Parallel Fixed-Point Spiking Neurons for Efficient Training of Spiking Neural Networks

Qirong Yang[1] **Guowei Peng**[1] **Xiurui Xie**[1] * **Yuning Yang**[1]
Qiugang Zhan[2] **Guisong Liu**[2]

[1] University of Electronic Science and Technology of China
[2] Southwestern University of Finance and Economics

Abstract

Spiking Neural Networks (SNNs) are inherently sequential due to temporal state dependencies, limiting the efficiency of parallel hardware such as GPUs and resulting in high training cost. In this work, we propose the Parallel Fixed-Point Spiking Neurons (PFSN), which reformulates the dynamics of leaky integrate-and-fire neurons into a unified fixed-point mapping, enabling parallel computation across all time steps. Unlike conventional sequential unrolling, the proposed formulation decouples temporal dependencies through iterative fixed-point updates, significantly improving computational efficiency. Furthermore, we introduce a learnable temporal propagation operator that generalizes predefined dynamics and allows adaptive modeling of task-specific temporal interactions without relying on explicit temporal recursion. Extensive experiments across diverse domains, including event-based recognition, sequential image classification, speech processing, and time-series forecasting, demonstrate that PFSN consistently achieves superior efficiency while maintaining or improving predictive performance compared to existing parallel SNN approaches. These results highlight the effectiveness of combining fixed-point formulations with learnable temporal structures for scalable and efficient SNN training. Code is available at `https://github.com/Ki-iong/PFSN`.

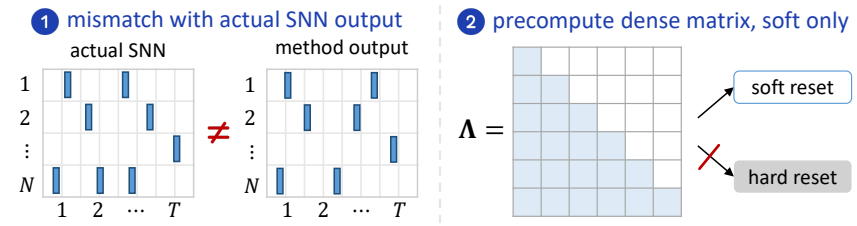
1 Introduction

Spiking Neural Networks (SNNs) have attracted increasing attention due to their event-driven nature and sparse computation paradigm, which offer significant advantages in energy-efficient processing [1, 2]. These properties make SNNs particularly suitable for deployment on resource-constrained edge devices and neuromorphic hardware [3]. Moreover, SNNs inherently incorporate temporal dynamics, enabling them to effectively capture time-dependent patterns in sequential and event-based data [4]. This capability has led to promising applications in areas such as dynamic vision sensing [5, 6], speech recognition [7, 8], and time-series modeling [9, 10].

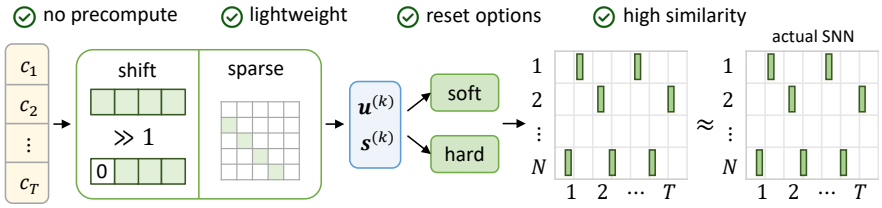
Despite these advantages, effectively leveraging the temporal modeling capability of SNNs remains challenging due to limited training efficiency [4, 11, 12]. In standard formulations such as the leaky integrate-and-fire (LIF) neuron, the membrane potential at each time step depends on the previous state, resulting in strictly sequential updates that enforce serial computation across time [13]. This inherent temporal dependency limits the utilization of modern parallel hardware such as GPUs [14], and the need to unroll the model over long temporal horizons further increases computational cost and reduces training efficiency [11]. To address this issue, recent works have explored parallelization strategies for SNN training. PSN [15] attempts to break temporal dependencies by introducing learnable propagation structures for parallel computation, but typically relax or omit the reset mechanism that is critical for nonlinear dynamics. In contrast, FPT [16] reformulates LIF dynamics

*Corresponding author (xiexiurui@uestc.edu.cn)

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Limitations of existing methods.



(b) Advantages of the proposed PFSN.

Figure 1: Comparison between existing parallel SNN methods and PFSN. Some existing methods deviate from the actual SNN dynamics, while others rely on precomputed dense matrices and are restricted to soft reset. PFSN employs lightweight shift or sparse propagation without dense precomputation, supports both soft and hard reset, and closely reproduces the outputs of sequential SNNs.

as a fixed-point problem [17] and accurately models soft-reset [18] behavior. However, it requires the precomputation of a decay matrix Λ, introducing additional computational overhead, and is primarily tailored to soft-reset dynamics, limiting its flexibility and general applicability.

To address these limitations, we propose Parallel Fixed-Point Spiking Neurons (PFSN), a unified fixed-point formulation derived from LIF dynamics with explicit reset, as illustrated in Fig. 1. PFSN represents temporal dependencies using a lightweight one-step propagation operator acting on reset membrane states, allowing the entire sequence of membrane states to be updated in parallel through fixed-point iteration. This formulation avoids dense decay-matrix precomputation and incorporates both soft and hard reset within the same framework, enabling efficient parallel computation while retaining reset-dependent nonlinear dynamics. Building on this formulation, we further introduce a learnable propagation operator to adapt temporal interactions to different tasks. The learnable variant operates on reset membrane states within the same fixed-point framework, improving modeling flexibility and performance while preserving the reset-aware structure and reducing reliance on iterative refinement. The main contributions of this paper are as follows:

- We propose PFSN, a parallel fixed-point formulation of spiking neurons, and provide theoretical analysis of its convergence and computational efficiency.
- We introduce a learnable propagation operator that generalizes fixed temporal dynamics and enables flexible modeling of task-specific temporal dependencies with reduced reliance on iterative refinement.
- We compare the simulation speed of PFSN with other parallel spiking neuron models and demonstrate its superior computational efficiency. We further evaluate the proposed approach on diverse tasks, including event-based data, sequential static image classification, speech classification, and time-series forecasting, demonstrating competitive performance compared to existing parallel spiking neuron models.

2 Related Work

2.1 Efficient Training of SNNs

Efforts to enhance SNN training efficiency primarily focus on temporal compression and gradient simplification. Timestep reduction is achieved through dynamic allocation [19], stage-wise shrinking [11], or stochastic latency mechanisms [20]. To bypass the costs of temporal backpropagation,

researchers have explored online gradient truncation [21, 22], single-step surrogate mapping [23], and temporal activation averaging [24]. Notably, LocalZO [25] employs zeroth-order optimization to eliminate surrogate gradients entirely, while T-RevSNN [26] minimizes overhead via reversible dynamics and selective neuron activation.

Parallelizing the inherently sequential spiking dynamics represents another critical research avenue. Some approaches exploit neuronal properties such as refractory periods to bound the effective sequential depth [27]. Another line of work reduces temporal dependencies by relaxing or even removing the reset mechanism. For instance, PSN [15] eliminates explicit reset, while SPSN [28] and PSU [29] further decouple state integration from spike generation, leading to simplified temporal interactions. Although these methods can improve computational efficiency, they typically deviate from the canonical spiking dynamics by weakening or discarding strict reset behaviors. In contrast, FPT [16] reformulates LIF dynamics as a fixed-point problem, providing a principled framework for parallelization while preserving neuronal fidelity.

2.2 Parallelization Strategies for Sequential Models

The inherent seriality of recurrent structures hinders their scalability on parallel hardware. To address this, early approaches replaced recurrent dynamics entirely with parallelizable convolutions [30, 31] or self-attention mechanisms [32]. For models retaining sequential dependencies, acceleration typically necessitates structural constraints, such as linearizing state computations to leverage Parallel Prefix Scan algorithms [33–37] or employing hierarchical sequence segmentation [38]. In the continuous-time domain, Neural ODEs achieve parallelism by adapting numerical techniques, including multiple shooting for segment-wise execution [39, 40] and log-signatures for coarser integration steps [41]. More recently, inspired by DEQ [42], sequential evaluation has been innovatively reframed as a fixed-point or root-finding problem, enabling more principled parallelization strategies.

2.3 Fixed-Point Methods in Neural Networks

In recent years, fixed-point solvers have attracted increasing attention in deep learning. DEQ [42] formulate neural networks as implicit systems with infinitely deep, weight-tied layers, where the forward pass is obtained by solving a fixed-point equation, and this paradigm has been extended to various tasks [43–45]. Building on this idea, recent works explore fixed-point formulations to accelerate sequential models. For instance, non-linear differential equations can be recast as fixed-point problems and solved using Newton-style iterations with quadratic convergence, enabling parallelizable updates even for sequential architectures such as RNNs and Neural ODEs [46]. In spiking neural networks, MPIS-SNNs [4] further connect neuron dynamics with fixed-point theory by modeling updates as weight-tied blocks across timesteps. A closely related approach is FPT [16], which reformulates LIF dynamics as a fixed-point iteration and reduces time complexity from $O(T)$ to $O(K)$. However, FPT incurs additional computational cost due to decay matrix precomputation and is tailored to soft-reset dynamics, which limits its generality.

3 Methods

In this section, we introduce PFSN, which reformulates LIF dynamics into a unified fixed-point form for parallel computation and supports efficient training with learnable operators.

