron i and Y to denote the overall firing rate, and then:

$$E(y_i/Y = n) = \frac{P(B_1^i, B_2^i, B_3^i, B_4^i|A)}{\sum_i P(B_1^i, B_2^i, B_3^i, B_4^i|A)}. \quad (4)$$

By multiplying and linearly combining the normalized results with the synaptic weights, the posterior probability can be calculated:

$$P(A = a|B_1, B_2, B_3, B_4) = \sum_l I(A^l = a) \sum_i \frac{P(B_1^i, B_2^i, B_3^i, B_4^i|A^l)}{\sum_l P(B_1^i, B_2^i, B_3^i, B_4^i|A^l)}. \quad (5)$$

The results of the accuracy test are shown in Figure 9B. The error rate of the stimulus estimation keeps decreasing as the sample size increases, and when there are 2,000 sampled neurons, the error rate of stimulus estimation is already quite small. In addition, the inference accuracy of the implementation on the FPGA is similar to that on the PC. Therefore, the STM we run on the FPGA board can guarantee the accuracy of inference.

In terms of performance, we compare the design with multithreading and multiprogramming implementations on traditional computing platforms, and the results are shown in Table 2. It shows the processing time for each neuron sampling when the number of sampled neurons is 4,000. It can be seen from the results that multithreading and multiprogramming do not achieve the desired speedup but have the opposite effect. The possible reasons for this situation have been analyzed as follows: (1) Multithreaded execution is not strictly parallel, and global interpreter locks (GILs) can prevent parallel execution of multiple threads, so it may not be possible to take full advantage of multicore CPUs; (2) In terms of multiprogramming, perhaps the problem did not reach a certain size, resulting in the process creation process taking longer than the runtime. In addition, communication between multiprocesses requires passing a large amount of sample data, which introduces some overhead. For the above reasons, we finally considered using vectorization operations to vectorize the

10.3389/fnins.2023.1291051

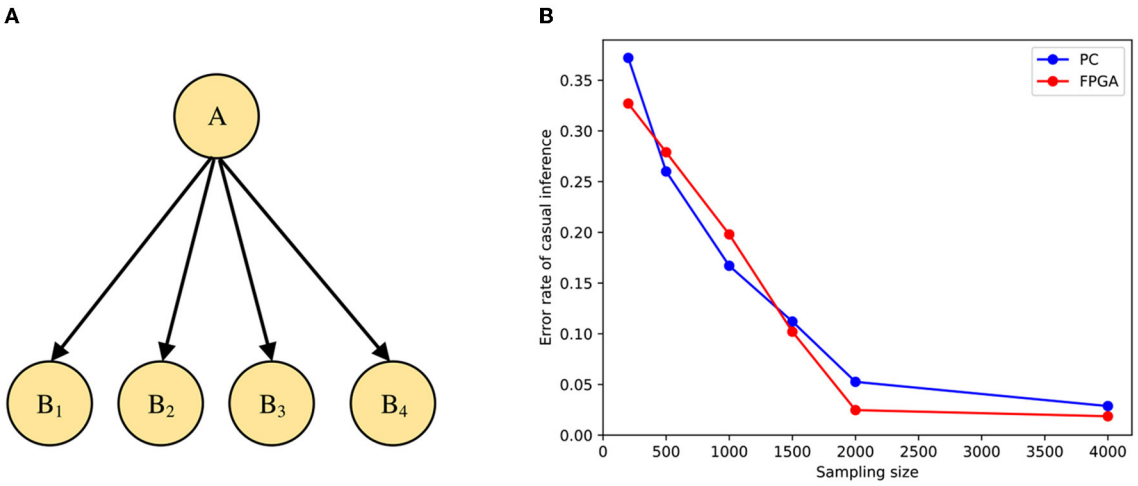


FIGURE 9
Simulation of causal inference. **(A)** The neural network architecture of the basic Bayesian network. **(B)** Comparison of error rates under PC and FPGA platforms.

TABLE 2 **Results of sampling time and speed-up of each neuron in the two-layer model.**

Processing time/neural (ms) \ Platform		Intel i7-10700 2.90 GHz	Intel i5-12500 2.50 GHz	ARM	Xilinx ZCU104
Normal		8.556	4.315	53.814	**0.389**
Multithreading	2	12.217	6.091		
	4	13.098	6.907		
	10	13.355	7.578		
	20	13.778	8.386		
	50	16.085	10.631		
	100	20.323	14.772		
Multiprogramming	2	344.00	250.88		
	4	394.43	278.46		
	8	564.03	454.81		
	16	948.47	844.73		
Vectorization		**3.662**	**2.993**		

Bold values represents the optimal time on the corresponding platform.

sample data to reduce the number of loops and avoid the speed limitations caused by nested loops.

From Table, we can see that vectorization is significantly faster than serial execution, multithreading, and multiprogramming, while the processing speed of the model on the FPGA is significantly better than that of the PC.

5.2 Causal inference with multi-layer neural network

The simulation in the previous section verified the causal inference under a simple model. The inference speed on the CPU decreases exponentially as the problem size increases when the need to shorten the inference time on the network model through improvements and optimizations becomes even more important. In this section, we will use a multi-layer neural network model to test large-scale Bayesian inference based on the sampling tree on the FPGA board. The STM is modeled by the Bayesian network shown in Figure 10A, where I_1, I_2 and I_3 denote the input stimuli in causal inference, A denotes the cause, and the rest are intermediate variables.

In this simulation, we use several spiking neurons to encode variables C_1, C_2, and C_3 respectively, and the distribution of these neurons follows the prior distribution $P(C_1, C_2)$ and $P(C_3)$. In addition, the tuning curves of these neurons are proportional to

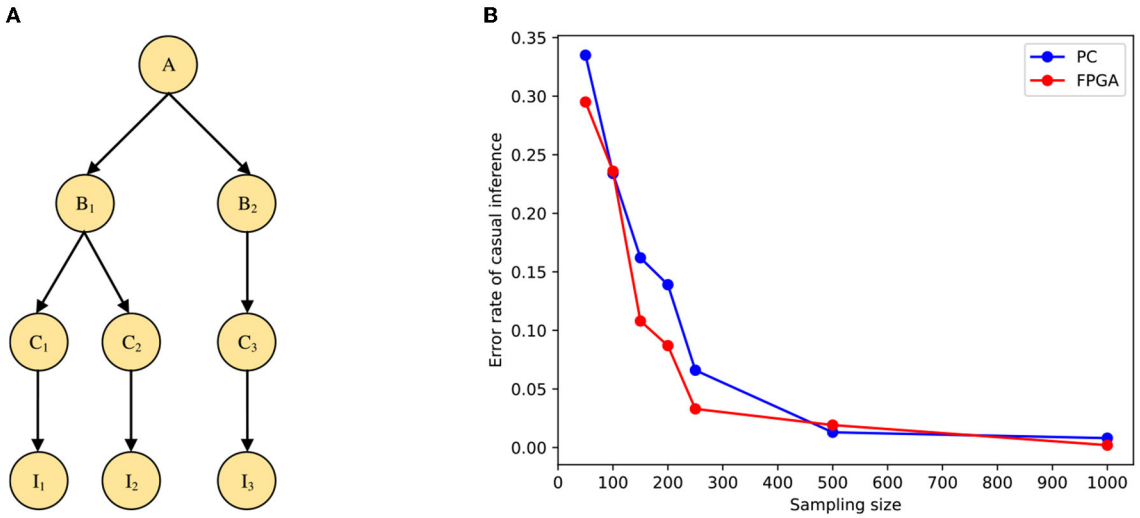


FIGURE 10
Simulation of causal inference with a multi-layer neural network. **(A)** The Bayesian model for multi-layer network structure. **(B)** Comparison of error rates under PC and FPGA platforms.

the distribution $P(I_1, I_2|C_1^i, C_2^i)$ and $P(I_3|C_3^j)$. We can obtain the average firing rates of spiking neurons C_1^i, C_2^i, and C_3^j, respectively:

$$E(C_1^i, C_2^i) = \frac{P(I_1, I_2|C_1^i, C_2^i)}{\sum_i P(I_1, I_2|C_1^i, C_2^i)}, \tag{6}$$

$$E(C_3^j) = \frac{P(I_3|C_3^j)}{\sum_j P(I_3|C_3^j)}. \tag{7}$$

The firing rate calculation of neurons in other layers is similar to this layer. The firing rate of each layer is multiplied and fed back to the next layer in the form of synaptic weights, and then the posterior probability can be calculated:

$$P(A = a|I_1, I_2, I_3) = \sum_l I(A^l = a) \sum_k \frac{P(B_1^k, B_2^k|A^l)}{\sum_l P(B_1^k, B_2^k|A^l)} \sum_{i,j} \frac{P(C_1^i, C_2^i, C_3^j|B_1^k, B_2^k)}{\sum_k P(C_1^i, C_2^i, C_3^j|B_1^k, B_2^k)} \frac{P(I_1, I_2|C_1^i, C_2^i)P(I_3|C_3^j)}{\sum_i P(I_1, I_2|C_1^i, C_2^i) \sum_j P(I_3|C_3^j)} \tag{8}$$

Similar to the simple model, the result of the STM under the multi-layer neural network on the FPGA is shown in Figure 10B. From the figure, we can see that the model running on the FPGA can guarantee the accuracy of the inference. Moreover, the performance comparison is shown in Table 3, in the multilayer network model, multithreading and multiprogramming are equally limited to achieve the desired results, so the same vectorization operation is used to optimize the program. We can also see the processing speed of the STM on FPGA is also improved compared with the traditional computing platform. In addition, we can find that due to the increase in the problem size of the multi-layer model, the acceleration of the model implemented on FPGA is more pronounced than in the two-layer model, even more than doubling.

5.3 Multisensory integration

In our daily life, we will obtain information from the outside world from the sense such as vision, hearing, and tough simultaneously, and the human brain can integrate this sensory information in the optimal way to get detailed information about an external object (Wozny et al., 2008). Some experiments have proved that the linear combination of different neuronal population activities with probabilistic population coding corresponds to the process of multisensory integration (Ma et al., 2006). Here, to demonstrate that our design can be generalized to other cognitive problems, we show that the STM on the FPGA board can solve multisensory integration problems with high performance and accuracy, and the final results can demonstrate that this work achieves good performance in the multisensory integration problem as well.

The simulation first considers the visual-auditory-haptic integration problem, and the STM is modeled by the Bayesian network shown in Figure 11A. Here S denotes the position of the object stimulus, S_V, S_H, and S_A denote visual, auditory, and haptic cues, respectively. We suppose that $P(S)$ is a uniform distribution, $P(S_V|S)$, $P(S_H|S)$, and $P(S_A|S)$ are three Gaussian distributions, respectively. When given S_V, S_H, and S_A, we can use importance sampling to infer the posterior probability of S, as:

$$P(S = s|S_V, S_H, S_A) = \sum_S I(S = s)P(S|S_V, S_H, S_A) = \sum iI(S_i = s)\frac{P(S_V, S_H, S_A|S_i)}{\sum iP(S_V, S_H, S_A|S_i)}, \; S_i \sim P(s). \tag{9}$$

10.3389/fnins.2023.1291051

TABLE 3 Results of sampling time and speed-up of each neuron in the multi-layer model.

Processing time/neural (ms) \ Platform		Intel i7-10700 2.90 GHz	Intel i5-12500 2.50 GHz	ARM	Xilinx ZCU104
Normal		1.103	0.635	12.75	**0.024**
Multithreading	2	1.048	0.622		
	4	1.019	0.617		
	10	1.006	0.617		
	20	1.012	0.618		
	50	1.012	0.618		
	100	1.013	0.624		
Multiprogramming	2	1.174	0.749		
	4	1.056	0.706		
	8	1.113	0.762		
	16	1.371	1.097		
Vectorization		**0.569**	**0.403**		

Bold values represents the optimal time on the corresponding platform.

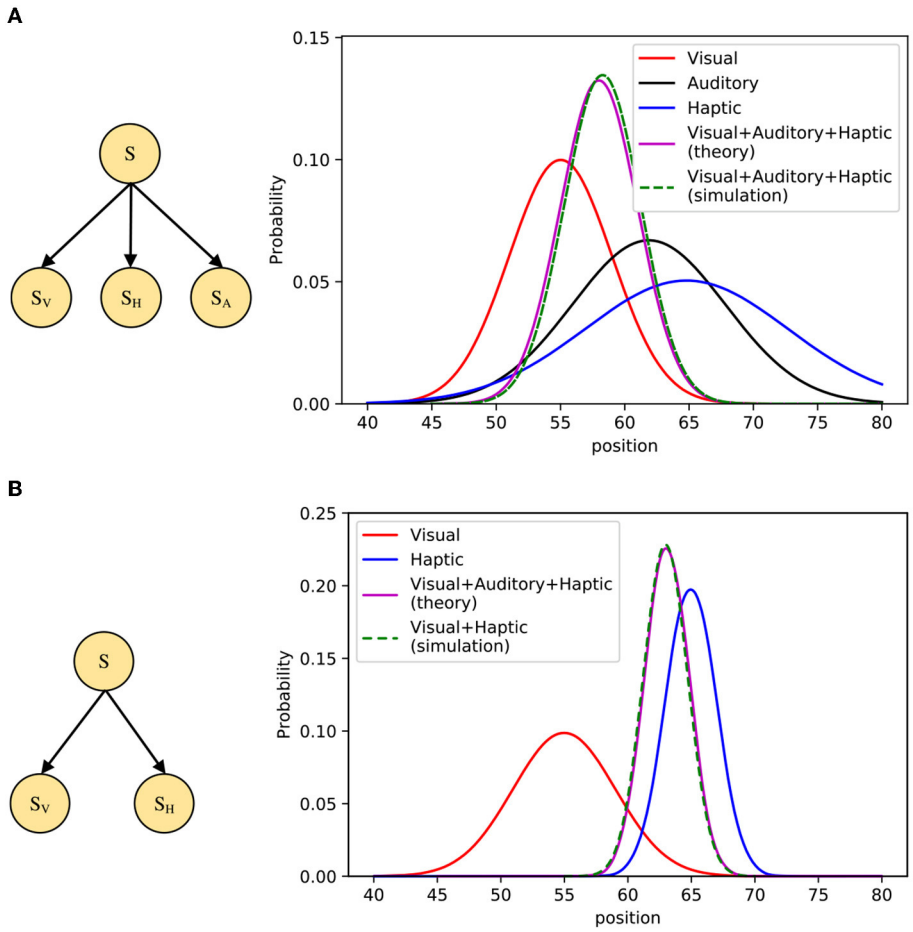


FIGURE 11
Simulation of multisensory integration. **(A)** Left: The Bayesian model for visual-auditory-haptic integration, Right: Comparison of model inference results and theoretical values on FPGA. **(B)** Left: The Bayesian model for visual-haptic integration, Right: Comparison of model inference results and theoretical values on FPGA.

TABLE 4 Results of sampling time and speed-up of each neuron in the simulation of multisensory integration.

Platform / Processing time/neural (ms)	Intel i7-10700 2.90 GHz	Intel i5-12500 2.50 GHz	ARM	Xilinx ZCU104
Normal	7.632	5.169	94.608	**0.328**
Vectorization	**3.882**	**2.160**		

Bold values represents the optimal time on the corresponding platform.

In our simulation, multisensory integration inference is achieved through neural circuits based on PPC and normalization. We use 1,000 spiking neurons to encode stimuli whose states follow the prior distribution $P(S)$. We suppose that the tuning curve of the neuron i is proportional to the distribution $P(S_V, S_H, S_A|S_i)$, and then use shunting inhibition and synaptic depression to make the output of spiking neurons normalized, the result will be fed into the next spiking neuron with synaptic weights $I(S_i = s)$. Figure 11A shows the simulation results, where the inference result obtained from the STM on the FPGA board is in good agreement with the theoretical values. Similar to the visual-auditory-haptic integration, we also add a simulation of visual-haptic integration to improve the completeness, which is illustrated in Figure 11B. Furthermore, the performance comparison is shown in Table 4, which shows a significant improvement in the sampling speed of each neuron on the FPGA. Since the results of multi-threading and multi-process experiments were not ideal in previous experiments, only vectorization methods are compared here. The results also show that the running speed on FPGA is still better than that on CPU.

6 Conclusion

In this work, we design an FPGA-based hardware accelerator for PGM-based SNNs with the help of the PYNQ framework. Firstly, the STM, as a novel SNN simulation model for causal inference, can convert a global complex inference problem into a local simple inference problem, thus realizing high-precision approximate inference. Furthermore, as a generalized neural network model, the STM does not formulate a neural network for a specific task and thus can be generalized to other problems. Our hardware implementation is based on this solid and innovative theoretical model, which solves the problem of slow model computation based on its realization of large-scale multi-layer complex model inference.

Secondly, As the first work to realize the hardware acceleration of the STM, we chose the FPGA platform as the acceleration platform of the model. For CPUs and GPUs, both of them need to go through operations such as fetching instructions, decoding, and various branch logic jumps, and the energy consumption of GPUs is too high. In contrast, the function of each logic unit of an FPGA is determined at the time of reprogramming and does not require these instruction operations, so FPGAs can enjoy lower latency and energy consumption. Compared to hardware platform ASICs, FPGAs are more flexible. Although ASICs are superior to FPGAs in terms of throughput, latency, and power consumption, their high cost and long cycle time cannot be ignored, and the design of an ASIC cannot be easily changed once it is completed. In contrast, FPGAs are programmable hardware that can be changed at any time according to demand without having to remanufacture the hardware, and this flexibility is the reason why we ultimately chose FPGAs. FPGA is a compromise between the above two platforms, although some aspects of the performance is not up to the two, but it is a combination of the advantages of the two. It also provides reasonable cost, low power consumption, and reconfigurability for neuromorphic computing acceleration.

