PICARDSNN: BREAKING THE SEQUENTIAL BARRIER OF SPIKING NEURAL NETWORKS WITH PICARD-PARALLEL-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Many important intelligence problems are inherently spatiotemporal: their solutions depend on how signals evolve and retain information over time, as in neuromorphic perception, event-based processing, and long-context sequence modeling. Such temporal dynamics require recurrent state updates, which purely feedforward networks cannot represent. Spiking Neural Networks (SNNs) are an attractive solution, combining nonlinear recurrent state dynamics, sparse event-driven communication, and energy-efficient inference. Their training, however, is hard: backpropagation through time (BPTT) unrolls the recurrence over T steps, resulting in inherently sequential computation over long temporal dependencies, limiting scalability and training efficiency on long sequences. We introduce **P**icard-**P**arallel-**T**raining (**PPT**), a parallel formulation of temporal training for SNNs, and use it to train **PicardSNN**. Instead of solving for the membrane potential, whose threshold-and-reset makes the recurrence a hard hybrid system, PPT solves for the pre-firing membrane potential, turning the dynamics into a linear recurrence with an explicit spike forcing term. PPT retains subtractive reset and isolates the spike-dependent nonlinearity as an explicit forcing term. Each Picard sweep freezes this term using the preceding iterate's spikes and solves the resulting affine recurrence through a logarithmic-depth parallel scan. A fused CUDA implementation combines forcing assembly and scan execution without materialising full-length intermediate forcing tensors. The reformulation is algebraically exact, while finite-sweep evaluation approximates the sequential trajectory unless a fixed point is reached. PPT supports sequences of up to 65,535 time steps and achieves up to **763×** forward and **348×** end-to-end speed-ups over sequential BPTT, with finite-sweep spike agreement and activation sparsity evaluated separately. Across four task families—neuromorphic audio, event-based vision, pixel-serial images, and the Long Range Arena—PicardSNN substantially outperforms Transformer baselines, remains competitive with specialised State-Space Models, and ranks among the strongest recurrent and spiking neural network methods.

1 INTRODUCTION

Many intelligence problems are spatiotemporal: success depends not on a single static input but on how information evolves and is integrated over time. Neuromorphic perception, event-based vision, and the long-horizon reasoning of language agents or embodied AI all require maintaining an internal state as evidence arrives, sometimes over thousands or tens of thousands of steps (Gu et al., 2021; Zhang et al., 2021). Purely feedforward networks cannot represent such persistent state, making recurrent computation essential. The central challenge is therefore to realize expressive recurrent dynamics without sacrificing computational efficiency on long-context sequences.

Long-context sequence modeling follows two dominant philosophies. Structured State-Space Models (SSMs) restrict the recurrence to be linear, enabling $\mathcal{O}(\log T)$-depth evaluation via parallel prefix scan and high training throughput. This linearity, however, is as much a limitation as an enabler: recent theory (Merrill et al., 2024; Muca Cirone et al., 2024) shows that linear SSMs cannot solve state-tracking problems that nonlinear recurrent networks handle naturally. The complementary philosophy preserves nonlinear recurrent dynamics—as in SNNs—for which direct temporal unrolling

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

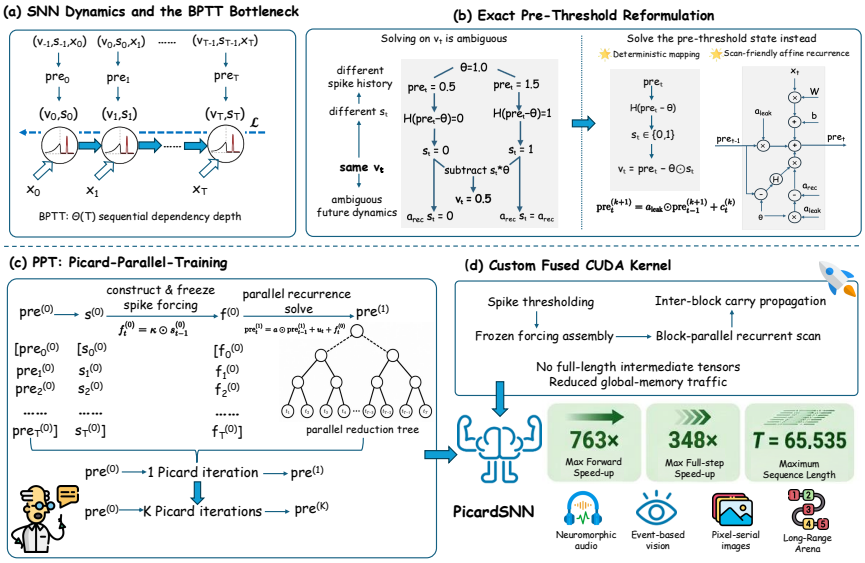

Figure 1: **Method overview.** (a) BPTT has $\Theta(T)$ sequential depth. (b) The reformulation removes reset-state ambiguity and exposes a scan-compatible recurrence. (c) PPT evaluates it through Picard iterations and parallel reductions. (d) A fused CUDA kernel eliminates full-length intermediates, enabling $T = 65{,}535$ with up to $763\times$ forward and $348\times$ full-step speed-up.

has $\Theta(T)$ sequential depth. This is not a universal limitation of SNN training: fixed-point parallel training and reset-aware boundary-compression methods already address temporal parallelism (Feng et al., 2025; Zhong et al., 2024; 2026). Our focus is the pre-threshold formulation and scan implementation of the subtractive-reset, diagonal self-feedback dynamics studied here.