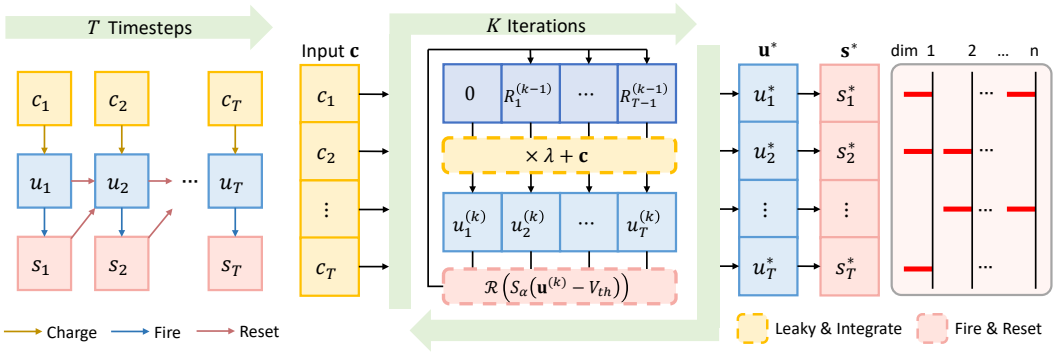
3.1 LIF Dynamics

LIF neuron is one of the most widely used models in SNNs due to its simplicity and biological plausibility. Formally, the LIF dynamics can be written as:

$$u_t = \lambda \mathcal{R}(u_{t-1}) + c_t \tag{1}$$

$$s_t = H(u_t - V_{\text{th}}) \tag{2}$$

where u_t is the membrane potential, c_t the input current, and $\lambda \in (0, 1]$ the leak factor. The spike $s_t \in 0, 1$ is generated by the Heaviside function $H(\cdot)$ with threshold V_{th}. The reset function $\mathcal{R}(\cdot)$ defines the post-spike update, where soft reset subtracts the threshold after firing and hard reset clears



(a) Leaky Integrate-and-Fire (b) Parallel Fixed-Point Spiking Neuron

Figure 2: Comparison between LIF and PFSN. LIF updates neuron states sequentially over T timesteps. In contrast, PFSN computes all timesteps in parallel by solving a fixed-point system with K iterations ($K \ll T$), and generates spikes from the converged membrane potentials.

the membrane potential:

$$\mathcal{R}(u_t) = \begin{cases} u_t - V_{\text{th}} s_t, & \text{soft reset} \\ (1 - s_t)\, u_t, & \text{hard reset} \end{cases} \tag{3}$$

By unfolding the recurrence over time, the dynamics can be expressed as a sequential process, as illustrated in Fig. 2(a):

$$\begin{cases} u_1 = c_1 \\ u_2 = \lambda \mathcal{R}(u_1) + c_2 \\ \quad \vdots \\ u_T = \lambda \mathcal{R}(u_{T-1}) + c_T \end{cases} \tag{4}$$

This formulation reveals a strict temporal dependency, where each state depends on its predecessor. In practice, SNNs are primarily trained on GPUs, such sequential dynamics prevent effective parallelization across time steps. As a result, both forward and backward propagation must be executed in a step-by-step manner, leading to underutilization of GPU parallelism and increased training time. This inherent inefficiency motivates the need for a parallelizable reformulation.

3.2 Fixed-Point Mapping

To enable efficient parallel training of SNNs, we reformulate the temporal dynamics into a vectorized representation by explicitly modeling all time steps simultaneously. Specifically, we denote the membrane potentials, spike outputs, and input currents across T time steps as

$$\mathbf{u} = [u_1, u_2, \ldots, u_T]^\top, \quad \mathbf{s} = [s_1, s_2, \ldots, s_T]^\top, \quad \mathbf{c} = [c_1, c_2, \ldots, c_T]^\top. \tag{5}$$

Based on the sequential formulation in Eq. (4), we use the above notations to rewrite the LIF dynamics into a parallel vectorized form:

$$\mathbf{u} = \lambda \mathcal{P}\big(\mathcal{R}(\mathbf{u})\big) + \mathbf{c}, \tag{6}$$

where $\mathcal{P}(\cdot)$ denotes a one-step temporal propagation operator. It can be implemented either by an explicit one-step shift or equivalently by a structured linear operator $\mathbf{P}$:

$$\mathbf{P}_{i,j} = \begin{cases} 1, & i = j + 1 \\ 0, & \text{otherwise} \end{cases}, \quad \text{s.t.} \quad (\mathbf{P}\,\mathcal{R}(\mathbf{u}))_t = \begin{cases} 0, & t = 1 \\ \mathcal{R}(u_{t-1}), & t > 1 \end{cases}. \tag{7}$$

With this operator formulation, the original sequential dynamics can be equivalently rewritten as a unified fixed-point mapping:

$$\Phi(\mathbf{u}) = \lambda \mathbf{P}\, \mathcal{R}(\mathbf{u}) + \mathbf{c}. \tag{8}$$

However, the Heaviside function $H(\cdot)$ is discontinuous, hindering the convergence of fixed-point iteration. We therefore replace it with a smooth surrogate and approximate the spike as $\mathbf{s} \approx$

$S_\alpha(\mathbf{u} - V_{\text{th}})$, where α controls the sharpness. Substituting this into the reset operator yields a differentiable approximation, leading to the surrogate fixed-point mapping:

$$\Phi(\mathbf{u}) \approx \hat{\Phi}_\alpha(\mathbf{u}) = \lambda \mathbf{P}\, \mathcal{R}_\alpha(\mathbf{u}) + \mathbf{c}, \tag{9}$$

where $\mathcal{R}_\alpha(\mathbf{u})$ denotes the reset function defined using the surrogate spike.

This formulation provides a unified perspective for modeling LIF neuron dynamics. By specifying different reset functions $\mathcal{R}(\cdot)$, neuron models with different reset mechanisms can be naturally recovered within the same framework.

3.3 Parallel Training

Forward Pass In the forward pass, we approximate the LIF dynamics via the iterative fixed-point mapping defined in Eq. (9). Starting from an initial state $\mathbf{u}_{(0)} = 0$, the membrane potential is iteratively updated as illustrated in Fig. 2(b). This process can be explicitly written as

$$\begin{cases} \mathbf{u}_{(1)} = \mathbf{c}, \\ \mathbf{u}_{(2)} = \lambda \mathbf{P}\, \mathcal{R}_\alpha(\mathbf{u}_{(1)}) + \mathbf{c}, \\ \quad \vdots \\ \mathbf{u}_{(K)} = \lambda \mathbf{P}\, \mathcal{R}_\alpha(\mathbf{u}_{(K-1)}) + \mathbf{c}, \end{cases} \tag{10}$$

$$\mathcal{R}_\alpha(\mathbf{u}_{(k)}) = \begin{cases} \mathbf{u}_{(k)} - V_{\text{th}} \cdot S_\alpha(\mathbf{u}_{(k)} - V_{\text{th}}), & \text{soft reset} \\ (\mathbf{1} - S_\alpha(\mathbf{u}_{(k)} - V_{\text{th}})) \odot \mathbf{u}_{(k)}, & \text{hard reset} \end{cases}, \quad 1 \le k \le K. \tag{11}$$

To ensure stable convergence, intermediate iterations employ a continuous surrogate function, whereas a hard threshold is applied at the stable state $\mathbf{u}^*$ to output discrete final spikes $\mathbf{s}^*$, ensuring that the model maintains consistent spiking semantics at inference time.

Backward Pass The backward pass follows the chain rule through the iterative process. Let $\mathcal{L}$ denote the loss function and $\mathbf{w}$ the network parameters. The gradient with respect to the parameters can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}^*} \frac{\partial \mathbf{s}^*}{\partial \mathbf{u}^*} \frac{\partial \mathbf{u}^*}{\partial \mathbf{w}}. \tag{12}$$

Here, $\frac{\partial \mathbf{s}^*}{\partial \mathbf{u}^*}$ is approximated using the surrogate gradient commonly adopted in SNN training. The remaining term $\frac{\partial \mathbf{u}^*}{\partial \mathbf{w}}$ can be computed recursively through the iterative updates. Specifically,

$$\begin{cases} \dfrac{\partial \mathbf{u}_{(1)}}{\partial \mathbf{w}} = \dfrac{\partial \mathbf{c}}{\partial \mathbf{w}}, \\ \dfrac{\partial \mathbf{u}_{(2)}}{\partial \mathbf{w}} = \lambda \mathbf{P} \dfrac{\partial \mathcal{R}_\alpha(\mathbf{u}_{(1)})}{\partial \mathbf{u}_{(1)}} \dfrac{\partial \mathbf{u}_{(1)}}{\partial \mathbf{w}} + \dfrac{\partial \mathbf{c}}{\partial \mathbf{w}}, \\ \quad \vdots \\ \dfrac{\partial \mathbf{u}_{(K)}}{\partial \mathbf{w}} = \lambda \mathbf{P} \dfrac{\partial \mathcal{R}_\alpha(\mathbf{u}_{(K-1)})}{\partial \mathbf{u}_{(K-1)}} \dfrac{\partial \mathbf{u}_{(K-1)}}{\partial \mathbf{w}} + \dfrac{\partial \mathbf{c}}{\partial \mathbf{w}}. \end{cases} \tag{13}$$

In this way, gradients are propagated sequentially along the fixed-point iterations, and the backward process can be efficiently implemented using automatic differentiation.

The detailed forward and backward propagation procedures are summarized in Appendix B.2.