Thirdly, The experimental results and data on causal inference validate our conclusion: in the two-layer model, we can then see that the inference accuracy of the implementation on the FPGA can approximate that of the implementation on the CPU, with an accuracy of up to 98%, and at the same time achieve a multifold speedup. The acceleration effect becomes more and more obvious as the problem size increases, which is proved in the multi-layer model, and from the results we can see that the acceleration effect in the multi-layer model is more than twice as much as that in the two-layer model. Moreover, in the experiments on multisensory integration, the experimental results also demonstrate that our design implementation can be used for other real-world cognitive problems while guaranteeing the accuracy of reasoning and the acceleration effect.

Finally, the hardware acceleration method proposed in the paper can simulate the working principle of biological neurons very well. Meanwhile, due to the characteristics of low power consumption and real-time response of FPGA, this method can have a wide range of applications in the embedded field. The realized causal inference problems can be used in policy evaluation, financial decision-making and other fields, and the multisensory integration can be used in vehicle environment perception, medical diagnosis and other fields. Specifically, in application scenarios such as smart home application environments, causal inference can be used to achieve reasoning about factors affecting health and provide personalized health advice. Sensory cues such as vision and hearing are combined to provide a better perceive the home environment and thus provide intelligent control. Our work provides a solution for such application scenarios and these practical applications are expected to promote the progress of the neuromorphic computing field and make it better meet the practical application requirements. In addition, so far the STM does not consider learning, which is an important aspect of adaptation between tasks. All the results of our simulations are based on inference with known prior probabilities and conditional probabilities. Therefore, in future work, we need to combine learning and inference into one framework and introduce some learning mechanisms to make the model more complete and flexible for multiple tasks.

Data availability statement

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

Author contributions

HL: Methodology, Data curation, Investigation, Software, Validation, Writing – original draft. BW: Methodology, Conceptualization, Supervision, Writing – review & editing. QL: Methodology, Writing – review & editing, Project administration. YF: Methodology, Writing – review & editing, Conceptualization, Formal analysis, Software, Supervision, Writing – original draft. JL: Formal analysis, Supervision, Writing – review & editing, Project administration. LA: Supervision, Writing – review & editing, Conceptualization, Methodology.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This work was partially supported by the National Natural Science Foundation of China (Grant No. 62072355), the Key Research and Development Program of Shaanxi Province of China (Grant No. 2022KWZ-10), the Natural Science Foundation of Guangdong Province of China (Grant No. 2022A1515011424), the Science and Technology Planning Project of Guangdong Province of China (Grant No. 2023A0505050126), the Outstanding Scientist Cultivation Program of Beijing Academy of Agriculture and Forestry Sciences (Grant No. JKZX202214), the Natural Science Foundation of Fujian Province of China (Grant No. 2022J01656), the Foundation of National Key Laboratory of Human Factors Engineering (Grant No. 6142222210101), and the Key Industry Innovation Chain Projects of Shaanxi, China (Grant No. 2021ZDLGY07-04).

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

References

Alais, D., and Burr, D. (2019). "Cue combination within a Bayesian framework," in *Multisensory Processes* (New York, NY: Springer), 9–31. doi: 10.1007/978-3-030-10461-0_2

Awano, H., and Hashimoto, M. (2020). "BYNQNET: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA," in *2020 Design, Automation and Test in Europe Conference and Exhibition* (Grenoble: IEEE), 1402–1407. doi: 10.23919/DATE48585.2020.9116302

Awano, H., and Hashimoto, M. (2023). B2N2: resource efficient Bayesian neural network accelerator using Bernoulli sampler on FPGA. *Integration* 89, 1–8. doi: 10.1016/j.vlsi.2022.11.005

Bialek, W., Rieke, F., van Steveninck, R., and Warland, D. (1999). *Spikes: Exploring the Neural Code (Computational Neuroscience)*. Cambridge, MA: The MIT Press.

Buesing, L., Bill, J., Nessler, B., and Maass, W. (2011). Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Comput. Biol.* 7, 188–200. doi: 10.1371/journal.pcbi.1002211

Cai, R., Ren, A., Liu, N., Ding, C., Wang, L., Qian, X., et al. (2018). VIBNN: hardware acceleration of Bayesian neural networks. *ACM SIGPLAN Notices* 53, 476–488. doi: 10.1145/3296957.3173212

Chandrasekaran, C. (2017). Computational principles and models of multisensory integration. *Curr. Opin. Neurobiol.* 43, 25–34. doi: 10.1016/j.conb.2016.11.002

Christensen, D. V., Dittmann, R., Linares-Barranco, B., Sebastian, A., et al. (2022). 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Comput. Eng.* 2, 022501. doi: 10.1088/2634-4386/ac4a83

Demis, H., Dharshan, K., Christopher, S., and Matthew, B. (2017). Neuroscience-inspired artificial intelligence. *Neuron* 95, 245–258. doi: 10.1016/j.neuron.2017.06.011

Ernst, M. O., and Banks, M. S. (2002). Humans integrate visual and haptic information in a statistically optimal fashion. *Nature* 415, 429–433. doi: 10.1038/415429a

Fan, H., Ferianc, M., Que, Z., Liu, S., Niu, X., Rodrigues, M. R., et al. (2022). FPGA-based acceleration for Bayesian convolutional neural networks. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 41, 5343–5356. doi: 10.1109/TCAD.2022.3160948

Fan, H., Ferianc, M., Rodrigues, M., Zhou, H., Niu, X., Luk, W., et al. (2021). "High-performance FPGA-based accelerator for Bayesian neural networks," in *2021 58th ACM/IEEE Design Automation Conference* (San Francisco, CA: IEEE), 1063–1068. doi: 10.1109/DAC18074.2021.9586137

Fang, H., Mei, Z., Shrestha, A., Zhao, Z., Li, Y., Qiu, Q., et al. (2020). "Encoding, model, and architecture: systematic optimization for spiking neural network in FPGAs," in *Proceedings of the 39th International Conference on Computer-Aided Design* (IEEE), 1–9. doi: 10.1145/3400302.3415608

Fang, Y., Yu, Z., Liu, J. K., and Chen, F. (2019). A unified neural circuit of causal inference and multisensory integration. *Neurocomputing* 358, 355–368. doi: 10.1016/j.neucom.2019.05.067

Ferianc, M., Que, Z., Fan, H., Luk, W., and Rodrigues, M. (2021). "Optimizing Bayesian recurrent neural networks on an FPGA-based accelerator," in *2021 International Conference on Field-Programmable Technology* (Auckland: IEEE), 1–10. doi: 10.1109/ICFPT52863.2021.9609847

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2022). Event-based vision: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 154–180. doi: 10.1109/TPAMI.2020.3008413

George, D., and Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *PLoS Comput. Biol.* 5, e1000532. doi: 10.1371/journal.pcbi.1000532

Han, J., Li, Z., Zheng, W., and Zhang, Y. (2020). Hardware implementation of spiking neural networks on FPGA. *Tsinghua Sci. Technol.* 25, 479–486. doi: 10.26599/TST.2019.9010019

Ju, X., Fang, B., Yan, R., Xu, X., and Tang, H. (2020). An FPGA implementation of deep spiking neural networks for low-power and fast classification. *Neural Comput.* 32, 182–204. doi: 10.1162/neco_a_01245

Kim, J., Koo, J., Kim, T., and Kim, J.-J. (2018). Efficient synapse memory structure for reconfigurable digital neuromorphic hardware. *Front. Neurosci.* 12, 829. doi: 10.3389/fnins.2018.00829

Körding, K. P., and Wolpert, D. M. (2004). Bayesian integration in sensorimotor learning. *Nature* 427, 244–247. doi: 10.1038/nature02169

Liu, K., Cui, X., Zhong, Y., Kuang, Y., Wang, Y., Tang, H., et al. (2019). A hardware implementation of SNN-based spatio-temporal memory model. *Front. Neurosci.* 13, 835. doi: 10.3389/fnins.2019.00835

Liu, L., Wang, D., Wang, Y., Lansner, A., Hemani, A., Yang, Y., et al. (2020). A "FPGA-based hardware accelerator for Bayesian confidence propagation neural network," in *2020 IEEE Nordic Circuits and Systems Conference* (Oslo: IEEE), 1–6. doi: 10.1109/NorCAS51424.2020.9265129

Ma, D., Shen, J., Gu, Z., Zhang, M., Zhu, X., Xu, X., et al. (2017). Darwin: a neuromorphic hardware co-processor based on spiking neural networks. *J. Syst. Archit.* 77, 43–51. doi: 10.1016/j.sysarc.2017.01.003

Ma, W., Beck, J. M., Latham, P. E., and Pouget, A. (2006). Bayesian inference with probabilistic population codes. *Nat. Neurosci.* 9, 1432–1438. doi: 10.1038/nn1790

Ma, W., and Jazayeri, M. (2014). Neural coding of uncertainty and probability. *Ann. Rev. Neurosci.* 37, 205–220. doi: 10.1146/annurev-neuro-071013-014017

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Nagata, K., and Watanabe, S. (2008). Exchange Monte Carlo sampling from Bayesian posterior for singular learning machines. *IEEE Trans. Neural Netw.* 19, 1253–1266. doi: 10.1109/TNN.2008.2000202

Que, Z., Nakahara, H., Fan, H., Li, H., Meng, J., Tsoi, K. H., et al. (2022). "Remarn: a reconfigurable multi-threaded multi-core accelerator for recurrent neural networks," in *ACM Transactions on Reconfigurable Technology and Systems* (New York, NY: ACM). doi: 10.1145/3534969

Shams, L., and Beierholm, U. R. (2010). Causal inference in perception. *Trends Cogn. Sci.* 14, 425–432. doi: 10.1016/j.tics.2010.07.001

Shen, J., Liu, J. K., and Wang, Y. (2021). Dynamic spatiotemporal pattern recognition with recurrent spiking neural network. *Neural Comput.* 33, 2971–2995. doi: 10.1162/neco_a_01432

Shi, L., and Griffiths, T. (2009). Neural implementation of hierarchical Bayesian inference by importance sampling. *Adv. Neural. Inf. Process Syst.* 22.

Shi, Z., Church, R. M., and Meck, W. H. (2013). Bayesian optimization of time perception. *Trends Cogn. Sci.* 17, 556–564. doi: 10.1016/j.tics.2013.09.009

Tung, C., Hou, K.-W., and Wu, C.-W. (2023). "A built-in self-calibration scheme for memristor-based spiking neural networks," in *2023 International VLSI Symposium on Technology, Systems and Applications (VLSI-TSA/VLSI-DAT)* (HsinChu: IEEE), 1–4. doi: 10.1109/VLSI-TSA/VLSI-DAT57221.2023.10134261

Tzanos, G., Kachris, C., and Soudris, D. (2019). "Hardware acceleration on gaussian naive bayes machine learning algorithm," in *2019 8th International Conference on Modern Circuits and Systems Technologies* (Thessaloniki: IEEE), 1–5. doi: 10.1109/MOCAST.2019.8741875

Wang, D. (2022). *Design and Implementation of FPGA-based Hardware Accelerator for Bayesian Confidence* [Master's Thesis]. Turku: The University of Turku Quality.

Wang, D., Xu, J., Li, F., Zhang, L., Cao, C., Stathis, D., et al. (2023). A memristor-based learning engine for synaptic trace-based online learning. *IEEE Trans. Biomed. Circuits Syst.* 17, 1153–1165. doi: 10.1109/TBCAS.2023.3291021

Wozny, D. R., Beierholm, U. R., and Shams, L. (2008). Human trimodal perception follows optimal statistical inference. *J. Vis.* 8, 24. doi: 10.1167/8.3.24

Xu, Q., Shen, J., Ran, X., Tang, H., Pan, G., Liu, J. K., et al. (2022). Robust transcoding sensory information with neural spikes. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1935–1946. doi: 10.1109/TNNLS.2021.3107449

Yang, Z., Guo, S., Fang, Y., and Liu, J. K. (2022). "Biologically plausible variational policy gradient with spiking recurrent winner-take-all networks," in *33rd British Machine Vision Conference 2022* (London: BMVA Press), 21–24.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Trans. Inf. Theory* 51, 2282–2312. doi: 10.1109/TIT.2005.850085

Yu, Z., Chen, F., and Liu, J. K. (2019). Sampling-tree model: efficient implementation of distributed Bayesian inference in neural networks. *IEEE Trans. Cogn. Dev. Syst.* 12, 497–510. doi: 10.1109/TCDS.2019.2927808

Yu, Z., Deng, F., Guo, S., Yan, Q., Liu, J. K., Chen, F., et al. (2018a). Emergent inference of hidden Markov models in spiking winner-take-all neural networks. *IEEE Trans. Cybern.* 50, 1347–1354. doi: 10.1109/TCYB.2018.2871144

Yu, Z., Liu, J. K., Jia, S., Zhang, Y., Zheng, Y., Tian, Y., et al. (2020). Toward the next generation of retinal neuroprosthesis: visual computation with spikes. *Engineering* 6, 449–461. doi: 10.1016/j.eng.2020.02.004

Yu, Z., Tian, Y., Huang, T., and Liu, J. K. (2018b). Winner-take-all as basic probabilistic inference unit of neuronal circuits. *arXiv* [preprint]. 10.48550/arXiv.1808.00675

Zador, A., Escola, S., Richards, B., Ölveczky, B., Bengio, Y., Boahen, K., et al. (2022). Toward next-generation artificial intelligence: catalyzing the NeuroAI revolution. *arXiv* [preprint]. doi: 10.1038/s41467-023-37180-x

Zhang, Y., Jia, S., Zheng, Y., Yu, Z., Tian, Y., Ma, S., et al. (2020). Reconstruction of natural visual scenes from neural spikes with deep neural networks. *Neural Netw.* 125, 19–30. doi: 10.1016/j.neunet.2020.01.033

Zhu, Y., Zhang, Y., Xie, X., and Huang, T. (2022). An FPGA accelerator for high-speed moving objects detection and tracking with a spike camera. *Neural Comput.* 34, 1812–1839. doi: 10.1162/neco_a_01507

frontiers | Frontiers in Neuroscience

TYPE Original Research
PUBLISHED 24 January 2024
DOI 10.3389/fnins.2023.1253830

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Zihan Pan,
Institute for Infocomm Research (A*STAR), Singapore
Pengfei Sun,
Ghent University, Belgium

*CORRESPONDENCE
Jing Wang
✉ wangjing2012@uestc.edu.cn

RECEIVED 06 July 2023
ACCEPTED 04 December 2023
PUBLISHED 24 January 2024

CITATION
Wang J (2024) Training multi-layer spiking neural networks with plastic synaptic weights and delays. *Front. Neurosci.* 17:1253830. doi: 10.3389/fnins.2023.1253830

Training multi-layer spiking neural networks with plastic synaptic weights and delays

Jing Wang*

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Spiking neural networks are usually considered as the third generation of neural networks, which hold the potential of ultra-low power consumption on corresponding hardware platforms and are very suitable for temporal information processing. However, how to efficiently train the spiking neural networks remains an open question, and most existing learning methods only consider the plasticity of synaptic weights. In this paper, we proposed a new supervised learning algorithm for multiple-layer spiking neural networks based on the typical SpikeProp method. In the proposed method, both the synaptic weights and delays are considered as adjustable parameters to improve both the biological plausibility and the learning performance. In addition, the proposed method inherits the advantages of SpikeProp, which can make full use of the temporal information of spikes. Various experiments are conducted to verify the performance of the proposed method, and the results demonstrate that the proposed method achieves a competitive learning performance compared with the existing related works. Finally, the differences between the proposed method and the existing mainstream multi-layer training algorithms are discussed.

KEYWORDS

spiking neural networks, supervised learning, synaptic weights, synaptic delays, SpikeProp

1 Introduction

Deep neural network (DNNs), as a mainstream algorithm of machine learning, has been applied to various fields, such as compute vision (He et al., 2016), speech separation (Subakan et al., 2021), path finding (Arulkumaran et al., 2017), etc. However, the current DNNs suffer from the problem of excessive power consumption, which limit their applications in energy-critical environments (Zhang M. et al., 2021). In contrast, the nervous systems in biological brains require very little energy to handle complex tasks. As a combination of both, the spiking neural networks (SNNs) (Maass, 1997) inherit the existing mature structures and algorithms in the DNNs, and further learn from the way of using spike trains to transmit information between biological neurons. As a result, SNNs have more complex spatial-temporal dynamics and are suitable for ultra-low power devices (Lan et al., 2021; Pan et al., 2021; Zhang et al., 2022). However, at present, there is no training algorithm for SNNs that can give full play to their characteristics, so how to efficiently train SNNs is still an open question.