3.4 Learnable Propagation Operator

While the fixed-point formulation in Eq. (9) enables a parallel representation of temporal dynamics, the temporal propagation operator $\lambda \mathbf{P}$ follows a predefined structure that limits its flexibility in modeling

diverse temporal dependencies. To improve expressiveness, we generalize $\lambda\mathbf{P}$ to a learnable matrix $\mathbf{A}$, enabling direct modeling of task-specific temporal interactions:

$$\hat{\Phi}_\alpha(\mathbf{u}) = \mathbf{A}\,\mathcal{R}_\alpha(\mathbf{u}) + \mathbf{c}. \tag{14}$$

In this formulation, $\mathbf{A}$ acts as a generalized temporal propagation operator. The original dynamics are recovered as a special case when $\mathbf{A} = \lambda\mathbf{P}$, indicating that the proposed formulation generalizes conventional temporal dynamics. Compared to the fixed structure $\lambda\mathbf{P}$, the learnable matrix $\mathbf{A}$ enables more flexible temporal interactions across time steps. With the introduction of learnable parameters, the model is no longer restricted to closely approximating the original LIF dynamics, thereby reducing the reliance on iterative refinement. In practice, the learnable PFSN can be efficiently implemented by initializing the membrane potential with $\mathbf{c}$ and performing only a single update step.

Different structural parameterizations of $\mathbf{A}$ correspond to different forms of temporal dependency modeling:

- **Diagonal shift structure**: constraining $\mathbf{A}$ to a single sub-diagonal yields a learnable counterpart of $\lambda\mathbf{P}$, where each time step depends only on its immediate predecessor.
- **Lower-triangular structure**: allowing $\mathbf{A}$ to be lower-triangular enables each time step to aggregate information from all previous states, capturing global temporal dependencies while preserving causality.
- **Fully dense structure**: an unconstrained $\mathbf{A}$ allows interactions across all time steps, including non-causal connections, providing maximal flexibility.

These variants span a spectrum from local to global temporal dependencies. In practice, the structure of $\mathbf{A}$ can be selected according to the task, enabling adaptation to different temporal patterns.

4 Theoretical Analysis

We provide theoretical analysis of the proposed PFSN, focusing on its convergence properties and computational efficiency.

4.1 Convergence Condition

The convergence of PFSN is governed by the contraction property of $\hat{\Phi}_\alpha$. Detailed proofs of the convergence theorem and the bounds for different reset mechanisms are provided in Appendix A. As shown in Theorem A.1, convergence is guaranteed if $\lambda L_{\mathcal{R}} < 1$. Since the delay operator $\mathbf{P}$ does not amplify perturbations, convergence is primarily determined by the reset mechanism.

For the soft-reset case, the reset operator leads to a clear contraction condition:

$$\lambda\left(1 + \frac{\alpha V_{\text{th}}}{4}\right) < 1. \tag{15}$$

In contrast, the hard-reset case yields a more conservative condition that depends on the membrane potential magnitude. Let $U_{\max}$ denote an upper bound of $\|\mathbf{u}\|_\infty$ along the fixed-point iteration. The contraction condition becomes

$$\lambda\left(1 + \frac{\alpha U_{\max}}{4}\right) < 1. \tag{16}$$

Here, $U_{\max}$ denotes an upper bound on the membrane potential magnitude during iteration. Although membrane potentials are typically regulated near V_{th} by leak and reset mechanisms, input currents can still drive them above threshold; thus, we keep $U_{\max}$ as a general bound. This conservative condition is supported in practice by hard reset, which suppresses residual membrane potentials and mitigates error accumulation. The empirical results in Sec. 5.1 further validate this convergence behavior.

Overall, these conditions reveal a trade-off between temporal memory and convergence stability. A larger leak factor λ strengthens temporal dependency but makes the contraction condition harder to satisfy, thus requiring a smaller surrogate slope α. Conversely, a smaller α improves stability but provides a smoother approximation to the hard spike function. We further provide experiments and discussions on how λ and α affect convergence in Appendix D.2.

4.2 Computational Efficiency

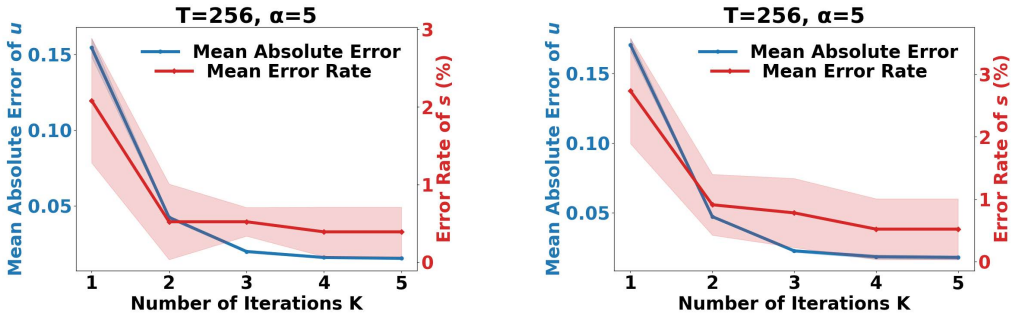
We first analyze the structural difference between PFSN and FPT. The temporal propagation in PFSN is highly structured and local, where each timestep interacts only with its immediate neighbors, resulting in a sparse propagation pattern. In contrast, FPT introduces a dense temporal operator that couples all timesteps, leading to global temporal interactions. This fundamental difference results in distinct computational characteristics between the two approaches.

From the perspective of practical computation, the sparse structure in PFSN allows the update to be implemented either via explicit shift operations or matrix multiplication. When implemented as a shift, the operation involves only element-wise data movement and scales linearly with the sequence length. Even when expressed in matrix form, the underlying operator remains highly sparse, leading to substantially reduced computation compared to dense operations. In contrast, FPT relies on a dense temporal operator, where matrix-vector multiplication involves all timesteps, typically resulting in quadratic complexity with respect to the sequence length. As a result, PFSN achieves lower effective computation and better efficiency in practice.

5 Experiments

We conduct comprehensive experiments to evaluate the computational efficiency and predictive performance of PFSN across diverse tasks. All experiments are implemented using the SpikingJelly framework [47], with detailed training configurations provided in Appendix C.

5.1 LIF Dynamics Simulation



(a) Hard reset

(b) Soft reset

Figure 3: Convergence of PFSN under different reset mechanisms. The mean absolute error (MAE) of membrane potentials and spike error rate decrease as the number of iterations increases, demonstrating convergence to the original LIF dynamics. Solid lines indicate the mean over three runs, and shaded regions denote the standard deviation.

We evaluate whether PFSN fixed-point iterations can reproduce sequential LIF dynamics. Specifically, we simulate a LIF neuron with $\lambda = 0.25$ and $V_{\text{th}} = 1$, using input currents sampled from a standard Gaussian distribution. As shown in Fig. 3, the membrane potential error and spike error rate decrease rapidly as the number of iterations increases, indicating that PFSN progressively approaches the original LIF trajectory. Both soft-reset and hard-reset variants exhibit stable convergence behavior, suggesting that the proposed fixed-point formulation preserves the essential temporal dynamics of LIF neurons. In practice, the approximation error becomes sufficiently small after only 2–3 iterations, showing that a small number of fixed-point updates is enough to obtain an accurate approximation. Therefore, we use a small iteration number in subsequent experiments to balance accuracy and computational efficiency. To complement this analysis with realistic, data-driven inputs, we provide controlled convergence experiments and task-level LIF-replacement evaluations on MNIST in the Appendix D.3.

Table 1: Comparison with existing spiking neuron models across various datasets.

Dataset	Method	Architecture	Timestep	Accuracy (%)
DVSGesture	LIF	VGG-11	20	95.83
	MPE-PSN [48]	7B-Net	16	97.92
	FPT [16]	VGG-11	20	**98.38 ± 0.17**
	PFSN-H w/ A (Ours)	VGG-11	20	98.23 ± 0.22
DVS-CIFAR10	LIF	VGG-11	10	84.10
	Sliding PSN [15]	VGG-11	10	85.90
	PSU / IPSU / RPSU [29]	VGGSNN	10	81.40 / 75.30 / 82.00
	MPE-PSN [48]	VGGSNN	8	86.69
	FPT [16]	VGG-11	10	85.50 ± 0.22
	PFSN-H w/ A (Ours)	VGG-11	10	**87.20 ± 0.44**
S-CIFAR10	LIF	ConvNet	32	81.00
	PSN [15]	ConvNet	32	88.45
	PSU / IPSU / RPSU [29]	ConvNet	32	79.52 / 87.28 / 84.44
	FPT [16]	ConvNet	32	89.53 ± 0.13
	PFSN-H w/ A (Ours)	ConvNet	32	**90.20 ± 0.08**
S-CIFAR100	LIF	ConvNet	32	55.76
	PSN [15]	ConvNet	32	62.21
	PSU / IPSU / RPSU [29]	ConvNet	32	51.65 / 59.76 / 57.89
	FPT [16]	ConvNet	32	64.50 ± 0.34
	PFSN-H w/ A (Ours)	ConvNet	32	**65.22 ± 0.15**
SHD	LIF	FCNet	15	**93.15**
	Sliding PSN [15]	FCNet	15	89.73
	PSU / IPSU / RPSU [29]	FCNet	15	90.64 / 91.30 / 92.49
	FPT [16]	FCNet	15	92.90 ± 0.51
	PFSN-H w/ A (Ours)	FCNet	15	93.06 ± 0.48

5.2 Computational Efficiency Comparison

We evaluate computational efficiency using the execution time ratio $t_{\text{LIF}}/t_{\text{model}}$, where t_{LIF} and t_{model} denote the total training and inference time of the LIF neuron and the compared model, respectively. For fair comparison, LIF is implemented using SpikingJelly, while parallel models use their official implementations. Experiments are conducted across varying time steps, with both FPT and PFSN configured to two iterations.

As shown in the Fig. 4, all parallel neuron models achieve consistent speedup over LIF, with gains increasing as the number of time steps grows, reflecting the removal of strict temporal dependency. Among them, PFSN achieves the best efficiency.

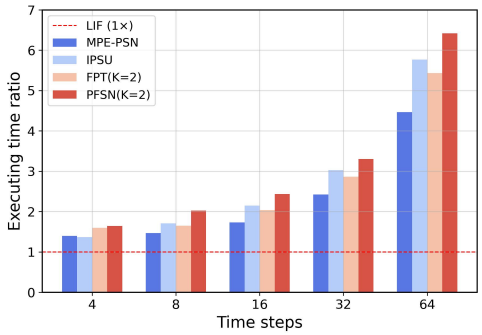


Figure 4: Executing time ratio of various parallel neurons across different time steps.

5.3 General Classification Performance

We evaluate the proposed PFSN on a diverse set of benchmarks, including neuromorphic datasets and sequential image classification tasks, to comprehensively assess its performance across different temporal dynamics and data modalities.

Event-based datasets. We evaluate PFSN on event-based datasets including DVSGesture [5] and DVS-CIFAR10 [49]. Following prior work, we adopt a modified VGG architecture from [50]. As shown in Tab. 1, PFSN achieves competitive or superior performance compared to existing parallel spiking neuron models. On DVSGesture, PFSN reaches 98.23%, comparable to the best result achieved by FPT. On the more challenging DVS-CIFAR10 dataset, PFSN achieves the highest

Table 2: Multivariate time series forecasting results on Metr-LA, Electricity, and Solar datasets.

Method	Metric	Metr-LA				Electricity				Solar				Avg.
		6	24	48	96	6	24	48	96	6	24	48	96	
FPT	$R^2 \uparrow$	0.847	0.603	0.381	0.232	0.951	0.922	0.910	0.868	0.965	0.885	0.785	0.728	0.756
	RSE $\downarrow$	0.413	0.665	0.829	0.924	0.278	0.351	0.376	0.455	0.188	0.340	0.465	0.523	0.484
	Time	85.6	91.1	103.7	119.0	66.8	72.6	80.4	95.7	116.9	120.6	126.9	124.1	101.7
PFSN	$R^2 \uparrow$	0.851	0.626	0.427	0.294	0.954	0.934	0.921	0.905	0.963	0.879	0.791	0.724	0.772
	RSE $\downarrow$	0.408	0.645	0.798	0.886	0.268	0.321	0.352	0.386	0.192	0.348	0.458	0.526	0.466
	Time	67.9	76.6	83.8	99.0	52.7	57.9	65.6	80.9	94.7	100.1	105.5	120.1	83.7

accuracy, outperforming PSN and FPT. These results demonstrate that PFSN effectively captures event-driven temporal dynamics.

Sequential CIFAR. We further evaluate PFSN on sequential image classification tasks, including S-CIFAR10 and S-CIFAR100, using the same network architecture as PSN [15] which is modified from [51]. PFSN consistently outperforms prior methods on both datasets, achieving 90.20% on S-CIFAR10 and 65.22% on S-CIFAR100. Compared to FPT and other parallel neuron models, PFSN demonstrates stronger capability in modeling long-range temporal dependencies in sequential inputs.

Spiking speech datasets. We also evaluate PFSN on the SHD dataset [7] for spiking speech recognition, using the network architecture from [52]. As shown in Tab. 1, PFSN achieves the best performance among all compared methods, reaching 93.06%. This result highlights the robustness of PFSN in handling temporal auditory signals and its effectiveness across different modalities.

5.4 Multivariate Time Series Forecasting

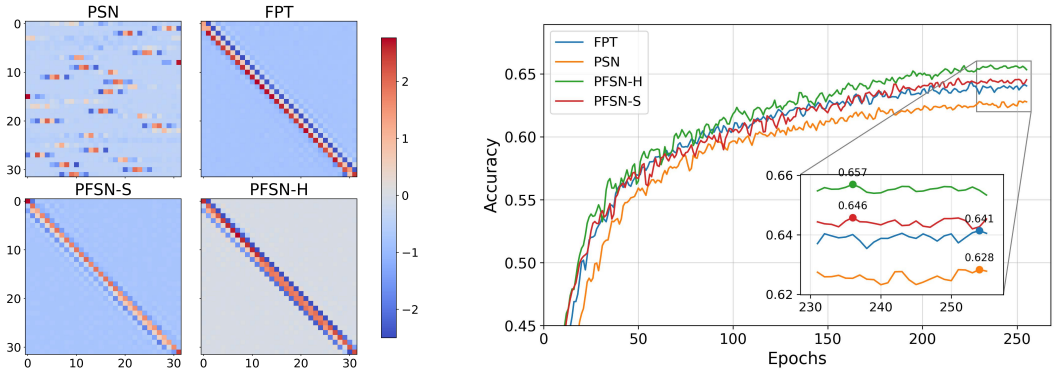
We extend our evaluation of PFSN to multivariate time series forecasting tasks using the Metr-LA [53], Electricity [54], and Solar [54] benchmarks across four prediction horizons (6, 24, 48, and 96). Consistent with prior work, experiments are conducted utilizing a modified iSpikformer architecture [9], with implementation details provided in the Appendix C.3. Performance is assessed via R^2, RSE, and training time (seconds per epoch), with results presented in Tab. 2.

As shown in Tab. 2, PFSN consistently matches or outperforms FPT while reducing computational cost. On average, it improves R^2 from 0.756 to 0.772 and reduces RSE from 0.484 to 0.466, while lowering training time from 101.7 to 83.7 seconds per epoch. The gains are more pronounced on Metr-LA at longer prediction horizons, where PFSN achieves higher accuracy. Similar trends are observed on Electricity, while on Solar, PFSN maintains comparable accuracy with reduced cost. Overall, these results indicate that PFSN preserves temporal modeling capacity while significantly improving efficiency.

5.5 Ablation Study

Effect of reset mechanisms. We analyze the impact of different reset mechanisms on temporal dynamics and classification performance on Sequential CIFAR100, as shown in Fig. 5. PSN, which lacks an explicit reset, achieves an accuracy of 62.8% but fails to learn a structured exponential decay pattern, instead showing an over-reliance on future temporal information. In contrast, FPT and PFSN-S adopt soft reset and produce smoother and more continuous temporal transitions, preserving local temporal dependencies, leading to improved accuracy of 64.1% and 64.6%, respectively. PFSN-H further introduces sharper truncation through hard reset, resulting in more localized temporal interactions and achieving the best performance of 65.7%. The introduction of reset mechanisms encourages the model to place greater emphasis on recent information rather than relying on distant temporal dependencies, with hard reset imposing a stronger form of temporal truncation.

Effect of learnable temporal propagation. We further evaluate the role of learnable temporal propagation, as summarized in Tab. 3. Without learnable parameters, PFSN performs comparably to LIF, indicating that the fixed formulation recovers standard neuron dynamics. Introducing a masked propagation matrix significantly improves performance, and further allowing the propagation matrix to be fully learnable yields the best results across both datasets. Compared with parameter-free PFSN,



(a) Temporal propagation matrices

(b) Test accuracy curves

Figure 5: Effect of reset mechanisms on temporal propagation and performance on S-CIFAR100.

Table 3: Effect of learnable temporal propagation on classification accuracy and training efficiency.

Dataset	Method	Accuracy (%)	Time/epoch (s)	Memory (GiB)
S-CIFAR10	LIF	81.00	64.38	1.043
	PFSN	80.91 $\pm$ 0.19	12.76	1.163
	PFSN w/ Masked $\mathbf{A}$	88.34 $\pm$ 0.24	13.34	1.274
	PFSN w/ $\mathbf{A}$	**90.20** $\pm$ **0.08**	12.91	1.274
S-CIFAR100	LIF	55.76	60.34	1.043
	PFSN	55.32 $\pm$ 0.35	12.06	1.163
	PFSN w/ Masked $\mathbf{A}$	63.53 $\pm$ 0.47	13.35	1.275
	PFSN w/ $\mathbf{A}$	**65.22** $\pm$ **0.15**	12.64	1.275

the learnable variants maintain similar per-epoch training times, with a modest increase in memory usage. These results demonstrate that adaptive temporal propagation enhances the model's ability to capture task-specific dependencies and improves classification performance while preserving training efficiency.

6 Conclusion

In this paper, we propose PFSN, a unified formulation that reformulates LIF dynamics as a fixed-point mapping to enable efficient parallel computation. By leveraging a structured temporal propagation operator, PFSN removes strict temporal dependencies and allows membrane potentials to be updated in parallel, thereby significantly improving training efficiency. We further introduce a learnable propagation mechanism that enhances the flexibility of temporal modeling, enabling the model to capture richer temporal dependencies and improving performance across diverse tasks. Theoretical analysis establishes convergence guarantees under mild conditions and provides insight into the stability and efficiency of the proposed formulation, offering advantages over existing fixed-point-based approaches. Extensive experiments on neuromorphic data, sequential image classification, speech recognition, and time-series forecasting show that PFSN consistently improves efficiency while achieving comparable or better accuracy than strong baselines. Overall, this work highlights the effectiveness of combining fixed-point formulations with learnable temporal structures, providing a scalable and efficient framework for spiking neural networks.