The current training algorithms for spiking neural networks include training the connection weights between neurons in the network and the delay in the transmission of spike trains between neurons. The first type of training algorithm is consistent with the goal of deep neural networks, which is to enable spiking neural networks to effectively complete tasks by training neural network weights. The training method can be divided into heuristic algorithms, conversion-based algorithms and BP-based algorithms. The

representative algorithms of heuristic algorithms are STDP learning algorithms (Caporale and Dan, 2008) and its variants (Yha et al., 2020; Wu et al., 2021c). This class of algorithms is largely based on biological neuroscience findings that when neurons fire together, wire together. The second methods are conversion-based methods (Wu et al., 2021a,b) and their basic idea is to first train a DNN, and then convert the parameters in the trained network to the corresponding SNN through a series of methods. Compared with other state-of-the-art SNN implementations, the inference time and total synaptic operations of the network trained by this method are reduced by at least one order of magnitude. When the length of the simulation time is only eight-time steps, the conversion-based network still achieves good performance in large datasets. The third method is the method based on the surrogate gradient learning (Neftci et al., 2019; Zhu et al., 2021). Although this method can achieve competitive results with DNN in short time steps, this method needs to save the state information of the SNN at each moment. Therefore the computing power and storage requirements are large. The fourth is the event-driven training algorithm (Gütig, 2016; Neftci et al., 2016; Zhang M. et al., 2021; Luo et al., 2022), which only adjusts the network parameters according to the spikes, greatly reducing the training costs.

The above-mentioned learning algorithms for network weights are mostly used for processing static or periodic data and are not effective for fast time-varying signals. The reason is that simply adjusting the synaptic weights cannot effectively extract the rich time-dependent relationship between spike trains in SNNs, while in the biological nervous system, different synapses have various delays in transmitting spike trains (Zhang et al., 2020; Han et al., 2021). In order to further enhence the ability of the SNN model to process fast time-varying data, based on the original synaptic weight training, a training algorithm for the transmission delay between synapses is added.

However, there is biological evidence that the brain's biological synaptic latency is not a constant and there is no uniformity in the rules of latency variation (Sun et al., 2023), so this is also an open area of research. The DL-ReSuMe (Taherkhani et al., 2015a) algorithm is proposed to merge the delay shift approach and ReSuMe-based weight adjustment to enhance the learning performance. After that, Multi-DL-ReSuMe is proposed to train multiple neurons to classify spatiotemporal spiking patterns (Taherkhani et al., 2015b). Shrestha and Orchard (2018) proposes a general backpropagation mechanism for learning synaptic weights and axonal delays which overcomes the problem of non-differentiability of the spike function and uses a temporal credit assignment policy for backpropagating error to preceding layers. Sun et al. (2022) proposes the rectified axonal delay (RAD) as an additional degree of freedom for training that can easily be incorporated into existing SNN frameworks. The new model can perform well on problems where timing matters using very few parameters. DW-ReSuMe (Han et al., 2021) is proposed to achieve a spike train learning task, which is combined with delay learning based on weight training. The RL-Squares-Based Learning Rule (Zhang Y. et al., 2021) is proposed to generate the desired spatiotemporal spike train. The gradient descent-based synaptic delay learning algorithm (Luo et al., 2022) is proposed to improve the sequential learning performance of single-spike neurons.

Although these methods increase the model's ability to process time-series-related data, they need to face the problem of exploding or vanishing gradients in the training process. It is necessary to select the hyper-parameters of the model and algorithm very carefully to make the model converge effectively.

The contribution of this paper includes the following points:

1. This paper introduces synaptic delays between neurons based on the Spike Response Model (SRM) (Gerstner, 1995) model, and combines the SpikeProp (Bohte et al., 2002) algorithm to propose a new learning algorithm for training synaptic delays. This algorithm effectively increases the ability of SNN to deal with fast time-varying tasks.
2. This paper proposes a gradient replacement strategy to effectively reduce the impact of the gradient explosion problem in the training process of the SpikeProp algorithm.
3. In this paper, the proposed training algorithm is applied to several different data sets for testing. The experimental results show that the method presented in this paper effectively increases the ability of SNN to process fast time-varying data.

2 Materials and methods

In this section, the basic theoretical knowledge of the neuron model that we used in this paper will be introduced first. Then, we will further introduce the learning algorithm of the SpikeProp. Finally, the proposed learning algorithm is presented and the processing of algorithm derivation is given in detail.

2.1 Neuron model

Inspired by the biological brains, spiking neural networks (SNNs) (Maass and Bishop, 2001), which are often referred to as the third generation of artificial neural networks, employ a spike function to replicate the information transfer observed in biological neurons. SNNs possess the unique capacity for biological plasticity, allowing them to encode external inputs into spike trains (Izhikevich, 2003). When these spike trains are processed, they are reduced to two fundamental factors (Pfeiffer and Pfeil, 2018): (1) spike time, which involves the relative timing of pre-synaptic and post-synaptic spikes, and (2) synaptic type, encompassing attributes like excitatory or inhibitory properties and the strength of synaptic connections.

In SNNs, every neuron remains silent until it receives a spike. Once receiving incoming information, each neuron experiences changes in membrane voltage. Output spikes are generated only when the total membrane voltage surpasses the neuron's threshold θ, after which they propagate backward (Ghosh-Dastidar and Adeli, 2009). One widely utilized spiking neuron model is the spike response model [SRM (Gerstner, 1995)]. The membrane voltage in the SRM is calculated as shown in Equation 1:

$$V_j^{l+1}(t) = \sum_i w_{ij}^{l+1} \cdot K\left(t - t_i^l - d_i^{l+1}\right), \tag{1}$$

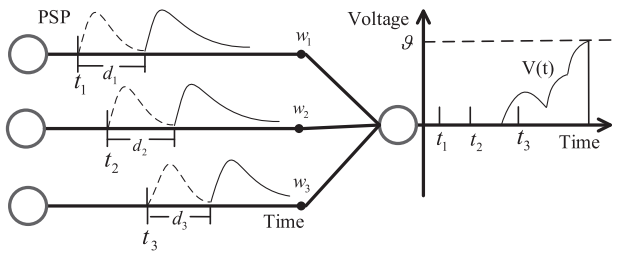


FIGURE 1
SRM neuron model with synaptic delay.

where V_j^{l+1} represents the membrane potential of the jth neuron in layer $l+1$, and the t_i^l is the firing time of the ith neuron in layer l. w_{ij}^{l+1} and d_{ij}^{l+1} are the synaptic efficacy and delay between these two neurons, respectively. The kernel $K(\cdot)$, which determines the shape of postsynaptic potentials (PSPs), is defined as Equation 2:

$$K(x) = V_{\text{norm}}\left[\exp\left(-\frac{x}{\tau_m}\right) - \exp\left(-\frac{x}{\tau_s}\right)\right], \quad x > 0, \tag{2}$$

where τ_m and τ_s represent the membrane time constant of neurons, respectively. V_{norm} is the result of PSPs being normalized, which makes the value of PSPs between 0 and 1 and is calculated by Equation 3:

$$V_{\text{norm}} = \frac{\beta^{\beta/(\beta-1)}}{\beta - 1}, \tag{3}$$

where $\beta = \tau_m/\tau_s$.

Figure 1 shows the SRM neuron model with synaptic delay. There are three pre-synaptic neurons with the weight w_i and the delay time d_i ($i = 1, 2, 3$). The delay time ensures the firing time at the synapse is delayed, which makes the membrane potential $V(t)$ change between t_2 and t_3.

2.2 Learning algorithm of the SpikeProp

The SpikeProp algorithm, an improvement upon the traditional backpropagation algorithm for artificial neural networks (ANN), is designed to facilitate the learning process of multi-layer feed-forward spiking neural networks (SNN) (Bohte et al., 2002). Notably, the algorithm imposes a constraint, allowing each neuron to fire at most once within each layer. The SpikeProp algorithm employs the time minimum mean square error function as its error function, which is illustrated in Equation 4:

$$E = \frac{1}{2}\sum_k \left(t_k^o - t_k^d\right)^2, \tag{4}$$

where t_k^o represents the time at which the output neuron k emits an actual spike, and t_k^d denotes its target spike time. SpikeProp utilizes the SRM model, which allows for the derivation of the relationship between firing time and membrane voltage through mathematical analysis. The weight update value is obtained by minimizing the mean square error, as defined in Equation 5:

$$\Delta w_{ij}^l = -\eta\frac{\partial E}{\partial w_{ij}^l} = -\eta\frac{\partial E}{\partial t_j^l}\frac{\partial t_j^l}{\partial V(t_j^l)}\frac{\partial V(t_j^l)}{\partial w_{ij}^l} = -\eta\cdot\delta_j^l K_{ij}^l, \tag{5}$$

where Δw_{ij}^l stands for the gradient of the synaptic weight between ith presynaptic neuron in layer $l-1$ and jth postsynaptic neuron in layer l. δ_j^l is the intermediate quantity for gradient calculation, which can be expressed as $\frac{\partial E}{\partial t_j^l}\frac{\partial t_j^l}{\partial V(t_j^l)}$, and $K_{ij}^l = \frac{\partial V(t_j^l)}{\partial w_{ij}^l} = K(t_j^l - t_i^{l-1} - d_{ij}^l)$ represents the unweighted postsynaptic potential. η is the learning rate. To simplify notation, we abbreviate V_j^l as V.

However, when the membrane voltage reaches the firing threshold, it promptly resets to the resting potential, resulting in a spike emission at that specific moment, rendering it non-differentiable. Consequently, the direct calculation of $\frac{\partial t_j}{\partial V(t_j^l)}$ within δ_j^l becomes unfeasible. To address this challenge, SpikeProp introduces the concept of a linear hypothesis. This entails assuming that within a sufficiently small neighborhood around $t = t_j^l$, the membrane potential can be reasonably approximated as a linear function of time, which is defined as:

$$\frac{\partial t_j^l}{\partial V(t_j^l)} = \frac{-1}{\partial V(t_j^l)/\partial t} = \frac{-1}{\sum_i w_{ij}^l\left(\partial K_{ij}^l(t_j^l)/\partial t\right)} \tag{6}$$

2.3 The proposed learning algorithm

SpikeProp primarily focuses on adjusting synaptic weights, which is an essential aspect of biological synaptic plasticity. However, it overlooks another crucial element, synaptic delay plasticity, which imposes limitations on its overall performance and diminishes its biological interpretability. Conversely, in addressing the non-differentiability issue resulting from spike discontinuity, SpikeProp resorts to the approach outlined in Equation 6. Unfortunately, this solution introduces another challenge: the possibility of a gradient explosion due to the rapid membrane voltage change near the firing threshold. In the subsequent sections, we present solutions to these two problems individually.

2.3.1 Learning algorithms with synaptic weights and delay plasticity

To maintain generality, let's assume that the network in question is a multi-layer fully connected network, with the final layer designated as the oth layer. Similar to the approach employed in SpikeProp, the network adopts the loss function defined in Equation 4. Subsequently, based on this loss function and employing the error-backpropagation algorithm, the synaptic adjustment rules for layer l are formulated as Equation 7:

$$\Delta d_{ij}^l(w_{ij}^l) = -\eta_{d(w)}\frac{\partial E}{\partial d_{ij}^l(w_{ij}^l)}, \tag{7}$$

where d_{ij}^l (w_{ij}^l) represents the synaptic delay (weight) of the connection between the ith neuron in layer $l-1$ and the jth neuron

in layer l. $\eta_{d(w)}$ represents the learning rate of delays (weights). Since the adjustment of synaptic weights is the same as that of SpikeProp (i.e., Equation 5), next, we only elaborate on the learning of synaptic delay.

- Output layer: For the delay of the output layer d_{jk}^o, based on the chain rule, we have Equation 8:

$$\frac{\partial E}{\partial d_{jk}^o} = \frac{\partial E}{\partial t_k^o}\frac{\partial t_k^o}{\partial V(t_k^o)}\frac{\partial V(t_k^o)}{\partial d_{jk}^o}. \tag{8}$$

Similarly, for the convenience of description, we define the intermediate quantity δ_k^o as Equation 9:

$$\delta_k^o = \frac{\partial E}{\partial t_k^o}\frac{\partial t_k^o}{\partial V(t_k^o)} = \frac{\partial t_k^o}{\partial V(t_k^o)} \cdot \left(t_k^o - t_k^d\right), \tag{9}$$

where $\partial t_k^o / \partial V(t_k^o)$ can be solved by Equation 6 like SpikeProp, but due to its drawbacks, we will give an alternative solution later. And the remaining terms in Equation 8 can be computed using Equation 10:

$$\frac{\partial V(t_k^o)}{\partial d_{jk}^o} = w_{jk}^o \cdot \xi_{kj}^o, \tag{10}$$

where

$$\begin{aligned}\xi_{kj}^o &= \frac{\partial K\left(t_k^o - t_j^{o-1} - d_{jk}^o\right)}{\partial d_{jk}^o} \\ &= V_{norm}\frac{1}{\tau_m}\exp\left(-\frac{t_k^o - t_j^{o-1} - d_{jk}^o}{\tau_m}\right) \\ &\quad - V_{norm}\frac{1}{\tau_s}\exp\left(-\frac{t_k^o - t_j^{o-1} - d_{jk}^o}{\tau_s}\right),\end{aligned} \tag{11}$$

- Hidden layer: For the delay of hidden layer d_{ij}^l, we have Equation 12:

$$\frac{\partial E}{\partial d_{ij}^l} = \frac{\partial E}{\partial t_j^l}\frac{\partial t_j^l}{\partial V(t_j^l)}\frac{\partial V(t_j^l)}{\partial d_{ij}^l} = \delta_j^l \cdot w_{ij}^l \cdot \xi_{ij}^l, \tag{12}$$

where t_j^l is the spike time of the jth neuron in layer l. $\delta_j^l = \frac{\partial E}{\partial t_j^l}\frac{\partial t_j^l}{\partial V(t_j^l)}$ with

$$\frac{\partial E}{\partial t_j^l} = \sum_k \frac{\partial E}{\partial t_k^{l+1}}\frac{\partial t_k^{l+1}}{\partial V(t_k^{l+1})}\frac{\partial V(t_k^{l+1})}{\partial t_j^l} = \sum_k \delta_k^{l+1} \cdot w_{jk}^{l+1}\xi_{kj}^{l+1}. \tag{13}$$

To sum up, there are Equation 14:

$$\begin{cases}\Delta w_{ij}^l = -\eta_w \cdot \delta_j^l \cdot K_{ij}^l, \\ \Delta d_{ij}^l = -\eta_d \cdot \delta_j^l \cdot w_{ij}^l \xi_{ij}^l,\end{cases} \tag{14}$$

with

$$\delta_j^l = \begin{cases}\frac{\partial t_j^l}{\partial V(t_j^l)} \cdot \left(t_j^l - t_j^d\right), & l = o \\ \frac{\partial t_j^l}{\partial V(t_j^l)} \cdot \sum_k \delta_k^{l+1} w_{jk}^{l+1}\xi_{jk}^{l+1}, & l < o\end{cases} \tag{15}$$

While the majority of SNN algorithms traditionally focus solely on updating synaptic weights, we introduce a novel approach by incorporating the adjustment of synaptic delays. This augmentation allows us to achieve joint training of both synaptic weights and delays. This dual-training approach offers two significant advantages: addressing the silent window problem and expanding the parameter space. Silent windows, a common occurrence in spiking neural networks, refer to time periods where no spiking activity takes place, potentially undermining the learning process. Weight updates alone struggle to resolve this issue. However, incorporating delay learning can effectively adjust the distribution of input spikes and mitigate this problem. Moreover, the joint training of both synaptic weights and delays provides a more extensive set of tunable parameters compared to weight-only updates. This expanded parameter space enhances the model's flexibility and can lead to improved overall performance.

2.3.2 Gradient replacement strategy

According to the above process, it can be seen that the term $\partial t_j^l / \partial V(t_j^l)$ in Equation 15 is very important for gradient calculation. If its value is calculated according to Equation 6, it means that the derivative of the membrane voltage at the firing time will be critical. More specifically, the analysis of Figure 2A reveals that during a gradual crossing of the membrane voltage threshold, the derivative approaches zero. Consequently, this leads to a significant increase in the magnitude of $|\partial t_j^l / \partial V(t_j^l)|$, resulting in a phenomenon known as gradient explosion. To address this issue, we draw upon the insights from the rectangle replacement function (Wu et al., 2018), which has demonstrated enhanced convergence in ablation experiments. Building upon this framework, we propose a novel replacement function that mitigates the problem of gradient explosion. As a result, Equation 6 can be replaced by Equation 16, as shown below.

$$\frac{\partial t_j^l}{\partial V(t_j^l)} = \begin{cases}\exp\left(-\frac{2\left(V(t_j^l)-\vartheta\right)^2}{\tau}\right), & |V(t_j^l) - \vartheta| > m, \\ \exp\left(-\frac{2m^2}{\tau}\right), & |V(t_j^l) - \vartheta| \le m.\end{cases} \tag{16}$$

where m ($0 < m < 1$) is a constant, and there is currently no accepted theoretical method for finding the optimal m on any dataset. But a good way to get an appropriate m for a specific task is to use parameter search.

Figure 2B shows the shape of replaced weight. The closer the membrane potential is to the firing time than to the threshold, the larger $\partial t_j^l / \partial V(t_j^l)$, which is consistent with the characteristics of the spike activity. However, it's important to emphasize that this value cannot become infinitely large since it would cause the value of Equation 6 to approach 0, leading the gradient to disappear. In our approach, we impose an upper limit, capping it at $\exp\left(-2m^2/\tau\right)$. This constraint ensures that the influence on $\partial t_j^l / \partial V(t_j^l)$ remains bounded. Actually, the function of the surrogate gradient can be various (Wu et al., 2018).