Acknowledgments and Disclosure of Funding

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant 62376228 and the Chengdu Science and Technology Program under Grant 2025-YF12-00009-RC. The authors declare no competing interests.

References

[1] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

[2] Nitin Rathi, Indranil Chakraborty, Adarsh Kosta, Abhronil Sengupta, Aayush Ankit, Priyadarshini Panda, and Kaushik Roy. Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware. *ACM Computing Surveys*, 55(12):1–49, 2023.

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee micro*, 38(1):82–99, 2018.

[4] Zhigao Cao, Meng Li, Xiashuang Wang, Haoyu Wang, Fan Wang, Youjun Li, and Zi-Gang Huang. Efficient training of spiking neural networks with multi-parallel implicit stream architecture. In *European Conference on Computer Vision*, pages 422–438. Springer, 2024.

[5] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

[6] Alex Vicente-Sola, Davide L Manna, Paul Kirkland, Gaetano Di Caterina, and Trevor J Bihl. Spiking neural networks for event-based action recognition: A new task to understand their advantage. *Neurocomputing*, 611:128657, 2025.

[7] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2744–2757, 2022. doi: 10.1109/TNNLS.2020.3044364.

[8] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in neuroscience*, 12:23, 2018.

[9] Changze Lv, Yansen Wang, Dongqi Han, Xiaoqing Zheng, Xuanjing Huang, and Dongsheng Li. Efficient and effective time-series forecasting with spiking neural networks. In *Proceedings of the 41st International Conference on Machine Learning*, pages 33624–33637, 2024.

[10] Changze Lv, Yansen Wang, Dongqi Han, Yifei Shen, Xiaoqing Zheng, Xuanjing Huang, and Dongsheng Li. Toward relative positional encoding in spiking transformers. *arXiv preprint arXiv:2501.16745*, 2025.

[11] Yongqi Ding, Lin Zuo, Mengmeng Jing, Pei He, and Yongjun Xiao. Shrinking your timestep: Towards low-latency neuromorphic object recognition with spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11811–11819, 2024.

[12] Zhengyang Zhuge, Peisong Wang, Xingting Yao, and Jian Cheng. Towards efficient spiking transformer: a token sparsification framework for training and inference acceleration. In *Forty-first International Conference on Machine Learning*, 2024.

[13] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

[14] Rajat Raina, Anand Madhavan, Andrew Y Ng, et al. Large-scale deep unsupervised learning using graphics processors. In *Icml*, volume 9, pages 873–880, 2009.

[15] Wei Fang, Zhaofei Yu, Zhaokun Zhou, Ding Chen, Yanqi Chen, Zhengyu Ma, Timothée Masquelier, and Yonghong Tian. Parallel spiking neurons with high efficiency and ability to learn long-term dependencies. *Advances in Neural Information Processing Systems*, 36:53674–53687, 2023.

[16] Wanjin Feng, Xingyu Gao, Wenqian Du, Hailong Shi, Peilin Zhao, Pengcheng Wu, and Chunyan Miao. Efficient parallel training methods for spiking neural networks with constant time complexity. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 16711–16726. PMLR, 13–19 Jul 2025.

[17] Ravi P Agarwal, Maria Meehan, and Donal O'regan. *Fixed point theory and applications*, volume 141. Cambridge university press, 2001.

[18] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13558–13567, 2020.

[19] Chen Li, Edward G Jones, and Steve Furber. Unleashing the potential of spiking neural networks with dynamic confidence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13350–13360, 2023.

[20] Srinivas Anumasa, Bhaskar Mukhoty, Velibor Bojkovic, Giulia De Masi, Huan Xiong, and Bin Gu. Enhancing training of spiking neural network with stochastic latency. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10900–10908, 2024.

[21] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. *Advances in neural information processing systems*, 35:20717–20730, 2022.

[22] Haiyan Jiang, Giulia De Masi, Huan Xiong, and Bin Gu. Ndot: neuronal dynamics-based online training for spiking neural networks. In *Forty-first International Conference on Machine Learning*, 2024.

[23] Kazuma Suetake, Shin-ichi Ikegawa, Ryuji Saiin, and Yoshihide Sawada. S3nn: Time step reduction of spiking surrogate gradients for training energy efficient single-step spiking neural networks. *Neural Networks*, 159:208–219, 2023.

[24] Zijian Wang, Peng Tao, and Luonan Chen. Brain-inspired chaotic spiking backpropagation. *National Science Review*, 11(6):nwae037, 2024.

[25] Bhaskar Mukhoty, Velibor Bojkovic, William De Vazelhes, Xiaohan Zhao, Giulia De Masi, Huan Xiong, and Bin Gu. Direct training of snn using local zeroth order method. *Advances in Neural Information Processing Systems*, 36:18994–19014, 2023.

[26] Jiakui Hu, Man Yao, Xuerui Qiu, Yuhong Chou, Yuxuan Cai, Ning Qiao, Yonghong Tian, Bo Xu, and Guoqi Li. High-performance temporal reversible spiking neural networks with $\mathcal{O}(L)$ training memory and $\mathcal{O}(1)$ inference cost. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 19516–19530. PMLR, 21–27 Jul 2024.

[27] Luke Taylor, Andrew King, and Nicol S Harper. Addressing the speed-accuracy simulation trade-off for adaptive spiking neurons. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 59360–59374. Curran Associates, Inc., 2023.

[28] Sidi Yaya Arnaud Yarga and Sean UN Wood. Accelerating snn training with stochastic parallelizable spiking neurons. In *2023 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2023.

[29] Yang Li, Yinqian Sun, Xiang He, Yiting Dong, Dongcheng Zhao, and Yi Zeng. Parallel spiking unit for efficient training of spiking neural networks. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2024.

[30] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[31] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[33] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39):851–876, 2007.

[34] Tao Lei, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 4470–4481, 2018.

[35] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. *arXiv preprint arXiv:1709.04057*, 2017.

[36] Feiqing Huang, Kexin Lu, Yuxi CAI, Zhen Qin, Yanwen Fang, Guangjian Tian, and Guodong Li. Encoding recurrence into transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=7YfHla7IxBJ`.

[37] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International conference on machine learning*, pages 26670–26698. PMLR, 2023.

[38] Yi Luo, Zhuo Chen, and Takuya Yoshioka. Dual-path rnn: Efficient long sequence modeling for time-domain single-channel speech separation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 46–50. IEEE, 2020.

[39] Stefano Massaroli, Michael Poli, Sho Sonoda, Taiji Suzuki, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Differentiable multiple shooting layers. *Advances in Neural Information Processing Systems*, 34: 16532–16544, 2021.

[40] Stefanie Gunther, Lars Ruthotto, Jacob B Schroder, Eric C Cyr, and Nicolas R Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020.

[41] James Morrill, Cristopher Salvi, Patrick Kidger, and James Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[42] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[43] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *Advances in neural information processing systems*, 33:5238–5250, 2020.

[44] Juncheng Liu, Bryan Hooi, Kenji Kawaguchi, and Xiaokui Xiao. Mgnni: Multiscale graph neural networks with implicit layers. *Advances in Neural Information Processing Systems*, 35:21358–21370, 2022.

[45] Zhichun Huang, Shaojie Bai, and J Zico Kolter. Implicit2: Implicit layers for implicit representations. *Advances in neural information processing systems*, 34:9639–9650, 2021.

[46] Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. In *The Twelfth International Conference on Learning Representations*, 2024.

[47] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023. doi: 10.1126/sciadv.adi1480. URL `https://www.science.org/doi/abs/10.1126/sciadv.adi1480`.

[48] Hanqi Chen, Lixing Yu, Shaojie Zhan, Penghui Yao, and Jiankun Shao. Time-independent spiking neuron via membrane potential estimation for efficient spiking neural networks. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2025.

[49] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017.

[50] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=_XNtisL32jv`.

[51] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2661–2671, October 2021.

[52] Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *ICLR*, 2024.

[53] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[54] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

A Convergence Proof for PFSN

Theorem A.1 (Contraction of PFSN Mapping)**.** *Assume the reset operator $\mathcal{R}_\alpha(\cdot)$ is Lipschitz continuous under the ℓ_1-norm with constant $L_\mathcal{R}$. If*

$$\lambda L_\mathcal{R} < 1, \tag{17}$$

where $0 < \lambda < 1$, then the mapping $\hat{\Phi}_\alpha(\mathbf{u}) = \lambda \mathbf{P}\mathcal{R}_\alpha(\mathbf{u}) + \mathbf{c}$ is a contraction under the ℓ_1-norm. Consequently, the iteration $\mathbf{u}^{(k)} = \hat{\Phi}_\alpha(\mathbf{u}^{(k-1)})$ converges to a unique fixed point $\mathbf{u}^*$.