3 Results

In this section, we test the performance of the proposed algorithm on several different datasets, including

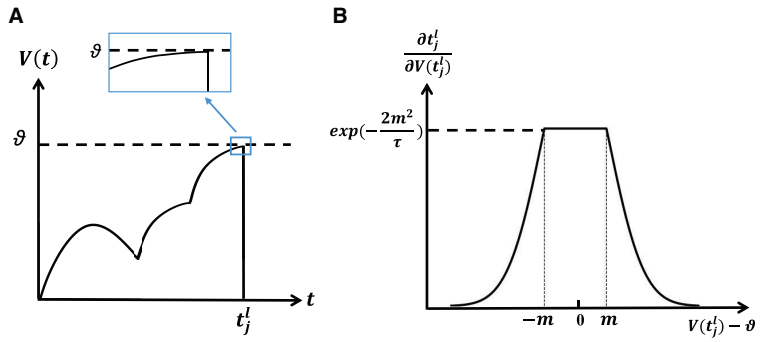


FIGURE 2
(A) Membrane potential reaches the threshold slowly, resulting in the exploding gradient. **(B)** The shape of the surrogate gradient function.

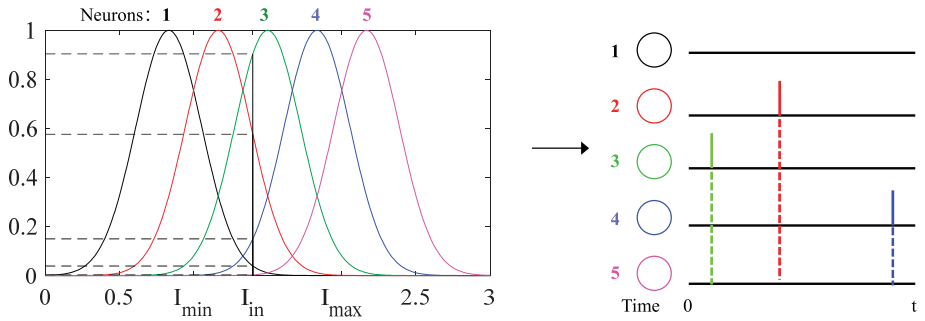


FIGURE 3
Schematic diagram of population encoding. Each neuron has its own Gaussian receiving function (different colors). I_{min} and I_{max} are the minimum and maximum values of the input current. When an input current I_{in} is fed, the corresponding value on the Gaussian function is its probability value. Finally, these values are encoded into the spiking times distributed in $[0, t]$. The larger probability value corresponds to the earlier spiking time and vice versa, and the spikes will not be emitted if the firing threshold is not reached, such as neurons "1" and "5."

Iris, Breast Cancer, Liver Disorders, Pima Diabetes, and Ionosphere. By comparison with other algorithms: SpikeProp (Shrestha and Song, 2015), SpikeTemp (Wang et al., 2015), SWAT (Wade et al., 2010), ReSuMe (Ponulak and Kasiński, 2010), SRESN (Dora et al., 2016), and MDL (Taherkhani et al., 2018), the proposed method has a good performance.

The datasets can be divided into two types: real-value datasets and image-related datasets. Samples from these databases cannot be fed directly into the network and need to be encoded into spike sequences. In real-value datasets, the population encoding (Bohte et al., 2002; Shrestha and Song, 2015; Wang et al., 2015; Taherkhani et al., 2018) is used to convert these values to input spike trains, as shown in Figure 3. In image datasets, the latency encoding (Hopfield, 1995; Hu et al., 2013) is used, as shown in Figure 4.

In the output layer each output neuron corresponds to a category, and when a training sample is fed, the corresponding output neuron is trained to fire at desired output spike time t_d generated by the dynamic decoding method (Luo et al., 2019; Zhang Y. et al., 2021), while the other neurons are kept silent.

3.1 Classification of Iris dataset

As one of the most well-known pattern recognition databases, it is divided into three categories. Each category has 50 samples with four attributes: sepal length, sepal width, petal length, and petal width. Among them, 25 samples of each category were used as the training set, and the others were used as the test set. The network structure is 25-20-3, and a sample is considered correctly classified if either its target neuron fire the most spikes or the membrane potential of its target neuron is the maximum when none of the output neurons fire. The network architecture, training epochs, train and test accuracy of our works and the contrasting methods are all depicted in Table 1. The contrast of train and test accuracy of all methods are further illustrated in Figure 5. From the results, the proposed method outperforms these methods: SpikeProp, SWAT, MDL, ReSuMe, and SpikeTemp. SRESN achieved the best test accuracy, and the proposed model is only slightly below it.

3.2 Classification of Breast Cancer dataset

The data were collected from clinical studies conducted between January 2014 and December 2014 and were derived from

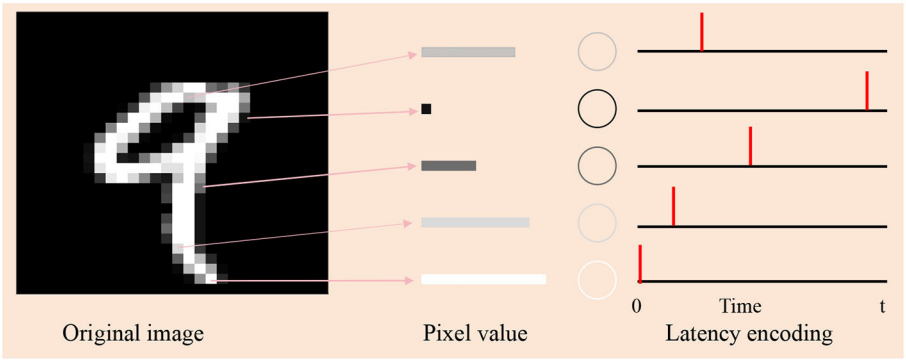


FIGURE 4
Schematic diagram of latency encoding. The subfigure on the left is an original image of the digit "9" in the MNIST. Its pixel values determine the brightness of each pixel, with higher values corresponding to whiter areas. Depending on the value, we draw five rectangles (in the middle subfigure) of different widths to represent different pixel values, with wider rectangles representing larger values. The right subfigure is the spiking times of the selected five neurons by latency encoding. The larger the pixel value, the earlier the firing time.

TABLE 1 Classification results of Iris database.

Method	Architecture	Epoch	Train (%d)	Test (%d)
SpikeProp	25-10-3	1,000	97.2 (1.9)	96.7 (1.6)
SWAT	24-312-3	500	96.7 (1.4)	92.4 (1.7)
ReSuMe	160-3	200	95.2 (1.4)	94.1 (2.0)
SRESN	24-(6-10)	102	96.9 (1.0)	97.3 (1.3)
MDL	169-360-3	100	99.8 (/)	95.7 (/)
SpikeTemp	120-87	/	100 (/)	96.7 (/)
This work (weight-only)	25-10-3	1,000	97.4 (1.2)	96.3 (1.1)
This work	25-10-3	1,000	98.1 (1.3)	97.0 (1.4)

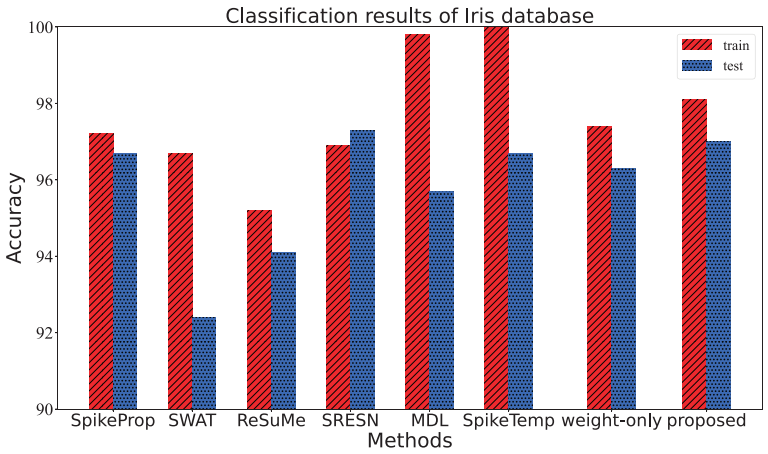


FIGURE 5
The training accuracy and test accuracy for various methods on Iris database. The best training accuracy of all methods is SpikeTemp, but the best test accuracy is SRESN which is 0.3% higher than the proposed algorithm.

microscopic biopsy images of breast lumps in patients with breast cancer. Each sample has 10 attributes to describe the characteristics of the nucleus of the mass, including radius, texture, perimeter, and so on. It is divided into two types of data: benign and malignant cancers. This dataset has 569 instances, of which 357 are benign and the remaining 212 are malignant. Among them, 179 benign samples and 106 malignant samples make up the training set, and the rest were used as test sets. The network architecture of the proposed

TABLE 2 Classification results of Breast Cancer database.

Method	Architecture	Epoch	Train (%d)	Test (%d)
SpikeProp	55-15-2	1,000	97.3 (0.6)	97.2 (0.6)
SWAT	54-702-2	500	96.5 (0.5)	95.8 (1.0)
ReSuMe	135-2	200	93.6 (0.7)	93.1 (0.8)
SRESN	54-(8-12)	102	97.7 (0.6)	97.2 (0.7)
MDL	/	100	98.2(/)	96.4 (/)
SpikeTemp	135-306	/	99.1 (/)	98.3 (/)
This work (weight-only)	55-15-2	1,000	97.2 (0.9)	96.8 (0.7)
This work	55-15-2	1,000	98.5 (0.8)	97.9 (0.5)

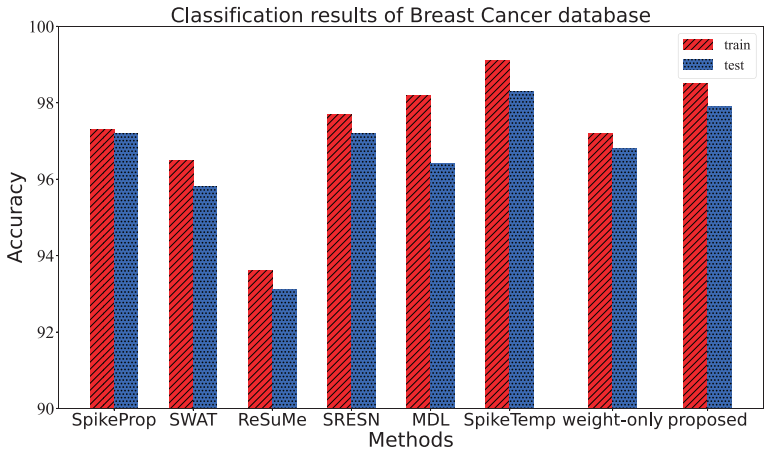


FIGURE 6
The training accuracy and test accuracy for various methods on Breast Cancer database. The training and test accuracy of the proposed method is the second, slightly lower than SpikeTemp.

method is 55-15-2. Two output neurons correspond to the benign sample and the malignant sample. If the benign output neuron fires the more spikes or has a larger membrane potential than the malignant one, the benign sample was correctly classified; and vice versa. The network architecture, training epochs, train and test accuracy of our works are shown in Table 2, along with those of the other methods. Additionally, the contrast of train and test accuracy of all methods are further illustrated in Figure 6. Experimental results show that this method outperforms most comparison algorithms including SpikeProp, SWAT, ReSuMe, SRESN, and MDL. And our model is only 0.4% lower than SpikeTemp.

3.3 Classification of Liver Disorders dataset

This dataset consisted of 345 samples of seven attributes, amongst which the first five attributes are blood data related to the development of liver disease, and the sixth attribute is the number of alcoholic drinks per day. It is a binary classification, with half of the data in each category comprising the training set and the other half comprising the test set. The input neurons, hidden neurons and output neurons are 37, 15, and 2, respectively. If the target neuron of a sample fires the overwhelming spikes or has the larger membrane potential, this sample is considered to be correctly classified. The performance of the proposed method in Table 3 and Figure 7 comes from the average of 20 trials with 3,000 training epochs, which outperforms other methods in terms of test accuracy.

3.4 Classification of Pima Diabetes dataset

This dataset contains women of at least 21 years of Pima Indian ancestry. It is also a binary classification to predict whether a patient has diabetes or not on the basis of eight attributes, including Pregnancy, Glucose, Glucose, etc. The data set includes 768 samples that are divided into training/test sets in a 1:1 ratio. The network structure is 55-20-2, and the conditions for correct classification are as follows: (1) the target output neuron of the input sample fires the most spikes, (2) the membrane potential of the target neuron overwhelms the other one when firing the same spikes. After 20 training trials of 3,000 epochs, the accuracy of our model is shown in Table 4. From Table 4 and Figure 8, the training and test accuracy of the proposed method are much higher than the all the other contrasting methods.

TABLE 3 Classification results of Liver Disorders database.

Method	Architecture	Epoch	Train (%d)	Test (%d)
SpikeProp	37-15-2	3,000	71.5 (5.2)	65.1 (4.7)
SWAT	36-468-2	500	74.8 (2.1)	60.9 (3.2)
ReSuMe	150-2	200	69.9 (5.3)	60.1 (3.4)
SRESN	36-(6-9)	715	60.4 (1.7)	59.7 (1.7)
MDL	246-360-2	100	69.9 (/)	61.8 (/)
SpikeTemp	150-226	/	93.0 (/)	58.3 (/)
This work (weight-only)	37-15-2	3,000	80.1 (2.7)	63.7 (2.4)
This work	37-15-2	3,000	85.6 (3.4)	66.7 (3.1)

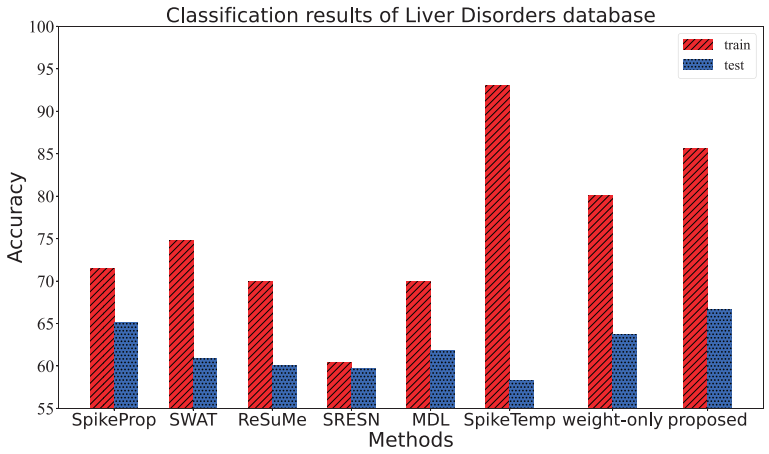


FIGURE 7
The training accuracy and test accuracy for various methods on Liver Disorders database. The training accuracy of the SpikeTemp is the best, while the test accuracy of the proposed method is the best.

TABLE 4 Classification results of Pima Diabetes database.

Method	Architecture	Epoch	Train (%d)	Test (%d)
SpikeProp	55-20-2	3,000	78.6 (2.5)	76.2 (1.8)
SWAT	54-702-2	500	77.0 (2.1)	72.1 (1.8)
ReSuMe	80-2	200	76.4 (1.5)	69.6 (2.0)
SRESN	54-(9-14)	254	70.5 (2.4)	69.9 (2.0)
MDL	/	100	72.1 (/)	70.6 (/)
SpikeTemp	80-431	/	77.5 (/)	67.6 (/)
This work (weight-only)	55-20-2	3,000	78.2 (1.7)	77.1 (0.7)
This work	55-20-2	3,000	79.2 (2.0)	77.0 (1.3)

3.5 Classification of Ionosphere dataset

This dataset contains 351 samples of radar data collected by Johns Hopkins University. Each sample consists of 35 attributes, the first 34 attributes are contiguous, and the last attribute is the category label. This data set has two categories, and equal numbers of the samples from each category were added to the training set and test set respectively. The network architecture of this experiment is 205-25-2. When a sample is fed into the network, the category is determined if its target neuron emits the most spikes, or if the target neuron has the maximum membrane potential when emitting the same number of spikes as the other neuron. The network is trained 3,000 epochs in each trial, and the average of 20 trials is shown in Table 5. From Table 5 and Figure 9, the test accuracy of the proposed model ranks only below spikeTemp with a slight gap, but obviously higher than the other five comparison algorithms.

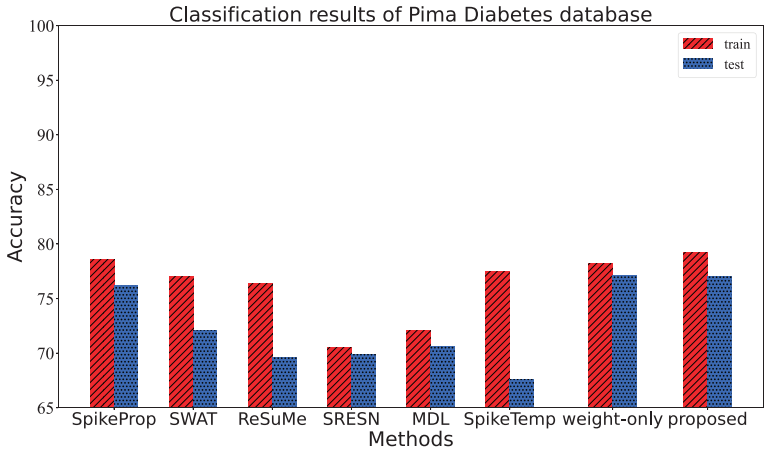


FIGURE 8
The training accuracy and test accuracy for various methods on Pima Diabetes database. The training accuracy and test accuracy of the proposed method is the best.

TABLE 5 Classification results of Ionosphere database.