Proof. Let $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^T$. Then

$$\|\hat{\Phi}_\alpha(\mathbf{u}_1) - \hat{\Phi}_\alpha(\mathbf{u}_2)\|_1 = \|\lambda \mathbf{P}\mathcal{R}_\alpha(\mathbf{u}_1) + \mathbf{c} - (\lambda \mathbf{P}\mathcal{R}_\alpha(\mathbf{u}_2) + \mathbf{c})\|_1 \tag{18}$$

$$= \|\lambda \mathbf{P}(\mathcal{R}_\alpha(\mathbf{u}_1) - \mathcal{R}_\alpha(\mathbf{u}_2))\|_1. \tag{19}$$

Since $0 < \lambda < 1$ and the ℓ_1-norm is submultiplicative, we have $\|\lambda \mathbf{P}\mathbf{x}\|_1 \leq \lambda \|\mathbf{P}\|_1 \|\mathbf{x}\|_1$, where $\|\mathbf{P}\|_1 = 1$ as each column of $\mathbf{P}$ contains at most one nonzero entry equal to 1. Thus,

$$\|\hat{\Phi}_\alpha(\mathbf{u}_1) - \hat{\Phi}_\alpha(\mathbf{u}_2)\|_1 \leq \lambda \|\mathbf{P}\|_1 \|\mathcal{R}_\alpha(\mathbf{u}_1) - \mathcal{R}_\alpha(\mathbf{u}_2)\|_1 \tag{20}$$

$$= \lambda \|\mathcal{R}_\alpha(\mathbf{u}_1) - \mathcal{R}_\alpha(\mathbf{u}_2)\|_1. \tag{21}$$

If $\mathcal{R}_\alpha$ is Lipschitz continuous with constant $L_\mathcal{R}$, then

$$\|\mathcal{R}_\alpha(\mathbf{u}_1) - \mathcal{R}_\alpha(\mathbf{u}_2)\|_1 \leq L_\mathcal{R} \|\mathbf{u}_1 - \mathbf{u}_2\|_1. \tag{22}$$

Substituting this into the above inequality yields

$$\|\hat{\Phi}_\alpha(\mathbf{u}_1) - \hat{\Phi}_\alpha(\mathbf{u}_2)\|_1 \leq \lambda L_\mathcal{R} \|\mathbf{u}_1 - \mathbf{u}_2\|_1. \tag{23}$$

By the Banach fixed-point theorem, if $\lambda L_\mathcal{R} < 1$, the mapping $\hat{\Phi}_\alpha$ is a contraction under the ℓ_1-norm and the iteration converges to a unique fixed point. □

Proposition A.1 (Lipschitz bound for soft reset)**.** *Assume the surrogate spike function is defined as $S_\alpha(x) = \sigma(\alpha x)$, where $\sigma(\cdot)$ is the sigmoid function. For the soft-reset operator $\mathcal{R}_\alpha^{\text{soft}}(\mathbf{u}) = \mathbf{u} - V_{\text{th}} S_\alpha(\mathbf{u} - V_{\text{th}})$, the mapping $\hat{\Phi}_\alpha$ is a contraction if*

$$\lambda \left(1 + \frac{\alpha V_{\text{th}}}{4}\right) < 1. \tag{24}$$

Proof. Since $S_\alpha(x) = \sigma(\alpha x)$, its derivative satisfies

$$S'_\alpha(x) = \alpha \sigma(\alpha x)(1 - \sigma(\alpha x)) \leq \frac{\alpha}{4}, \tag{25}$$

implying that S_α is Lipschitz continuous with constant $L_\alpha = \frac{\alpha}{4}$.

Let $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^T$. Then

$$\|\mathcal{R}_\alpha^{\text{soft}}(\mathbf{u}_1) - \mathcal{R}_\alpha^{\text{soft}}(\mathbf{u}_2)\|_1 = \|\mathbf{u}_1 - \mathbf{u}_2 - V_{\text{th}}\big(S_\alpha(\mathbf{u}_1 - V_{\text{th}}) - S_\alpha(\mathbf{u}_2 - V_{\text{th}})\big)\|_1 \tag{26}$$

$$\leq \|\mathbf{u}_1 - \mathbf{u}_2\|_1 + V_{\text{th}} \|S_\alpha(\mathbf{u}_1 - V_{\text{th}}) - S_\alpha(\mathbf{u}_2 - V_{\text{th}})\|_1 \tag{27}$$

$$\leq \left(1 + \frac{\alpha V_{\text{th}}}{4}\right) \|\mathbf{u}_1 - \mathbf{u}_2\|_1. \tag{28}$$

□

Proposition A.2 (Lipschitz bound for hard reset)**.** *Assume the surrogate spike function is defined as $S_\alpha(x) = \sigma(\alpha x)$, where $\sigma(\cdot)$ is the sigmoid function. For the hard-reset operator $\mathcal{R}_\alpha^{\text{hard}}(\mathbf{u}) = (\mathbf{1} - S_\alpha(\mathbf{u} - V_{\text{th}})) \odot \mathbf{u}$, the mapping $\hat{\Phi}_\alpha$ is a contraction if*

$$\lambda \left(1 + \frac{\alpha \|\mathbf{u}\|_\infty}{4}\right) < 1. \tag{29}$$

Proof. Using the same bound as above, S_α is Lipschitz continuous with constant $L_\alpha = \frac{\alpha}{4}$.

Let $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^T$. Then

$$\begin{aligned}
\|\mathcal{R}_\alpha^{\mathrm{hard}}(\mathbf{u}_1) - \mathcal{R}_\alpha^{\mathrm{hard}}(\mathbf{u}_2)\|_1 &= \|(\mathbf{1} - S_\alpha(\mathbf{u}_1 - V_{\mathrm{th}})) \odot \mathbf{u}_1 - (\mathbf{1} - S_\alpha(\mathbf{u}_2 - V_{\mathrm{th}})) \odot \mathbf{u}_2\|_1 && (30)\\
&\le \|(\mathbf{1} - S_\alpha(\mathbf{u}_1 - V_{\mathrm{th}})) \odot (\mathbf{u}_1 - \mathbf{u}_2)\|_1 && (31)\\
&\quad + \|(S_\alpha(\mathbf{u}_2 - V_{\mathrm{th}}) - S_\alpha(\mathbf{u}_1 - V_{\mathrm{th}})) \odot \mathbf{u}_2\|_1 && (32)\\
&\le \|\mathbf{u}_1 - \mathbf{u}_2\|_1 + \frac{\alpha}{4}\|\mathbf{u}_2\|_\infty \|\mathbf{u}_1 - \mathbf{u}_2\|_1 && (33)\\
&\le \left(1 + \frac{\alpha\|\mathbf{u}\|_\infty}{4}\right) \|\mathbf{u}_1 - \mathbf{u}_2\|_1. && (34)
\end{aligned}$$

□

B Implementation of PFSN

B.1 Fixed-Point Iteration

As mentioned in the main text, the temporal propagation operator $\mathcal{P}(\cdot)$ can be implemented in multiple equivalent ways. In particular, the main text discusses two representative implementations:

- **Shift-based implementation.** The most direct implementation is to shift the reset membrane potential by one step along the temporal dimension, which achieves linear-time propagation without additional matrix storage.
- **Matrix-based implementation.** The same operation can be written as multiplication with a delay matrix $\mathbf{P}$, i.e., $\mathcal{P}(\mathbf{x}) = \mathbf{P}\mathbf{x}$. This form is convenient for mathematical derivation and well suited for parallel computation on GPUs.

Beyond the two implementations discussed in the main text, $\mathcal{P}$ can also be realized by a one-dimensional causal convolution. Specifically, by padding one zero at the beginning of the temporal sequence and applying a fixed kernel $\boldsymbol{\kappa} = [1, 0]^\top$ to each local temporal window, we have

$$\mathcal{P}(\mathbf{x})_t = \boldsymbol{\kappa}^\top \begin{bmatrix} \tilde{x}_t \\ \tilde{x}_{t+1} \end{bmatrix}, \qquad \tilde{\mathbf{x}} = [0, x_1, x_2, \ldots, x_T], \qquad \boldsymbol{\kappa} = [1, 0]^\top. \tag{35}$$

This operation copies $\tilde{x}_t = x_{t-1}$ to the current time step, thereby realizing the same one-step temporal propagation as $\mathcal{P}(\cdot)$.

B.2 Forward and Backward Training

Algorithm 1 Forward Pass of PFSN

Require: Input current $\mathbf{c}$, threshold potential V_{th}, decay factor λ, iterations K
Ensure: Membrane potentials $\mathbf{u}^*$, spike outputs $\mathbf{s}^*$

Initialize $\mathbf{u}^{(0)} = \mathbf{0}$ and $\mathbf{s}^{(0)} = \mathbf{0}$
for $k = 1$ to K **do**
  Compute reset membrane potential: $\mathbf{r}^{(k-1)} = \mathcal{R}_\alpha\big(\mathbf{u}^{(k-1)}\big)$
  Update membrane potential: $\mathbf{u}^{(k)} = \lambda\mathcal{P}\big(\mathbf{r}^{(k-1)}\big) + \mathbf{c}$
  Compute surrogate spike outputs: $\mathbf{s}^{(k)} = S_\alpha\big(\mathbf{u}^{(k)} - V_{\mathrm{th}}\big)$
end for
Set equilibrium membrane potential: $\mathbf{u}^* = \mathbf{u}^{(K)}$
Compute final spike outputs: $\mathbf{s}^* = H\big(\mathbf{u}^* - V_{\mathrm{th}}\big)$

Alg. 1 summarizes the forward procedure of PFSN. In the forward pass, we unroll the fixed-point iteration for K steps. At each iteration, the reset state is first computed as $\mathbf{r}^{(k-1)} = \mathcal{R}_\alpha(\mathbf{u}^{(k-1)})$, and the membrane potential is then updated by propagating this reset state through $\mathcal{P}$. After K iterations, the final membrane potential is taken as the equilibrium estimate, from which the spike output is generated.