Method	Architecture	Epoch	Train (%d)	Test (%d)
SpikeProp	205-25-2	3,000	89.0 (7.9)	86.5 (7.2)
SWAT	204-2652-2	500	86.5 (7.2)	90.0 (2.3)
ReSuMe	231-2	200	94.6 (0.6)	89.5 (1.8)
SRESN	204-(16-23)	1,018	91.9 (1.8)	88.6 (1.6)
MDL	/	100	96.0 (/)	90.5 (/)
SpikeTemp	231-223	/	86.8 (/)	91.5 (/)
This work (weight-only)	205-25-2	3,000	90.6 (2.7)	87.2 (1.8)
This work	205-25-2	3,000	92.7 (4.1)	90.7 (2.5)

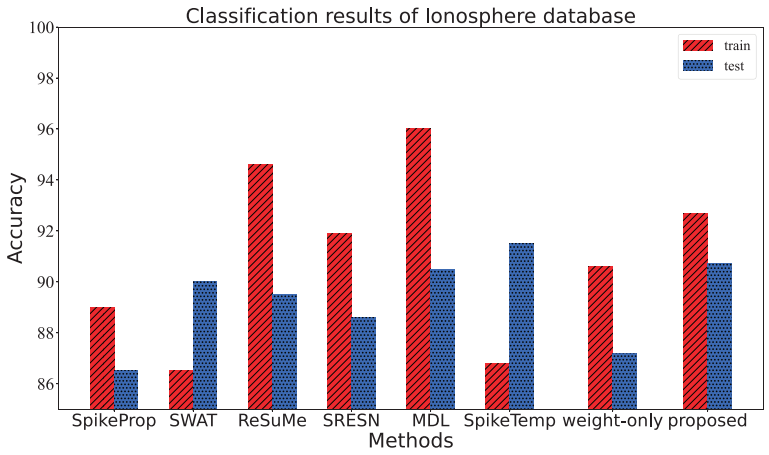


FIGURE 9
The training accuracy and test accuracy for various methods on Ionosphere database. The proposed method gets the best test accuracy and the smallest gap between training accuracy and test accuracy.

TABLE 6 Comparison with other works on MNIST dataset.

Method	Architecture	Learning method	Acc (%d)
Mostafa (2017)	784-800-10	Temporal backpropagation	97.20
Tavanaei and Maida (2019)	784-1000-10	STDP-based backpropagation	96.60
Comsa et al. (2020)	784-340-10	Temporal backpropagation	97.9
ANN (Kheradpisheh and Masquelier, 2020)	784-400-10	Backpropagation with Adam	98.10
S4NN (Kheradpisheh and Masquelier, 2020)	784-400-10	Temporal backpropagation	97.4
This work (weight-only)	784-800-10	Temporal backpropagation	96.7
This work	784-800-10	Temporal backpropagation	97.6

3.6 Classification of MNIST dataset

To exploit and testify the image information learning capacity of the proposed method, we conducted experiments on the widely used MNIST dataset (LeCun et al., 1998), which is a popular choice in deep learning research (Mostafa, 2017; Tavanaei and Maida, 2019; Comsa et al., 2020). This dataset comprises 60,000 training samples and 10,000 testing samples, each of which has a visual scale of 28 × 28 pixels. The pixels are encoded as spike trains through the latency encoding method (Hopfield, 1995) and then fed in the proposed method with the architecture 784-800-10. We compare the experimental results with some effective ANN and SNN works, as detailed in Table 6. The results show that the proposed model has a comparable performance on image data, which outperforms the contrasting SNN models, and only trivially falls behind artificial neural networks in terms of accuracy.

4 Discussion

In this paper, we proposed a new supervised learning algorithm for multi-layer spiking neural networks, which considers the plasticity of both synaptic weights and delays. Various experiments are conducted to verify the performance of the proposed learning method, and the experimental results support its superiority. Actually, how to train multilayer spiking neural networks remains an open question. The existing learning rules can be classified as ANN-to-SNN, surrogate gradient method, and spike-driven learning algorithms. In the following, we will compare the proposed method with these methods.

4.1 Compared to ANN-to-SNN methods

The ANN-to-SNN methods are proposed to avoid the difficult training of deep spiking neural networks. However, most of the ANN-to-SNN methods are based on the spike rate information, the time information has not been fully leveraged. In addition, the existing ANN-to-SNN conversion methods can only deal with image datasets. Those datasets with rich temporal information like speech and video can not be addressed by these methods. Our algorithm adopts temporal coding to make full use of the time information of spikes and has more potential to process these temporal datasets.

4.2 Compared to surrogate gradient methods

Due to the non-differentiable spike function, directly training spiking neural networks is very difficult. To resolve this problem, surrogate gradient-based learning algorithms are proposed. By using a surrogate gradient, these methods do not need to calculate the exact gradients. However, these methods need to do backpropagation at every time step and cost a lot of computing sources. In contrast, our algorithm holds the potential of enabling training and inference in low-power devices.

4.3 Compared to spike-driven methods

There are various spike-driven learning methods, such as SpikeProp and STDBP. However, most existing spike-driven learning algorithms only consider the plasticity of synaptic weights and ignore synaptic delay adjustment. In our algorithm, both the synaptic weights and delays are considered adjustable variables to improve both the biological plausibility and the learning performance. Experimental results demonstrate that our algorithm achieves a competitive learning performance compared with the existing related works. In the future, we would like to further extend the application of spike-driven learning algorithms on large-scale datasets and other practical applications (Liu and Li, 2022; Liu et al., 2022a,b).

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

JW: Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

Funding

This work was supported by the National Science Foundation of China under Grant 61976043.

Conflict of interest

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

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: a brief survey. *IEEE Signal Process. Mag.* 34, 26–38. doi: 10.1109/MSP.2017.2743240

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Caporale, N., and Dan, Y. (2008). Spike timing-dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.* 31, 25–46. doi: 10.1146/annurev.neuro.31.060407.125639

Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., Alakuijala, J., et al. (2020). "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Barcelona: IEEE), 8529–8533. doi: 10.1109/ICASSP40776.2020.9053856

Dora, S., Subramanian, K., Suresh, S., and Sundararajan, N. (2016). Development of a self-regulating evolving spiking neural network for classification problem. *Neurocomputing* 171, 1216–1229. doi: 10.1016/j.neucom.2015.07.086

Gerstner, W. (1995). Time structure of the activity in neural network models. *Phys. Rev. E* 51, 738. doi: 10.1103/PhysRevE.51.738

Ghosh-Dastidar, S., and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308. doi: 10.1142/S0129065709002002

Gütig, R. (2016). Spiking neurons can discover predictive features by aggregate-label learning. *Science* 351, aab4113. doi: 10.1126/science.aab4113

Han, Y., Xiang, S., Ren, Z., Fu, C., Wen, A., Hao, Y., et al. (2021). Delay-weight plasticity-based supervised learning in optical spiking neural networks. *Photonics Res.* 9, 119–127. doi: 10.1364/PRJ.413742

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)* (Las Vegas, NV: IEEE), 770–778. doi: 10.1109/CVPR.2016.90

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature* 376, 33–36. doi: 10.1038/376033a0

Hu, J., Tang, H., Tan, K. C., Li, H., and Shi, L. (2013). A spike-timing-based integrated model for pattern recognition. *Neural Comput.* 25, 450–472. doi: 10.1162/NECO_a_00395

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440

Kheradpisheh, S. R., and Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *Int. J. Neural Syst.* 30, 2050027. doi: 10.1142/S0129065720500276

Lan, Y., Wang, X., and Wang, Y. (2021). Spatio-temporal sequential memory model with mini-column neural network. *Front. Neurosci.* 15, 374. doi: 10.3389/fnins.2021.650430

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

Liu, Q., and Li, X. (2022). Efficient low-rank matrix factorization based on $\ell 1,\varepsilon$-norm for online background subtraction. *IEEE Trans. Circuits Syst. Video Technol.* 32, 4900–4904. doi: 10.1109/TCSVT.2021.3129503

Liu, Q., Li, X., Cao, H., and Wu, Y. (2022a). From simulated to visual data: a robust low-rank tensor completion approach using ℓp-regression for outlier resistance. *IEEE Trans. Circuits Syst. Video Technol.* 32, 3462–3474. doi: 10.1109/TCSVT.2021.3114208

Liu, Q., Li, X., and Yang, J. (2022b). Optimum codesign for image denoising between type-2 fuzzy identifier and matrix completion denoiser. *IEEE Trans. Fuzzy Syst.* 30, 287–292. doi: 10.1109/TFUZZ.2020.3030498

Luo, X., Qu, H., Wang, Y., Yi, Z., Zhang, J., Zhang, M., et al. (2022). Supervised learning in multilayer spiking neural networks with spike temporal error backpropagation. *IEEE Trans. Neural Netw. Learn. Syst.* 34, 10141–10153. doi: 10.1109/TNNLS.2022.3164930

Luo, X., Qu, H., Zhang, Y., and Chen, Y. (2019). First error-based supervised learning algorithm for spiking neural networks. *Front. Neurosci.* 13, 559. doi: 10.3389/fnins.2019.00559

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Maass, W., and Bishop, C. M. (2001). *Pulsed Neural Networks*. Cambridge, MA: MIT press.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Neftci, E. O., Charles, A., Somnath, P., and Georgios, D. (2016). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11, 324. doi: 10.3389/fnins.2017.00324

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

Pan, Z., Zhang, M., Wu, J., Wang, J., and Li, H. (2021). Multi-tone phase coding of interaural time difference for sound source localization with spiking neural networks. *IEEE/ACM Trans. Audio Speech Lang. Process.* 29, 2656–2670. doi: 10.1109/TASLP.2021.3100684

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 Kasiński, A. (2010). Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Shrestha, S., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems, Vol. 31*, 1419–1428.

Shrestha, S. B., and Song, Q. (2015). Adaptive learning rate of spikeprop based on weight convergence analysis. *Neural Netw.* 63, 185–198. doi: 10.1016/j.neunet.2014.12.001

Subakan, C., Ravanelli, M., Cornell, S., Bronzi, M., and Zhong, J. (2021). "Attention is all you need in speech separation," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Toronto, ON: IEEE), 21–25. doi: 10.1109/ICASSP39728.2021.9413901

Sun, P., Eqlimi, E., Chua, Y., Devos, P., and Botteldooren, D. (2023). "Adaptive axonal delays in feedforward spiking neural networks for accurate spoken word recognition," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Rhodes Island: IEEE), 1–5. doi: 10.1109/ICASSP49357.2023.10094768

Sun, P., Zhu, L., and Botteldooren, D. (2022). "Axonal delay as a short-term memory for feed forward deep spiking neural networks," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Singapore: IEEE), 8932–8936. doi: 10.1109/ICASSP43922.2022.9747411

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015a). Dl-resume: a delay learning-based remote supervised method for

spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3137–3149. doi: 10.1109/TNNLS.2015.2404938

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015b). "Multi-dl-resume: multiple neurons delay learning remote supervised method," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–7. doi: 10.1109/IJCNN.2015.7280743

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2018). A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 5394–5407. doi: 10.1109/TNNLS.2018.2797801

Tavanaei, A., and Maida, A. (2019). Bp-stdp: approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330, 39–47. doi: 10.1016/j.neucom.2018.11.014

Wade, J. J., McDaid, L. J., Santos, J. A., and Sayers, H. M. (2010). Swat: a spiking neural network training algorithm for classification problems. *IEEE Trans. Neural Netw.* 21, 1817–1830. doi: 10.1109/TNN.2010.2074212

Wang, J., Belatreche, A., Maguire, L. P., and McGinnity, T. M. (2015). Spiketemp: an enhanced rank-order-based learning approach for spiking neural networks with adaptive structure. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 30–43. doi: 10.1109/TNNLS.2015.2501322

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., Tan, K. C., et al. (2021a). A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 34, 446–460. doi: 10.1109/TNNLS.2021.3095724

Wu, J., Xu, C., Han, X., Zhou, D., Zhang, M., Li, H., et al. (2021b). Progressive tandem learning for pattern recognition with deep spiking neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 7824–7840. doi: 10.1109/TPAMI.2021.3114196

Wu, J., Zhan, Y., Peng, Z., Ji, X., and Wang, C. (2021c). Efficient design of spiking neural network with stdp learning based on fast cordic. *IEEE Trans. Circuits Syst. I: Regul. Pap.* 68, 2522–2534. doi: 10.1109/TCSI.2021.3061766

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12, 331. doi: 10.3389/fnins.2018.00331

Yha, B., Xh, A., Meng, D. A., and Bo, X. (2020). A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Netw.* 121, 387–395. doi: 10.1016/j.neunet.2019.09.007

Zhang, M., Wang, J., Wu, J., Belatreche, A., Amornpaisannon, B., Zhang, Z., et al. (2021). Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 1947–1958. doi: 10.1109/TNNLS.2021.3110991

Zhang, M., Wu, J., Belatreche, A., Pan, Z., Xie, X., Chua, Y., et al. (2020). Supervised learning in spiking neural networks with synaptic delay-weight plasticity. *Neurocomputing* 409, 103–118. doi: 10.1016/j.neucom.2020.03.079

Zhang, Y., Chen, Y., Zhang, J., Luo, X., Zhang, M., Qu, H., et al. (2022). Minicolumn-based episodic memory model with spiking neurons, dendrites and delays. *IEEE Trans. Neural Netw. Learn. Syst.* 1–15. doi: 10.1109/TNNLS.2022.3213688

Zhang, Y., Qu, H., Luo, X., Chen, Y., Wang, Y., Zhang, M., et al. (2021). A new recursive least squares-based learning algorithm for spiking neurons. *Neural Netw.* 138, 110–125. doi: 10.1016/j.neunet.2021.01.016

Zhu, X., Zhao, B., Ma, D., and Tang, H. (2021). An efficient learning algorithm for direct training deep spiking neural networks. *IEEE Trans. Cogn. Develop. Syst.* 14, 847–856. doi: 10.1109/TCDS.2021.3073846

frontiers | Frontiers in Computational Neuroscience

TYPE Original Research
PUBLISHED 16 May 2024
DOI 10.3389/fncom.2024.1240348

OPEN ACCESS

EDITED BY
Lei Deng,
Tsinghua University, China

REVIEWED BY
Shuangming Yang,
Tianjin University, China
Desmond Loke,
Singapore University of Technology and Design, Singapore
Zhaodong Chen,
Nvidia, United States

*CORRESPONDENCE
Kyle Daruwalla
✉ daruwal@cshl.edu

RECEIVED 14 June 2023
ACCEPTED 26 April 2024
PUBLISHED 16 May 2024

CITATION
Daruwalla K and Lipasti M (2024) Information bottleneck-based Hebbian learning rule naturally ties working memory and synaptic updates.
Front. Comput. Neurosci. 18:1240348.
doi: 10.3389/fncom.2024.1240348

Information bottleneck-based Hebbian learning rule naturally ties working memory and synaptic updates

Kyle Daruwalla[1]* and Mikko Lipasti[2]

[1]Cold Spring Harbor Laboratory, Long Island, NY, United States, [2]Electrical and Computer Engineering Department, University of Wisconsin-Madison, Madison, WI, United States

Deep neural feedforward networks are effective models for a wide array of problems, but training and deploying such networks presents a significant energy cost. Spiking neural networks (SNNs), which are modeled after biologically realistic neurons, offer a potential solution when deployed correctly on neuromorphic computing hardware. Still, many applications train SNNs *offline*, and running network training directly on neuromorphic hardware is an ongoing research problem. The primary hurdle is that back-propagation, which makes training such artificial deep networks possible, is biologically implausible. Neuroscientists are uncertain about how the brain would propagate a precise error signal backward through a network of neurons. Recent progress addresses part of this question, e.g., the weight transport problem, but a complete solution remains intangible. In contrast, novel learning rules based on the information bottleneck (IB) train each layer of a network independently, circumventing the need to propagate errors across layers. Instead, propagation is implicit due the layers' feedforward connectivity. These rules take the form of a three-factor Hebbian update a global error signal modulates local synaptic updates within each layer. Unfortunately, the global signal for a given layer requires processing multiple samples concurrently, and the brain only sees a single sample at a time. We propose a new three-factor update rule where the global signal correctly captures information across samples via an auxiliary memory network. The auxiliary network can be trained *a priori* independently of the dataset being used with the primary network. We demonstrate comparable performance to baselines on image classification tasks. Interestingly, unlike back-propagation-like schemes where there is no link between learning and memory, our rule presents a direct connection between working memory and synaptic updates. To the best of our knowledge, this is the first rule to make this link explicit. We explore these implications in initial experiments examining the effect of memory capacity on learning performance. Moving forward, this work suggests an alternate view of learning where each layer balances memory-informed compression against task performance. This view naturally encompasses several key aspects of neural computation, including memory, efficiency, and locality.

KEYWORDS

neuromorphic computing, Neural Network, learning rule, information bottleneck, back-propagation

1 Introduction

The success of deep learning demonstrates the usefulness of large feedforward neural networks for solving a variety of tasks, but the energy cost associated with such networks presents an ongoing problem (Strubell et al., 2019). Neuromorphic computing platforms and spiking neural networks (SNNs), which model the power efficient properties of biological neural networks, offer a possible solution (Christensen et al., 2022). While recent advances allow SNNs to be trained *offline* (Neftci et al., 2019), these approaches only

benefit from energy-efficient inference even though training continues to be the dominant energy bottleneck for deep learning. Though there are many strategies for training SNNs, it is widely believed that the most effective technique will be a biologically plausible learning rule (Zenke et al., 2021). While reproducing biology is not a strict requirement, the engineering constraints of neuromorphic hardware naturally align with biological constraints. Namely, we identify three defining properties of biologically plausible learning rules that directly impact energy efficiency: *locality, asynchrony, and real-time processing*. These three properties reduce the communication overhead and coordination required by a neuromorphic chip which are large sources of power consumption (Christensen et al., 2022).

Unfortunately, training spiking neural networks directly on hardware is challenging, since the driving factor behind deep learning's success—back-propagation—is not considered to be biologically plausible (Lillicrap et al., 2020). Specifically, it is unclear how neurons might propagate a precise error signal within a forward/backward pass framework like back-propagation. A large body of work has been devoted to establishing plausible alternatives or approximations for this error propagation scheme (Balduzzi et al., 2015; Scellier and Bengio, 2017; Akrout et al., 2019; Lillicrap et al., 2020). While these approaches do address some of the issues with back-propagation, implausible elements, like separate inference and learning phases, still persist in many cases.

Our work joins a body of recent literature that addresses biological plausibility by suggesting fundamentally different approaches to training networks from back-propagation (Payeur et al., 2021; Meulemans et al., 2022; Aceituno et al., 2023). These approaches modulate local Hebbian updates using top-down signals based on alternative objectives such as optimizing a control policy. Our work is similar in that we propose a dramatically different training objective. In contrast, we rely on recent advances in deep learning that train feedforward networks by balancing an information bottleneck objective (Ma et al., 2019). Unlike back-propagation, where an error signal computed at the end of the network is propagated to the front (see Figure 1A), this method, called the Hilbert-Schmidt Independence Criterion (HSIC) bottleneck, applies the information bottleneck to each layer in the network independently. Layer-wise optimization is biologically plausible as shown in Figure 1B. Compared to related work, where the performance of the final layer affects training of prior layers through top-down signals, our objective is fully localized at each layer.

Our contributions include:

1. We show that optimizing the HSIC bottleneck via gradient descent emits a three-factor learning rule (Frémaux and Gerstner, 2016) composed of a local Hebbian component and a global layer-wise modulating signal.
2. The HSIC bottleneck depends on a batch of samples, and this is reflected in our update rule. Unfortunately, the brain only sees a single sample at a time. We show that the local component only requires the current sample, and that the global component can be accurately computed by an auxiliary network. The auxiliary networks acts as a working memory with post-processing, and the effective "batch size" corresponds to its capacity.
3. We demonstrate the empirical performance of our update rule by comparing it against baselines on synthetic datasets as well as MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky, 2009).
4. To the best of our knowledge, our rule is the first to make a direct connection between working memory and synaptic updates. We explore this connection in some initial experiments on memory size and learning performance.

1.1 Preliminaries and related work

Several works have presented approximations to back-propagation. Variants of feedback alignment (Lillicrap et al., 2014; Liao et al., 2016; Akrout et al., 2019) address the weight transport problem. Target propagation (Ahmad et al., 2020; Frenkel et al., 2021) and equilibrium propagation (Scellier and Bengio, 2017) propose alternative mechanisms for propagating error. Yet, all these methods require separate inference (forward) and learning (backward) phases. More recently, deep feedback control methods (Meulemans et al., 2022; Aceituno et al., 2023) use top-down signaling from a controller to optimize forward and backward weights concurrently. Unlike prior methods which address biological plausibility piecemeal, these techniques are plausible by design. We follow this approach to creating plausible learning rules, but we differ by focusing on layer-wise objectives instead of top-down control. Table 1 shows a comprehensive comparison between learning rule definitions. Only direct feedback control (DFC) and our work satisfy all objectives, but they represent two different solutions to the problem of biologically plausible learning rules. DFC is framed as a control problem with continuous dynamics, and the resulting weight update requires multi-compartment neurons. The authors note that this makes their work better suited for analog neuromorphic hardware. In contrast, our rule can be mapped to both digital or analog hardware, since time is denoted by sequences of samples and not physical time. Additionally, we do not put constraints on the neurons required to implement the rule.

Layer-wise objectives (Belilovsky et al., 2019; Nøkland and Eidnes, 2019), like the one used in this work, offer an alternative that avoids the weight transport problem entirely. Moreover, our objective emits a biologically plausible three-factor learning rule which can be applied concurrently with inference. Pogodin and Latham (2020) draw similar intuition in their work on the plausible HSIC (pHSIC) learning rule. But in order to make experiments with the pHSIC computationally feasible, the authors used an approximation where the network receives a batch of 256 samples at once. In contrast, their proposed biologically plausible rule only receives information from two samples—the current one and previous one—which reduces the accuracy of the HSIC estimate. This motivates our work, in which we derive an alternate rule where only the global component depends on past samples, while the local component only requires the current pre- and post-synaptic activity. Furthermore, we show that this global component can be computed using an auxiliary network. This allows us to achieve performance much closer to back-propagation without compromising the biological plausibility of the rule.

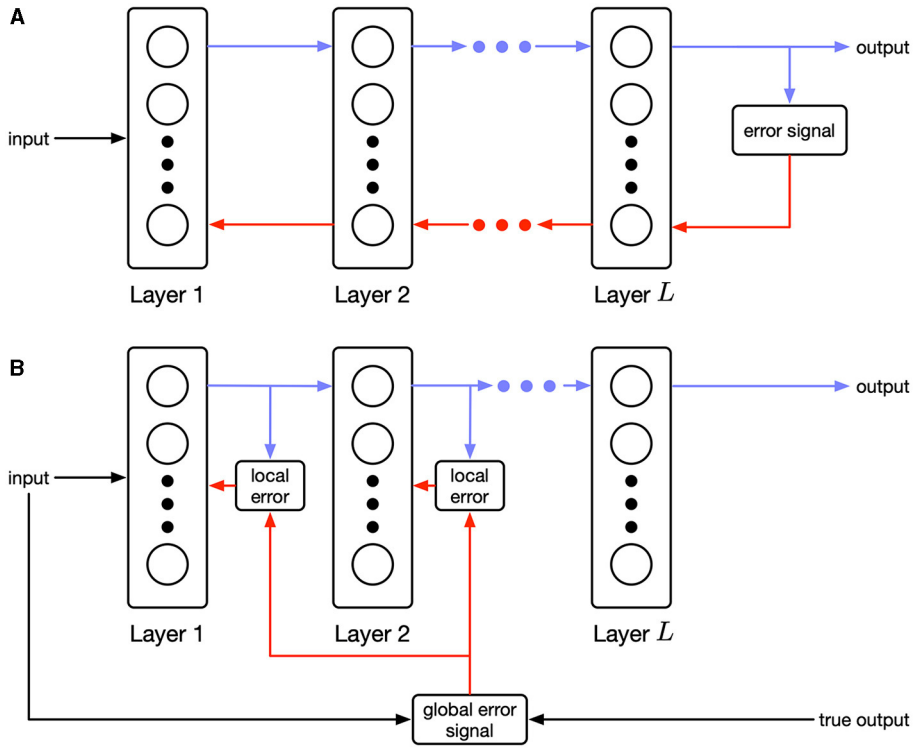


FIGURE 1
(A) Sequential (explicit) error propagation requires precise information transfer *backwards* between layers. **(B)** Parallel (implicit) error propagation uses only local information in combination with a global modulating signal. Biological rules of this form are known as three-factor learning rules (Frémaux and Gerstner, 2016).

TABLE 1 **A comparison of various learning algorithms categorized by four properties.**

Learning algorithm	Weight transport-free?	Local?	Asynchronous?	Real-time?
Back-propagation (BP)	✗	✗	✗	✗
Feedback alignment (FA)	✓	✗	✗	✗
Direct FA (DFA)	✓	✓	✗	✗
Single sparse DFA (SDFA)	✓	✓	✓	✗
Equilibrium propagation (EP)	✓	✓	✗	✗
Target propagation (TP)	✓	✓	✗	✗
Direct random TP (DRTP)	✓	✓	✓	✗
Plausible HSIC (pHSIC)	✓	✓	✓	✗
Direct feedback control (DFC)	✓	✓	✓	✓
Our work	✓	✓	✓	✓

Weight transport-free rules do not require separate forward and backward networks with aligned weight parameters. *Local* rules utilize only locally available information. *Asynchronous* rules do not require a full forward pass before updating the weights of each layer. *Real-time* rules operate on samples arriving sequentially in time (not batches).

1.1.1 Other uses of information theoretic objectives for spiking neural networks

Information bottleneck and other information theoretic quantities have been used in the context of training SNNs before. Yang and Chen (2023a,b) utilize an information bottleneck objective in the final layer of an SNN to train networks that are robust to noisy input distributions. Yang and Chen (2023a) improves on the standard information bottleneck by considering higher order terms. Similarly, Yang et al. (2022) trains networks with an additional minimum entropy criterion to promote robust learning in SNNs. Still, all works rely on back-propagation through time (BPTT) and surrogate gradient descent to train their SNNs.

1.1.2 Hardware substrates for implementing biological neural networks

While our work does not directly deal with hardware implementations of biological networks, our contributions are

motivated by the possible power efficiency benefits of biologically plausible learning rules. As such, we will briefly discuss various platforms for physical realization of neuromorphic computing.

The landscape of neuromorphic hardware is vast and varied. At one extreme, platforms like Intel's Loihi (Davies et al., 2018, 2021) use conventional CMOS technologies to create a digital array of biological neurons. While such systems are useful for exploring SNN applications, it is widely accepted that the primary power efficiency of neuromorphic hardware will come from novel device technologies. The most common devices are memristors (Yan et al., 2023) and resistive memory (Bianchi et al., 2023). Less common substrates based on metal-organic transistors (Wang et al., 2023) and thermal-guiding structures (Loke et al., 2016) exist as well. These devices are designed to mimic various functions of a biological synapse especially its plastic conductance. While the specifics differ, all devices have electrical properties that allow the conductance to be adjustable. Successful demonstrations of neuromorphic device arrays show their ability to simulate SNNs at a much lower power consumption than conventional computing. Yet, all rely on purely local update rules which fail to scale to very deep networks. Circumventing this limitation requires non-local circuitry, and the goal of any biologically plausible rule, including ours, is to limit the power overhead of these components.

1.1.3 Notation

We will briefly introduce the notation used in the paper.

- Vectors are indicated in bold and lower-case (e.g., $\mathbf{x}$).
- Matrices are indicated in bold and upper-case (e.g., $\mathbf{W}$).
- Superscripts refer to different layers of a feedforward network (e.g., $\mathbf{z}^\ell$ is the ℓ-th layer).
- Subscripts refer to individual samples (e.g., $\mathbf{x}_i$ is the i-th sample).
- Brackets refer to elements within a matrix or vector (e.g., $[\mathbf{x}]_i$ is the i-th element of $\mathbf{x}$).

2 Methods and materials

In this section, we describe our learning rule and its derivation in detail. Section 2.1 introduces the information bottleneck for deep networks. Then in Section 2.2, we derive a gradient descent rule for this objective, as well as introduce reasonable approximations such that the final rule is a three-factor Hebbian update. Lastly, in Section 2.2.1 we describe how the modulating factor in our rule can be computed using an auxiliary network.

2.1 The information bottleneck

Given an input random variable, X, an output label random variable, Y, and hidden representation, T, the information bottleneck (IB) principle is described by Equation (1)

$$\min_{\mathbb{P}_{T|X}} I(X;T) - \gamma I(Y;T) \tag{1}$$

where $I(A;B)$ is the mutual information between two random variables. Intuitively, this expression adjusts T to achieve a balance between information compression and output preservation.

Since computing the mutual information of two random variables requires knowledge of their distributions, Ma et al. (2019) propose using the Hilbert-Schmidt Independence Criterion (HSIC) as a proxy for mutual information. Given a finite number of samples, N, a statistical estimate, shown in Equation (2), for the HSIC (Gretton et al., 2005) is

$$\begin{aligned}\mathrm{HSIC}(X,Y) &= (N-1)^{-2}\mathrm{tr}(\mathbf{K_X H K_Y H}) \\ &\leq \frac{1}{(N-1)^2}\sum_{p=1}^{N}\bar{k}(\mathbf{x}_p,\mathbf{x}_p)\bar{k}(\mathbf{y}_p,\mathbf{y}_p)\end{aligned} \tag{2}$$

$$\begin{aligned}[\mathbf{K_X H}]t_{pq} &= \bar{k}(\mathbf{x}_p,\mathbf{x}_q) \\ &= k(\mathbf{x}_p,\mathbf{x}_q) - \tfrac{1}{N}\textstyle\sum_{n=1}^{N}k(\mathbf{x}_p,\mathbf{x}_n)\end{aligned} \tag{3}$$

$$[\mathbf{K_X}]_{pq} = k(\mathbf{x}_p,\mathbf{x}_q) = \exp\left(-\frac{\|\mathbf{x}_p-\mathbf{x}_q\|^2}{\sigma^2}\right) \tag{4}$$

where Equations (3) and (4) define the centered and uncentered kernel matrices, respectively.

Using these definitions Ma et al. (2019) define the HSIC objective—a loss function for training feedforward neural networks by balancing the IB at each layer. Consider a feedforward neural network with L layers where the output of layer ℓ is

$$\mathbf{z}^\ell = f(\boldsymbol{\theta}^\ell, \mathbf{z}^{\ell-1})$$

where f describes the forward computation (including nonlinear activation) of a single layer given parameters, $\boldsymbol{\theta}^\ell$, and inputs, $\mathbf{z}^{\ell-1}$. For example, a fully-connected layer of artificial neurons with a ReLU activation is described by $\boldsymbol{\theta}^\ell = \{\mathbf{W}^\ell, \mathbf{b}^\ell\}$ and $f(\boldsymbol{\theta}^\ell, \mathbf{z}^{\ell-1}) = \mathrm{relu}(\mathbf{W}^\ell\mathbf{z}^{\ell-1} + \mathbf{b}^\ell)$. In this work we will define f for both artificial neuron layers and rate-encoded leaky-integrate neuron layers.

We train the network to minimize

$$\begin{aligned}\mathcal{L}_{\mathrm{HSIC}}(X,Y,Z^\ell) &= \mathrm{HSIC}(X,Z^\ell) - \gamma\mathrm{HSIC}(Y,Z^\ell) \\ &\forall \ell \in \{1,\ldots,L\}\end{aligned} \tag{5}$$

where $Z = \{Z^\ell\}_{\ell=1}^{L}$ are the output distributions at each hidden layer. Note that there is a separate objective for each layer. As a result, there is no explicit error propagation across layers, and the error propagation is implicit due to forward connectivity as shown in Figure 1.

2.2 Deriving a biologically plausible rule for the HSIC bottleneck

In this work, we seek to derive biologically plausible rule for optimizing Equation (5). Computing this quantity requires a batch of N samples, but we want a rule that operates on single samples arriving sequentially over time. So, we will make a minor notational change to the indexing in Equation (5) for clarity. Our indices will range over $\{0,-1,\ldots,-(N-1)\}$ instead of $\{1,2,\ldots,N\}$, so that x_0 refers to the current input sample, x_{-1} refers to the previous input

sample, etc. We operate on N samples, but we are explicit that these samples arrive at different points in time.

Now, we take the gradient of $\mathcal{L}_{\text{HSIC}}$ with respect to $\boldsymbol{\theta}^\ell$ and applying gradient descent. Doing this, we obtain the following update rule:

$$\begin{aligned}
\Delta\boldsymbol{\theta}^\ell \propto \nabla_{\boldsymbol{\theta}^\ell}\mathcal{L}_{\text{HSIC}} &= \frac{1}{(N-1)^2}\sum_{p=0}^{-(N-1)}\left[\bar{k}(\mathbf{x}_p,\mathbf{x}_p)-\gamma\bar{k}(\mathbf{y}_p,\mathbf{y}_p)\right]\\
&\quad \nabla_{\boldsymbol{\theta}^\ell}\bar{k}(\mathbf{z}_p^\ell,\mathbf{z}_p^\ell)\\
\nabla_{\boldsymbol{\theta}^\ell}\bar{k}(\mathbf{z}_p^\ell,\mathbf{z}_p^\ell) &= \frac{2}{N\sigma^2}\sum_{n=0}^{-(N-1)}k(\mathbf{z}_p^\ell,\mathbf{z}_n^\ell)(\mathbf{z}_p^\ell-\mathbf{z}_n^\ell)(\nabla_{\boldsymbol{\theta}^\ell}\mathbf{z}_p^\ell-\nabla_{\boldsymbol{\theta}^\ell}\mathbf{z}_n^\ell)\\
&= \frac{2}{N\sigma^2}\sum_{n=0}^{-(N-1)}k(\mathbf{z}_p^\ell,\mathbf{z}_n^\ell)(\mathbf{z}_p^\ell-\mathbf{z}_n^\ell)(\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_p^{\ell-1})\\
&\quad -\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_n^{\ell-1}))
\end{aligned} \tag{6}$$

where $\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_p^{\ell-1})$ describes how the post-synaptic activity varies as a function of pre-synaptic activity. We call N, the batch size in the deep learning, the *effective batch size* in our work. This rule is similar to the basic rule in Pogodin and Latham (2020), except that they replace $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)$ with $\bar{k}(\mathbf{z}_p,\mathbf{z}_p)$ and do not use a centered kernel matrix.