Algorithm 2 Backward Pass of PFSN

Require: Input current $\mathbf{c}$, threshold potential V_{th}, decay factor λ, iterations K
Ensure: Gradient of the spike output with respect to network weights, $\frac{\partial \mathbf{s}^*}{\partial \mathbf{w}}$

1: Compute the derivative of the final spike output: $\frac{\partial \mathbf{s}^*}{\partial \mathbf{u}^*} \leftarrow \frac{\partial S_\alpha\left(\mathbf{u}^{(K)}-V_{\text{th}}\right)}{\partial \mathbf{u}^{(K)}}$.
2: Initialize the membrane gradient: $\frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{w}} \leftarrow \frac{\partial \mathbf{c}}{\partial \mathbf{w}}$.
3: **for** $k = 2$ to K **do**
4: Compute the derivative of the reset membrane potential:

$$\frac{\partial \mathbf{r}^{(k-1)}}{\partial \mathbf{u}^{(k-1)}} \leftarrow \begin{cases} \mathbf{I} - V_{\text{th}} \dfrac{\partial S_\alpha\left(\mathbf{u}^{(k-1)} - V_{\text{th}}\right)}{\partial \mathbf{u}^{(k-1)}}, & \text{soft reset,} \\ \mathbf{I} - \operatorname{diag}\left(\mathbf{s}^{(k-1)}\right) - \operatorname{diag}\left(\mathbf{u}^{(k-1)}\right) \dfrac{\partial S_\alpha\left(\mathbf{u}^{(k-1)} - V_{\text{th}}\right)}{\partial \mathbf{u}^{(k-1)}}, & \text{hard reset.} \end{cases}$$

5: Update the membrane gradient:

$$\frac{\partial \mathbf{u}^{(k)}}{\partial \mathbf{w}} \leftarrow \lambda \mathbf{P} \frac{\partial \mathbf{r}^{(k-1)}}{\partial \mathbf{u}^{(k-1)}} \frac{\partial \mathbf{u}^{(k-1)}}{\partial \mathbf{w}} + \frac{\partial \mathbf{c}}{\partial \mathbf{w}}.$$

6: **end for**
7: Compute the final gradient: $\frac{\partial \mathbf{s}^*}{\partial \mathbf{w}} \leftarrow \frac{\partial \mathbf{s}^*}{\partial \mathbf{u}^*} \frac{\partial \mathbf{u}^{(K)}}{\partial \mathbf{w}}$.

Alg. 2 summarizes the backward training procedure of PFSN. Gradients are propagated through the unrolled iterations using the chain rule. Since the spike function is non-differentiable, its derivative is approximated by the surrogate function $S_\alpha(\cdot)$. The derivative of the reset operation is instantiated according to the reset mechanism, covering both soft-reset and hard-reset cases.

C Experimental Details

C.1 Environments for Computational Efficiency Comparison

The computational efficiency comparison is conducted in a Docker image with Python 3.10 on Ubuntu 22.04. The software environment uses PyTorch 2.1.0 and CUDA 12.1. All experiments are performed on a server equipped with one NVIDIA RTX 3090 GPU with 24GB memory and 20 vCPUs based on the AMD EPYC 7642 48-Core Processor.

C.2 Training Hyper-parameters

Table 4: Training hyper-parameters used for different datasets.

Hyper-parameter	DVS-Gesture	DVS-CIFAR10	S-CIFAR10	S-CIFAR100	SHD
Epoch	200	200	256	256	100
Batch size	16	16	128	128	256
T	20	10	32	32	15
λ	0.5	0.5	0.5	0.5	0.5
α	1	1	1	1	1
Reset	hard	hard	hard	hard	hard
Learnable	✓	✓	✓	✓	✓
Learning rate	0.01	0.01	0.1	0.1	0.001
Optimizer	SGD	SGD	SGD	SGD	Adam

Tab. 4 summarizes the main training hyper-parameters used for the General Classification Performance experiments. For all datasets, we adopt the hard-reset mechanism and enable the learnable propagation operator. The decay factor is fixed to $\lambda = 0.5$ across all tasks for a fair comparison. Other hyper-parameters, including the number of training epochs, batch size, simulation timestep T, learning rate, and optimizer, are selected according to the dataset scale and task characteristics.

Table 5: Statistics of the multivariate time series forecasting datasets.

Dataset	Metr-LA	Electricity	Solar
Samples	34,272	26,304	52,560
Variables	207	321	137
Length	12	168	168
Train/Valid/Test	$(0.7, 0.2, 0.1)$	$(0.6, 0.2, 0.2)$	$(0.6, 0.2, 0.2)$

For multivariate time series forecasting, we evaluate PFSN on Metr-LA, Electricity, and Solar, with dataset statistics summarized in Tab. 5. To ensure consistency across datasets, we adopt a unified model and optimization configuration for all forecasting experiments. The hidden dimension is set to 512, the feed-forward dimension to 1024, the network depth to 2, and the number of attention heads to 8. The model is trained with the Adam optimizer using a learning rate of 1×10^{-4}, and mean squared error (MSE) is used as the training loss. The spiking-neuron-related settings are detailed in Appendix C.3.

C.3 Modified iSpikformer Architecture

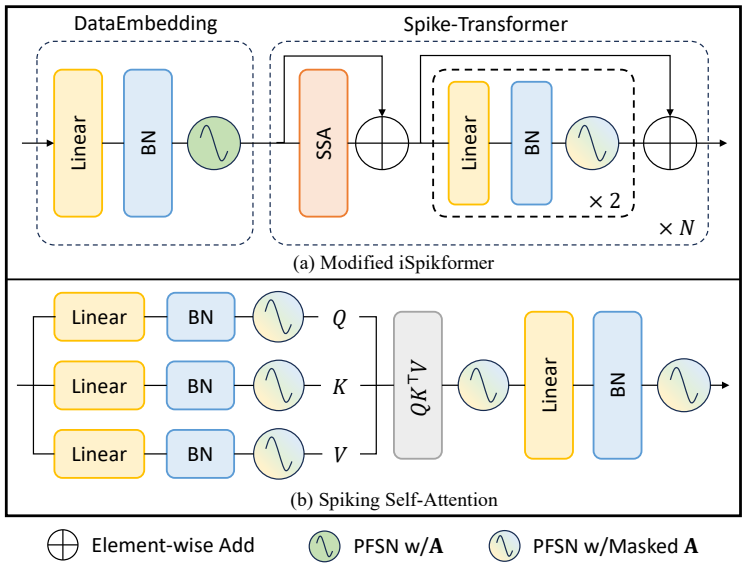


Figure 6: Modified iSpikformer architecture for multivariate time series forecasting. PFSN w/ $\mathbf{A}$ is used in the embedding layer to model cross-variate interactions, while PFSN w/ Masked $\mathbf{A}$ is used in the Spike-Transformer blocks to enable structured propagation along the representation dimension.

To evaluate PFSN on multivariate time series forecasting, we modify the iSpikformer architecture to better exploit the parallel modeling capability of the proposed neuron. As illustrated in Fig. 6, instead of using the original encoder to construct an additional spiking time dimension, we apply PFSN directly along the original dimensions of the forecasting model. In the embedding layer, PFSN is applied along the variable/channel dimension, enabling cross-variate interactions to be modeled in a parallel spiking form. We use PFSN w/ $\mathbf{A}$ in this stage, where the fully learnable propagation operator $\mathbf{A}$ is initialized with $\lambda = 0.85$.

For the Spike-Transformer blocks, we replace the original LIF neurons with PFSN modules. In the spiking self-attention (SSA) module, PFSN is inserted after the query, key, value, attention-output, and final projection transformations, and is applied along the representation dimension. To preserve structured dependency modeling, we use PFSN w/ Masked $\mathbf{A}$, where the learnable propagation operator is constrained to a lower-triangular form and initialized with $\lambda = 0.85$. The same design is adopted in the feed-forward block after each linear transformation. This modified architecture allows PFSN to serve not only as a spiking activation function, but also as a structured propagation operator over task-relevant representation axes.

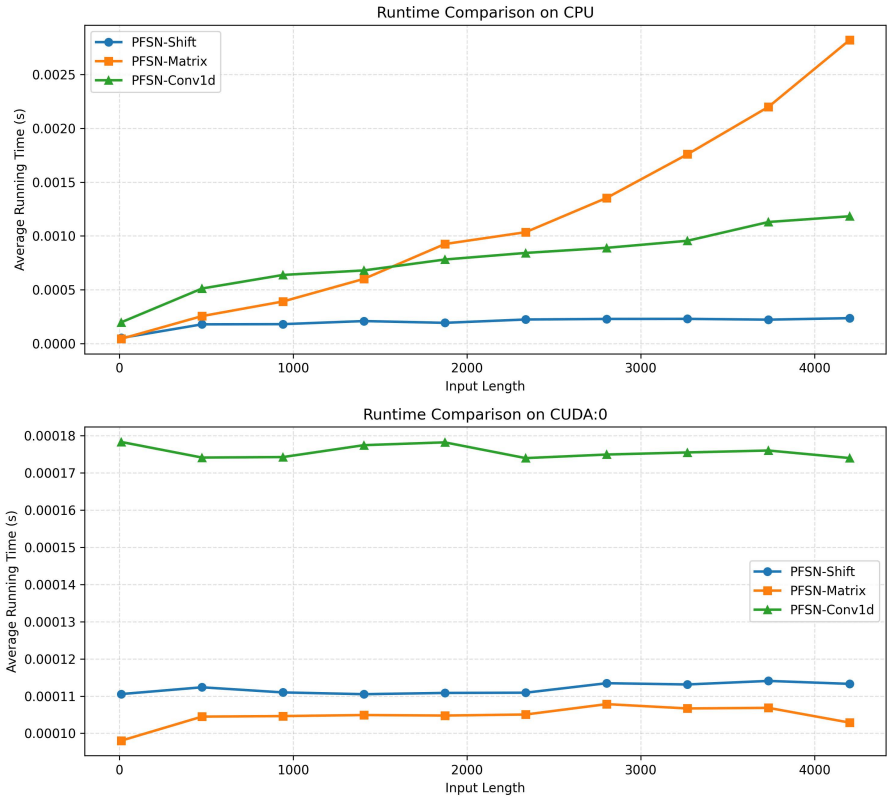


Figure 7: Runtime comparison of three propagation implementations, including direct shift, matrix multiplication, and 1D convolution, on CPU and GPU under different input lengths.