Without modifications, Equation (6) is not biologically plausible. $\nabla_{\boldsymbol{\theta}^\ell}\bar{k}(\mathbf{z}_p^\ell,\mathbf{z}_p^\ell)$ cannot be called Hebbian when p is not equal to zero, since it depends on non-local information from the past. We solve this by making a simplifying approximation. We assume that $\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_p^{\ell-1})=0$ when $p\neq 0$. In other words, the weights at the current time step do not affect past outputs. With this assumption, we find that

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}^\ell}\bar{k}(\mathbf{z}_p^\ell,\mathbf{z}_p^\ell)\\
&=\begin{cases}\frac{2}{N\sigma^2}\sum_{n=1}^{-(N-1)}k(\mathbf{z}_0^\ell,\mathbf{z}_n^\ell)(\mathbf{z}_0^\ell-\mathbf{z}_n^\ell)\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_0^{\ell-1}) & p=0\\ \frac{2}{N\sigma^2}k(\mathbf{z}_0^\ell,\mathbf{z}_p^\ell)(\mathbf{z}_0^\ell-\mathbf{z}_p^\ell)\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,\mathbf{z}_0^{\ell-1}) & p\neq 0\end{cases}
\end{aligned}$$

Notably, the local term, $\nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,z_0^{\ell-1})$, does not depend on the summation indices and can be factored out. This leads us to our final three-factor update:

$$\begin{aligned}
\Delta\mathbf{W}^\ell &\propto \beta\odot\xi\\
\beta &= \nabla_{\boldsymbol{\theta}^\ell}f(\boldsymbol{\theta}^\ell,z_0^{\ell-1})\\
\xi &= \frac{2}{\sigma^2N(N-1)^2}\left(\sum_{p=1}^{-(N-1)}\left[\bar{k}(\mathbf{x}_p,\mathbf{x}_p)-\gamma\bar{k}(\mathbf{y}_p,\mathbf{y}_p)\right]\alpha(\mathbf{z}_p^\ell)+\right.\\
&\quad \left.\sum_{n=1}^{-(N-1)}\alpha(\mathbf{z}_n^\ell)\left[\bar{k}(\mathbf{x}_0,\mathbf{x}_0)-\gamma\bar{k}(\mathbf{y}_0,\mathbf{y}_0)\right]\right)\\
\alpha(\mathbf{z}_p^\ell) &= k(\mathbf{z}_0^\ell,\mathbf{z}_p^\ell)(\mathbf{z}_0^\ell-\mathbf{z}_p^\ell)
\end{aligned} \tag{7}$$

Note that β is now a local term that only depends on the current pre- and post-synaptic activity. ξ is a modulating term that adjusts the synaptic update layer-wise. This establishes a three-factor learning rule for Equation (5). In the next section, we discuss how ξ, despite appearing complex, is easy to compute using an auxiliary network of neurons.

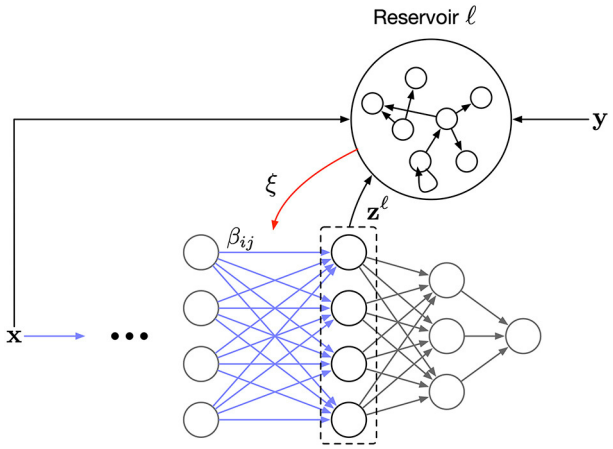


FIGURE 2
The overall network architecture. Each layer has a corresponding auxiliary reservoir network. The synaptic update, β in Equation (7), is modulated by a layer-wise error signal, ξ, that is the readout from the reservoir.

To understand the behavior of our rule, we begin by focusing on $\alpha(\mathbf{z}_p^\ell)$. This term drives together the layer representations $\mathbf{z}_0^\ell$ and $\mathbf{z}_p^\ell$, but the strength is weighted by the similarity kernel, $k(\mathbf{z}_0^\ell,\mathbf{z}_p^\ell)$. This similarity drive is modulated by the term $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)-\gamma\bar{k}(\mathbf{y}_p,\mathbf{y}_p)$. Note that we are specifically focused on the diagonal terms of the centered kernel matrices. This takes a special form:

$$\bar{k}(\mathbf{x}_p,\mathbf{x}_p)=1-\frac{1}{N}\sum_{n=0}^{-(N-1)}k(\mathbf{x}_p,\mathbf{x}_n)$$

The summation measures the average similarity of $\mathbf{x}_p$ to other samples. As a result, $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)$ acts as a "surprise" signal. If $\mathbf{x}_p$ is identical to all other samples, then $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)=0$. Conversely, if $\mathbf{x}_p$ is unlike any other sample, then $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)\to 1$. When taken together, the full term $\bar{k}(\mathbf{x}_p,\mathbf{x}_p)-\gamma\bar{k}(\mathbf{y}_p,\mathbf{y}_p)$ measures surprise along a decision boundary. If both the input and output is surprising or not surprising, this term goes to zero (where the relative surprise signals are balanced by γ). On the other hand, if the input is surprising, but the output is not, then the similarity drive of $\alpha(\mathbf{z}_p^\ell)$ is strengthened. In effect, this compresses away the differences in the input distribution and more closely matches the desired output. For the opposite case, when the input is not surprising, but the output is, the sign is reversed on $\alpha(\mathbf{z}_p^\ell)$. As expected, this forces the layer to drive the output representations apart.

2.2.1 Computing the modulating signal with an auxiliary network

In order to compute ξ in Equation (7), we require a neural circuit capable of storing information for future use. Recurrent networks can provide such functionality, and Sussillo and Abbott (2009) demonstrate how a reservoir network can be trained to compute complex signals using a local update rule.

For each layer, we construct an auxiliary network of rate-encoded leaky-integrate neurons whose dynamics are governed by

Equation (8)

$$\begin{aligned} \tau_r \frac{\mathrm{d}\mathbf{u}_r}{\mathrm{d}t} &= -\mathbf{u}_r + \lambda \mathbf{W}_r \mathbf{r} + \mathbf{W}_i \mathbf{r}_i + \mathbf{W}_{\mathrm{fb}} \mathbf{r}_o \\ \mathbf{r} &= \tanh(\mathbf{u}_r) \\ \mathbf{r}_o &= \mathbf{W}_o \mathbf{u}_r \end{aligned} \tag{8}$$

where $\mathbf{u}_r$ are the recurrent neuron membrane potentials, $\mathbf{r}_i$ is the input signal activity, and $\mathbf{r}_o$ is the readout activity. λ is a hyper-parameter that controls the chaos level of the recurrent population.

Following Sussillo and Abbott (2009), we train the auxiliary network using a least mean squares (LMS) FORCE learning rule shown below in Equation (9) (which is a local update),

$$\Delta \mathbf{W}_o \propto (\mathbf{r}_o - \boldsymbol{\xi}) \mathbf{r}^\top \tag{9}$$

where ξ is the *true* global error signal in Equation (7).

Training of the reservoir can be done *a priori* so that the reservoir is fixed during the primary learning phase. Alternatively, we can also compute ξ without needing to train the auxiliary network. Instead, a set of buffers can be constructed to remember that last N inputs, targets, and current layer's activity. This can be achieved using a delay line circuit where populations of neurons are connected with the appropriately chosen synaptic delay. Given the output of these buffers, computing ξ is trivial since $\bar{k}$, α, and the summation are all implementable functions using neurons.

Figure 2 illustrates the full design of the proposed learning scheme. The reservoir serves as a working memory where the capacity of the memory determines the effective batch size. To the best of our knowledge, our rule is the first to modulate the Hebbian updates of a synapse based on past information stored in a working memory. Furthermore, having a controllable effective batch size means we can study the effect of memory capacity on the learning convergence.

2.2.2 Extensions to convolutional neural networks and spiking neural networks

When feasible, we use fully-connected layers in this work, because they are biologically plausible. Unfortunately, the best performing artificial networks for visual tasks utilize convolution layers. A biologically plausible implementation of convolution itself is an open topic of research (Pogodin et al., 2021). Still, we can support convolutional layers in our rule, since the local term β in Equation (7) is agnostic to the layer function, f. The resulting update is not biologically plausible, but any issues stem from f, not the learning rule.

Similar to how we extend our rule to convolution neural networks, we can extend it to other network types including spiking neural networks. For spiking neural networks, the primary challenge is the non-linearity of the spike threshold function. Standard techniques such as surrogate gradients or probabilistic threshold functions can be used to correctly derive β (Neftci et al., 2019). Another important feature of SNNs is recurrence or feedback connections. Traditionally, this is handled by back-propagation-like algorithms by unrolling the network evaluations over time and treating the temporal dimension spatially. A similar approach could be done with our rule, but unlike back-propagation through time, the HSIC update does not introduce coupling across timesteps. In this way, we are able to remain biologically plausible even in the presence of recurrence. Yet, when dealing with recurrent connections, the stability of weight updates is always a concern. It is not immediately clear that our rule as given should be stable. The analysis and potential augmentations to our approach for recurrent networks is out of the scope of this paper, and we leave it for future work.

TABLE 2 Reservoir experiment parameters.

Parameter name	Symbol	Value
Simulation time step	Δt	1 ms
Neuron time constant	τ_r	5 ms
Sample time constant	Δt_{sample}	20 ms
Reservoir recurrent strength	λ	1.2
Effective batch size	N	10
HSIC balance parameter	γ	2
HSIC scale parameter	σ	0.5
Learning rate	η	5×10^{-4}
Num. of epochs	T	10

3 Results

We test our approach on a variety of synthetic and benchmark datasets. The code to reproduce each experiment is available at https://github.com/darsnack/biological-hsic/ along with instructions. Since our method processes samples one at a time, training networks can be a computationally intensive process. To make experimentation tractable, our evaluation is broken down into three stages:

1. We test the ability for a reservoir network to learn to compute the global modulatory signal, ξ, in Equation (7) as describe in Section 2.2.1.
2. We train multi-layer networks of rate-encoded leaky-integrate neurons on small synthetic datasets. We avoid simulating the reservoir for computational efficiency.
3. We train deep multi-layer networks of artificial neurons on larger scale machine learning benchmark datasets. We use artificial neurons instead of biological neurons for computational efficiency.

3.1 Reservoir experiments

First, we verify the ability for the reservoir to reproduce the true signal ξ in Equation (7). We use a recurrent population of 2,000 leaky-integrate neurons with $\tau_r = 5$ ms and $\lambda = 1.2$ (as described in Section 2.2.1). For the input and output signals, we use a hundred random samples from MNIST. Corresponding random hidden activation signals, $Z \in \mathbb{R}^{10 \times 100}$, are drawn from Unif(0, 1). Each sample is presented to the network for 10 ms and the network is trained for 10 epochs. For evaluation, we generate a hundred new inputs and process them with the network with all parameters fixed

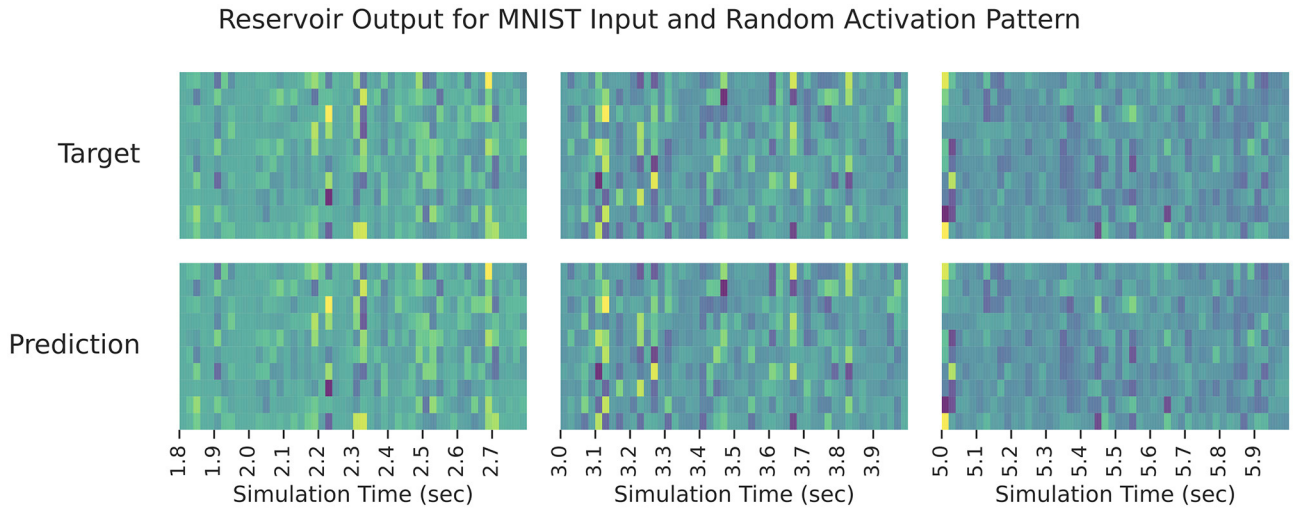


FIGURE 3
The reservoir output when learning ξ in Equation (7). Each sub-panel column displays the output during a different time interval of testing. The rows in a given sub-panel correspond to each element of true/predicted ξ. In all cases, the predicted output signal matches the target signal closely.

(i.e., no learning). A complete list of experimental parameters is in Table 2.

The results can be seen in Figure 3 which shows the reservoir output for all elements of the predicted modulatory signal against the target modulatory signal, ξ. The network is able to close match the target shortly after learning begins, and it is able to persist its performance long after learning stops. Note that this demonstrates how the auxiliary network can be pre-trained to compute ξ—it is not learning a data-specific computation, it is learning how to buffer N samples of its input and post-process them to compute ξ.

3.2 Small dataset experiments

Now, we show that our learning rule is capable of training multi-layer networks of leaky-integrate neurons to solve "small scale" tasks. For computational efficiency, we no longer simulate the reservoir network, but compute its readout, ξ, directly. All networks in this set of experiments use the following neuron model in Equation (10):

$$\tau_m \frac{\mathrm{d}\mathbf{u}^\ell}{\mathrm{d}t} = -\mathbf{u}^\ell + W^\ell \mathbf{z}^{\ell-1} + \mathbf{b}^\ell \qquad \mathbf{z}^\ell = \mathrm{relu}(\mathbf{u}^\ell) \tag{10}$$

where τ_m is the neuron time constant, and $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$ are parameters. A full description of the experimental parameters is in Table 3.

We consider two synthetic datasets consisting of a thousand samples each (a hundred samples are held out for testing). First, we generate a linearly separable set of labeled points drawn from $\mathrm{Unif}([0,1]\times[0,1])$ and train a network with a single hidden layer on them [shown in Figure 4 (top left)]. Since only a linear classifier is required to predict this dataset, we also generate a non-linear XOR dataset and train a network with two hidden layers on it [shown in Figure 4 (bottom left)]. The final layer of both networks is trained against a task-specific cross-entropy objective. The results on both datasets are shown in Figure 4. Within a few epochs, our rule is able to achieve nearly 100% accuracy on both datasets. Additionally, we also show the value of the HSIC bottleneck (Equation 5) in the rightmost column of Figure 4 during training. This demonstrates that our rule does reduce the HSIC bottleneck across all layers, and this reduction corresponds to an improvement in test accuracy on the task.

TABLE 3 Small dataset experiment parameters.

Parameter name	Symbol	Value
Simulation time step	Δt	1 ms
Neuron time constant	τ_m	5 ms
Sample time constant	Δt_{sample}	20 ms
Effective batch size	N	64
HSIC balance parameter	γ	10
HSIC input scale parameter	σ_x	0.3
HSIC layer scale parameter	σ_z	2
HSIC output scale parameter	σ_y	0.25
Learning rate	η	1×10^{-4}
Num. of epochs	N/A	25
Num. of random seeds	N/A	30

3.3 Large dataset experiments

Finally, we test our rule on two standard machine learning benchmark vision datasets—MNIST and CIFAR-10. We use

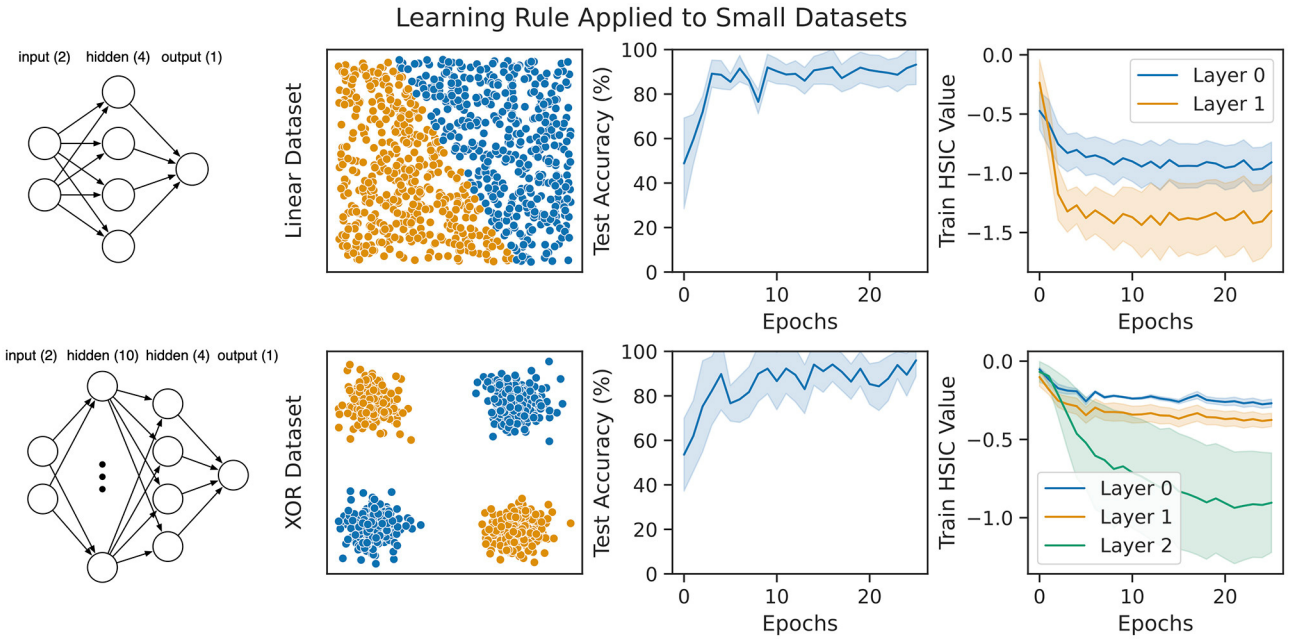


FIGURE 4
Our learning rule applied to two synthetic datasets with multi-layer feedforward networks of leaky-integrate neurons (see Equation 10). **(Top)** A linearly separable dataset drawn from Unif($[0, 1] \times [0, 1]$). The network trained on this dataset consists of a single hidden layer with four neurons. **(Bottom)** A non-linear XOR dataset with small Gaussian noise added around each input cluster. The network trained on this dataset consists of two hidden layers of size ten and four, respectively. For both datasets, our models converge to nearly 100% accuracy, and this coincides with decreasing the HSIC bottleneck at every layer.