D Additional Experiments

D.1 Runtime Comparison of Propagation Implementations

To evaluate the implementation efficiency of the temporal propagation operator, we compare three realizations of the one-step delay operation: direct shift, matrix multiplication, and 1D convolution. The shift-based implementation directly shifts the reset membrane potential by one step, the matrix-based implementation uses a lower-diagonal propagation matrix with non-zero values only on the first sub-diagonal, and the convolution-based implementation realizes the same causal dependency with a kernel size of 2. We measure the average forward running time under different input lengths on both CPU and GPU with identical input dimensions and neuron parameters.

As shown in Fig. 7, the three implementations show different runtime characteristics. On CPU, direct shift achieves the lowest latency and grows slowly with sequence length, while matrix multiplication becomes more expensive for long sequences because the sparse propagation matrix is executed by a general matrix multiplication routine. The convolution-based version is faster than matrix multiplication on CPU for long sequences, but still suffers from padding and layout transformation overhead. On GPU, matrix multiplication is slightly faster under the tested setting due to optimized kernels, while direct shift remains comparable. In contrast, 1D convolution is consistently slower, mainly due to padding, layout transformation, and the low computational intensity of single-channel convolution. These results suggest that matrix multiplication benefits from sufficient GPU compute, whereas direct shift provides a lightweight implementation under more resource-constrained settings.

D.2 Effects of λ and α on Convergence

To study the effects of λ and α on the finite-iteration behavior of PFSN, we evaluate the spike error rate under different parameter combinations with a fixed iteration number $K = 3$. We focus on the hard-reset setting and, for each (λ, α) pair, generate random input sequences to compare the spike

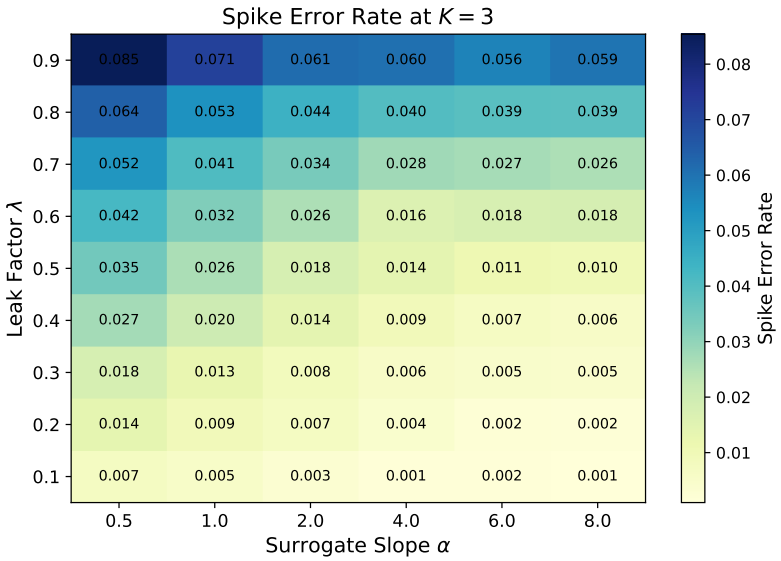


Figure 8: Effects of λ and α on the convergence of PFSN. The spike error rate is evaluated at a fixed iteration number $K = 3$ with respect to the sequential LIF output.

output of the shift-based PFSN with the reference output produced by the sequential hard-reset LIF neuron.

As shown in Fig. 8, the spike error rate is more sensitive to λ than to α. Smaller λ values lead to consistently lower error, indicating that weaker temporal dependence can be approximated more accurately with only a few iterations. As λ increases, the error becomes larger, which is consistent with the contraction condition in Eq. 16, where stronger temporal feedback weakens the convergence guarantee. The effect of α is slightly different: although a smaller α gives a tighter theoretical bound by reducing the Lipschitz constant of the surrogate reset mapping in Eq. 9, the heatmap measures spike error with respect to the original hard-threshold LIF dynamics. Therefore, a larger α can reduce the mismatch between the surrogate spike used during iteration and the final hard spike output. This reflects a trade-off between smoothness for contraction and accuracy for approximating hard LIF spikes under a limited iteration budget.

D.3 Convergence under Data-Driven Inputs

We evaluate non-learnable PFSN-H under realistic, MNIST-driven input currents through two complementary experiments. Both experiments use $T = 256$, $\lambda = 0.5$, and $\alpha = 5$.

Controlled convergence with fixed input currents. We pass MNIST test samples through a pretrained single-layer fully connected SNN (784×10) to obtain task-driven input currents. We then fix both the input currents and network parameters and vary only the number of fixed-point iterations K. Under these identical conditions, we compare PFSN-H with sequential hard-reset LIF using the normalized membrane-potential MSE and spike error rate.

Table 6: Controlled convergence under fixed MNIST-driven input currents.

K	Normalized voltage MSE	Spike error rate (%)
1	1.10362	2.2436
2	0.33030	2.0480
3	0.12928	1.6102
4	0.08513	1.7104
5	0.07182	1.6578

As shown in Tab. 6, the normalized membrane-potential MSE decreases consistently as K increases, indicating progressively closer agreement with sequential LIF dynamics under fixed, realistic input

currents. The spike error rate decreases overall from 2.24% to 1.66%, although it exhibits minor fluctuations at later iterations.

Task-level performance after LIF replacement. To assess the effect of the remaining approximation error on classification, we independently train a single-layer fully connected network (784×10) using non-learnable PFSN-H for each K. During inference, we evaluate the trained PFSN-H model and then replace its neurons with sequential hard-reset LIF without retraining. Both evaluations use the same MNIST test samples and learned synaptic weights.

Table 7: MNIST classification accuracy and approximation error before and after replacing PFSN-H with sequential hard-reset LIF.

K	PFSN-H accuracy (%)	LIF accuracy (%)	Normalized voltage MSE	Spike error rate (%)
1	92.40	92.28	0.27467	2.2625
2	92.24	92.47	0.08212	2.0497
3	92.35	92.38	0.03232	1.6204
4	92.36	92.44	0.02142	1.7201
5	92.37	92.43	0.01814	1.6679

Tab. 7 shows that classification accuracy remains comparable after LIF replacement. The absolute accuracy difference is at most 0.23% across the evaluated settings and is only 0.03% at $K = 3$. The normalized membrane-potential MSE also decreases across these settings, while the spike error rate shows an overall downward trend. Together, the controlled experiment isolates the effect of K on approximation quality, and the replacement experiment shows that the remaining discrepancy has little effect on MNIST classification accuracy. These results support the effectiveness of a small number of fixed-point iterations under realistic, data-driven inputs.

E Discussion and Limitation

PFSN reformulates LIF dynamics as a parallel fixed-point mapping, enabling membrane potentials across time steps to be updated simultaneously. Together with the learnable propagation operator, PFSN extends standard LIF dynamics and can be integrated into different SNN architectures. Its effectiveness across diverse tasks, including event-based recognition, sequential image classification, and time-series forecasting, demonstrates the flexibility of PFSN as a general parallel neuron module.

Nevertheless, several limitations remain. First, due to the difficulty of accessing neuromorphic chips and conducting systematic experiments on them, our evaluation mainly focuses on GPU-based training efficiency. Since PFSN preserves the original spiking neuron dynamics, we infer that it can in principle be deployed on existing neuromorphic hardware, while the efficiency of the shift-based implementation further suggests its potential for resource-constrained devices. However, practical deployment and hardware-level efficiency still require further validation. Second, although learnable propagation operators improve expressiveness, they introduce additional parameters and require structural constraints, such as lower-triangular forms, to preserve causality. Future work will explore hardware-aware implementations, customized CUDA acceleration, and a more complete theoretical analysis of learnable propagation dynamics.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introduction are consistent with the paper's main contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of the proposed method are discussed in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The assumptions for the theoretical results are explicitly stated, including the Lipschitz condition of the surrogate reset operator and the contraction condition required for convergence. Complete proofs are provided in Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all necessary details to reproduce the main experimental results in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide a publicly available code repository at https://github.com/Kiong/PFSN, including implementations and instructions for reproducing the experimental results.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: The paper reports the necessary experimental details in Sec. 5 and Appendix C, including datasets, model configurations, hyperparameters, optimizers, and training settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report results over multiple runs where applicable and include experiments across different settings to illustrate the stability of the results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The required details on computational resources are provided in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Both positive and negative societal impacts of the work are discussed in Appendix E.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not involve data or models that pose a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper properly credits all used assets and respects their licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new datasets or external assets are introduced beyond the provided code.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as part of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.