TABLE 4 **Large scale experiment parameters.**

Parameter name	Symbol	MNIST value	CIFAR-10 value
Back-propagation learning rate	N/A	1×10^{-3}	5×10^{-4}
Our rule learning rate	N/A	1×10^{-4}	5×10^{-5}
Effective batch size	N	8	4
HSIC balance parameter	γ	2	50
HSIC input scale parameter	σ_x	0.5	1
HSIC layer scale parameter	σ_z	1	0.5
HSIC output scale parameter	σ_y	0.1	0.1
Num. of epochs	N/A	25	50
Num. of random seeds	N/A	30	15

feedforward networks of artificial neurons with a ReLU activation function (this is done to keep our experiments tractable). Our rule is compared against a back-propagation baseline.

We use a multi-layer fully-connected network on MNIST with architecture FC(128) $\rightarrow$ FC(64) $\rightarrow$ FC(10). For CIFAR-10, we use a convolutional neural network with architecture Conv($3 \times 3 \times 128$) $\rightarrow$ Avg. Pool(2) $\rightarrow$ Conv($3 \times 3 \times 256$) $\rightarrow$ Avg. Pool(2) $\rightarrow$ FC(10). Both the baseline and our rule are trained using an Adam optimizer with default hyper-parameters. A complete list of experiment parameters is in Table 4.

Figure 5 shows the test performance over the course of training. Even though learning occurs more slowly, our rule reaches comparable the performance with the back-propagation baseline. While the gap between back-propagation and our rule widens on CIFAR-10, it does reach 60% test accuracy which is higher than training just the last layer (this reaches only 39%). Given that each layer in our method has no explicit information about the performance of other layers, the fact that hierarchical learning is possible is remarkable.

3.4 Effects of memory capacity

One of the novel features of our rule is the ability to control the memory capacity of the update. To explore this parameter, we repeat the same CIFAR-10 experiments as before for various effective batch sizes. The results are shown in Figure 6. Not only does the final training performance increase as a function of batch size, the rate of learning also increases. Unfortunately, this improvement is logarithmic, leading to diminishing returns.

4 Discussion

In this work, we proposed a three-factor learning rule for training feedforward networks based on the information bottleneck

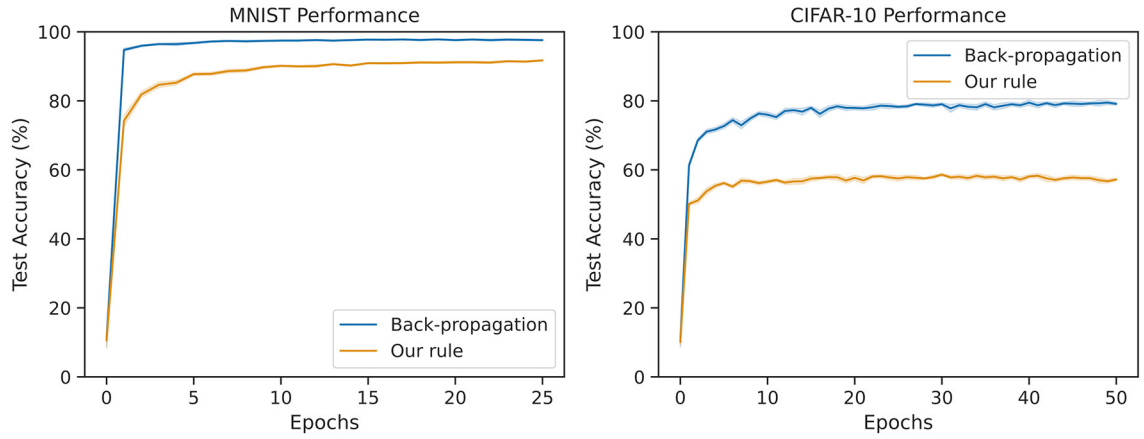


FIGURE 5
Average test accuracy over many trials on MNIST and CIFAR-10 for back-propagation and our method. The MNIST network is an MLP with 128 and 64 hidden neurons. The CIFAR-10 network is a CNN with 128 and 256 features followed by a single fully-connected output layer. Our rule reaches above 91% accuracy on MNIST (within 7% of the baseline). Our rule reaches around 61% accuracy on CIFAR-10 (within 19% of the baseline).

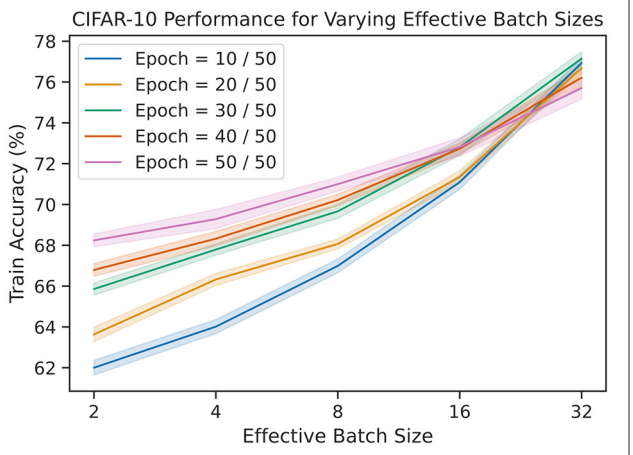


FIGURE 6
The train accuracies on CIFAR-10 for varying number of epochs and effective batch sizes. We see that accuracy improves logarithmically as a function of batch size.

principle. The rule is biologically plausible, and we are able to scale up to reasonable performance on MNIST. We do this by factoring our weight update into a local component and global component. The local component depends only on the current synaptic activity, so it can be implemented via Hebbian learning. In contrast to prior work, our global component uses information across many samples seen over time. We show that this content can be stored in an auxiliary reservoir network, and the readout of the reservoir can be used to modulate the local weight updates. To the best of our knowledge, this is the first biological learning rule to tightly couple the synaptic updates with a working memory capacity. We verified the efficacy of our rule on synthetic datasets, MNIST, and CIFAR-10, and we explored the effect of the size of the working memory capacity on the learning performance.

Even though our rule does perform reasonably well, there is room for improvement. The rule performs best when it is able to distinguish between different high dimensional inputs. The resolution at which it separates inputs is controlled by the parameter, σ, in the kernel function (Equation 4). The use of a fixed σ is partly responsible for the slow down in convergence in Figure 5. In Ma et al. (2019), the authors propose using multiple networks trained with the different values of σ and averaging the output across networks. This allows the overall network to separate the data at different resolutions. Future work can consider a population of networks with varying σ to achieve the same effect. Addressing the resolution issue will be important for improving the speed and scalability of the learning method.

Additionally, our rule is strongly supervised. While the mechanism for synaptic updates is biologically plausible, the overall learning paradigm is not. Note that the purpose of the label information in the global signal is to indicate whether the output for the current sample should be the same or different from previous samples. In other words, it might be possible to replace the term $\bar{k}(\mathbf{y}_0, \mathbf{y}_p)$ in Equation (7) with a binary teaching signal. This would allow the rule to operate under weak supervision. Alternatively, we could use contrastive learning, where output distribution, Y, is replaced by the output of a different network. Ideally, this other network should process a different, but related modality (e.g., a visual network and auditory network that are trained against each other using a contrastive approach).

Most importantly, while our rule is certainly biologically plausible, it remains to be seen if it is an accurate model for circuitry in the brain. Since rules based on the information bottleneck are relatively new, the corresponding experimental evidence must still be obtained. Yet, we note that our auxiliary reservoir serves a similar role to the "blackboard" circuit proposed in Mumford (1991). This circuit, present in the thalamus, receives projected connections from the visual cortex, similar to how each layer projects its output onto the reservoir. Furthermore, Mumford suggests that this circuit acts as a temporal buffer and sends signals

that capture information over longer timescales back to the cortex like our reservoir.

So, while it is uncertain whether our exact rule and memory circuit are present in biology, we suggest that an in-depth exploration of memory-modulated learning rules is necessary. Even in the absence of a biological counterpart, our rule captures important properties necessary for neuromorphic hardware—locality, asynchrony, and real-time processing. We achieve this by suggesting a fundamentally different objective training deep neural networks, in line with recent work. We hope this work prompts further exploration of novel, non-back-propagation-based approaches for learning.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: http://yann.lecun.com/exdb/mnist/.

Author contributions

KD derived the theory and learning rule and performed the simulations. ML helped design the experiments and provided feedback on the theory. KD and ML contributed to writing the manuscript. All authors contributed to the article and approved the submitted version.

Funding

This work was funded by the US Air Force Research Laboratory and the National Science Foundation.

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

References

Aceituno, P. V., Farinha, M. T., Loidl, R., and Grewe, B. F. (2023). Learning cortical hierarchies with temporal Hebbian updates. *Front. Comput. Neurosci.* 17:1136010. doi: 10.3389/fncom.2023.1136010

Ahmad, N., van Gerven, M., and Ambrogioni, L. (2020). GAIT-prop: a biologically plausible learning rule derived from backpropagation of error. *Adv. Neur. Inf. Process. Syst.* 33, 10913–10923.

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., and Tweed, D. B. (2019). Deep learning without weight transport. *Adv. Neur. Inf. Process. Syst.* 31, 9.

Balduzzi, D., Vanchinathan, H., and Buhmann, J. (2015). "Kickback cuts Backprop's red-tape: biologically plausible credit assignment in neural networks," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 7.

Belilovsky, E., Eickenberg, M., and Oyallon, E. (2019). Greedy layerwise learning can scale to ImageNet. *Proc. Mach. Learn. Res.* 97, 583–593. doi: 10.48550/arXiv.1812.11446

Bianchi, S., Muñoz-Martin, I., Covi, E., Bricalli, A., Piccolboni, G., Regev, A., et al. (2023). A self-adaptive hardware with resistive switching synapses for experience-based neurocomputing. *Nat. Commun.* 14:1565. doi: 10.1038/s41467-023-37097-5

Christensen, D. V., Dittmann, R., Linares-Barranco, B., Sebastian, A., Le Gallo, M., Redaelli, A., et al. (2022). 2022 roadmap on neuromorphic computing and engineering. *Neuromorp. Comp. Eng.* 2:022501. doi: 10.1088/2634-4386/ac4a83

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

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

Frémaux, N., and Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Front. Neural Circ.* 9:85. doi: 10.3389/fncir.2015.00085

Frenkel, C., Lefebvre, M., and Bol, D. (2021). Learning without feedback: fixed random learning signals allow for feedforward training of deep neural networks. *Front. Neurosci.* 15:629892. doi: 10.3389/fnins.2021.629892

Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. (2005). "Measuring statistical dependence with Hilbert-Schmidt Norms," in *Algorithmic Learning Theory, Vol. 3734*, eds. D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell (Berlin, Heidelberg: Springer Berlin Heidelberg), 63–77.

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*, 60. Available online at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

LeCun, Y., Cortes, C., and Burges, C. J. (1998). *MNIST Handwritten Digit Database*. Available online at: http://yann.lecun.com/exdb/mnist/ (accessed April 26, 2024).

Liao, Q., Leibo, J. Z., and Poggio, T. (2016). "How important is weight symmetry in backpropagation?," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 1837–1844.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support learning in deep neural networks. *arXiv* [preprint]. *arXiv:1411.0247 [cs, q-bio]*.

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

Loke, D., Skelton, J. M., Chong, T.-C., and Elliott, S. R. (2016). Design of a nanoscale, CMOS-integrable, thermal-guiding structure for boolean-logic and neuromorphic computation. *ACS Appl. Mater. Interf.* 8, 34530–34536. doi: 10.1021/acsami.6b10667

Ma, W.-D. K., Lewis, J. P., and Kleijn, W. B. (2019). The HSIC Bottleneck: deep learning without back-propagation. *arXiv* [preprint]. *arXiv:1908.01580 [cs, stat]*.

Meulemans, A., Farinha, M. T., Cervera, M., Sacramento, J., and Grewe, B. F. (2022). "Minimizing control for credit assignment with strong feedback," in *Proceedings of the 39th International Conference on Machine Learning* (Baltimore, MD: PMLR).

Mumford, D. (1991). On the computational architecture of the neocortex: I. The role of the thalamo-cortical loop. *Biol. Cybernet.* 65, 135–145. doi: 10.1007/BF00202389

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

Nøkland, A., and Eidnes, L. H. (2019). Training neural networks with local error signals. *Proc. Mach. Learn. Res.* 97, 4839–4850.

Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A., and Naud, R. (2021). Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nat. Neurosci.* 24, 1010–1019. doi: 10.1038/s41593-021-00857-x

Pogodin, R., and Latham, P. E. (2020). Kernelized information bottleneck leads to biologically plausible 3-factor Hebbian learning in deep networks. *Adv. Neural Inf. Process. Syst.* 33:12.

Pogodin, R., Lillicrap, T. P., Mehta, Y., and Latham, P. E. (2021). "Towards biologically plausible convolutional networks," in *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, 13.

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

Strubell, E., Ganesh, A., and McCallum, A. (2019). "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (Florence: Association for Computational Linguistics), 3645–3650.

Sussillo, D., and Abbott, L. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron* 63, 544–557. doi: 10.1016/j.neuron.2009.07.018

Wang, Q., Zhao, C., Sun, Y., Xu, R., Li, C., Wang, C., et al. (2023). Synaptic transistor with multiple biological functions based on metal-organic frameworks combined with the LIF model of a spiking neural network to recognize temporal information. *Microsyst. Nanoeng.* 9:96. doi: 10.1038/s41378-023-00566-4

Yan, X., Jia, X., Zhang, Y., Shi, S., Wang, L., Shao, Y., et al. (2023). A low-power Si:HfO2 ferroelectric tunnel memristor for spiking neural networks. *Nano Energy* 107:108091. doi: 10.1016/j.nanoen.2022.108091

Yang, S., and Chen, B. (2023a). Effective surrogate gradient learning with high-order information bottleneck for spike-based machine intelligence. *IEEE Transact. Neural Netw. Learn. Syst.* 1–15. doi: 10.1109/TNNLS.2023.3329525

Yang, S., and Chen, B. (2023b). SNIB: improving spike-based machine learning using nonlinear information bottleneck. *IEEE Transact. Syst. Man Cybernet.* 53, 7852–7863. doi: 10.1109/TSMC.2023.3300318

Yang, S., Tan, J., and Chen, B. (2022). Robust spike-based continual meta-learning improved by restricted minimum error entropy criterion. *Entropy* 24:455. doi: 10.3390/e24040455

Zenke, F., Bohté, S. M., Clopath, C., Comșa, I. M., Göltz, J., Maass, W., et al. (2021). Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron* 109, 571–575. doi: 10.1016/j.neuron.2021.01.009

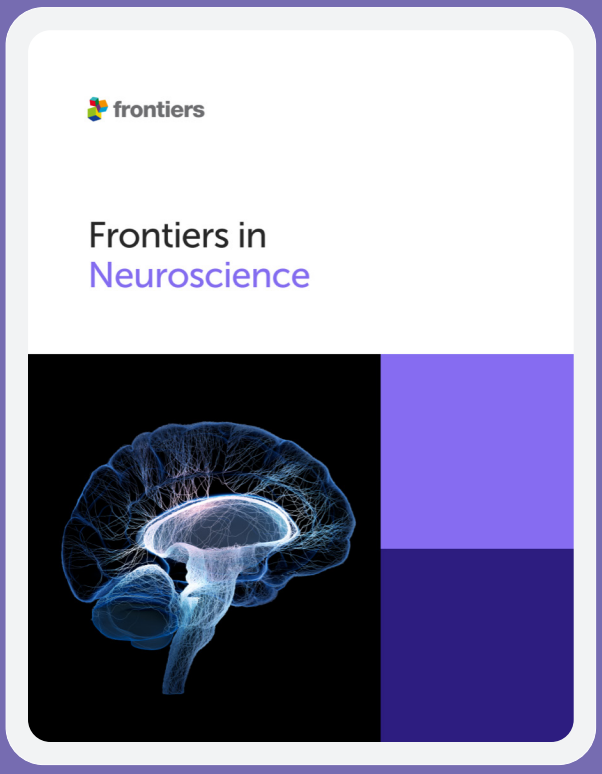
Frontiers in Neuroscience

Provides a holistic understanding of brain function from genes to behavior

Part of the most cited neuroscience journal series which explores the brain - from the new eras of causation and anatomical neurosciences to neuroeconomics and neuroenergetics.

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