SNNs (Maass, 1997; Yamazaki et al., 2022; Chen et al., 2026) implement such nonlinear recurrent dynamics using leaky integrate-and-fire neurons that communicate through binary spikes, enabling sparse event-driven computation with high energy efficiency and natural compatibility with neuromorphic hardware. However, unlike linear SSMs, SNNs cannot be parallelised by a straightforward prefix-scan because three properties of the LIF cell violate its assumptions:

- **Temporal dependence.** The membrane potential at step t depends on $t-1$, forming a causal chain of length T.
- **Hybrid state.** The neuron state combines a continuous membrane potential and a discrete spike, whereas scan-parallelism assumes a single continuous state with associative updates.
- **Non-smooth gate.** The Heaviside spike function is discontinuous, making the local linearisation required by fast nonlinear solvers ill-defined.

Our central observation is that these three barriers are not equally fundamental. Temporal dependence is intrinsic to recurrent models, but it need not imply sequential $\Theta(T)$ computation: causal recurrences can still be evaluated in logarithmic parallel depth. The latter two barriers, however, arise from the choice of state variable. By tracking the pre-threshold membrane potential pre_t instead of v_t, the subtractive-reset LIF dynamics becomes a linear recurrence with an explicit Heaviside forcing term (§3.2.1). This collapses the hybrid state into a single continuous variable and makes $\mathrm{pre} \mapsto H(\mathrm{pre}-\theta)$ single-valued, avoiding the oscillation caused by ambiguity in fixed-point iteration on v_t. Based on this reformulation, Picard iteration freezes the forcing term at the previous iterate, reducing each sweep to a linear bidiagonal solve that can be evaluated in $\mathcal{O}(\log T)$ parallel depth. The contraction analysis applies to a smooth threshold approximation under the stated condition. For the hard threshold, any fixed point exactly corresponds to the sequential rollout, while finite sweeps need not reach the fixed point but can still achieve high spike agreement in practice. We call this training procedure **Picard-Parallel-Training (PPT)**. Applying PPT to SNNs yields **PicardSNN**, illustrated in Figure 1.

Contributions.

1. **Pre-threshold Picard scanning.** We formulate subtractive-reset LIF dynamics in terms of the pre-threshold potential, separating the affine temporal recurrence from an explicit spike-dependent forcing term. Freezing this term at each Picard iteration enables an $\mathcal{O}(\log T)$-depth parallel scan while retaining the original reset mechanism. The reformulation is algebraically exact, while finite-sweep evaluation can achieve high spike agreement with the sequential trajectory without requiring full convergence, enabling faster computation.
2. **A fused CUDA implementation.** We implement a fused kernel that combines thresholding, forcing assembly, parallel time reduction, and cross-block carry propagation without materialising full-length intermediate tensors.
3. **PicardSNN and Empirical validation.** Building on PPT, we develop PicardSNN, a parallel-trained SNN that retains the original neuron dynamics while achieving strong performance across diverse sequence tasks.

2 Related Work

Long-sequence modeling. The core challenge of long-sequence modeling is retaining task-relevant information over long horizons within a finite state (Amos et al., 2023; Danieli et al., 2025). Attention (Vaswani et al., 2017) retains the full context but incurs quadratic memory and computation. SMT (Kumar & Isola, 2026) uses a Transformer teacher to pretrain RNN memory (Zhao et al., 2020; Oruh et al., 2022), preserving long-range information with linear-time inference (Grantner et al., 2026). SSMs take a different route: HiPPO (Gu et al., 2020), S4 (Gu et al., 2021), S4D (Gu et al., 2022), S5 (Smith et al., 2023), and Mamba (Gu & Dao, 2024; Dao & Gu, 2024) simplify and accelerate linear recurrences through parallel scan (Blelloch, 2002; Sengupta et al., 2010). Despite strong performance on Long Range Arena (Tay et al., 2020), their linear recurrence imposes a fundamental expressivity limitation (Merrill et al., 2024; Muca Cirone et al., 2024).

SNNs for sequence modeling. Recent advances in surrogate-gradient training (Neftci et al., 2019; Zenke & Ganguli, 2018) have enabled recurrent SNNs with conventional feedback connectivity to achieve strong performance on temporal classification (Bellec et al., 2018; Yin et al., 2021; 2023; Bittar & Garner, 2022; Hammouamri et al., 2024; Lv et al., 2024; Makovkin et al., 2025). However, their training still relies on sequential BPTT, making long-sequence optimisation increasingly expensive. Another direction instead combines spiking computation with linear state-space models: SpikeS4 (Du et al., 2024) and Binary S4D (Stan & Rhodes, 2023) introduce spiking activations into S4, while an S6-based stochastic spiking neuron (Bal & Sengupta, 2024) improves sparsity but suffers from noisy gradients and reduced accuracy. LMUFormer (Liu et al., 2024) combines Legendre Memory Units (Voelker et al., 2019) with convolution and spiking activations. SpikingSSMs (Shen et al., 2025) further integrate neuronal dynamics into an SSM backbone and use a lightweight surrogate network for parallel spike generation. SPikE-SSM (Zhong et al., 2024) and Dyn-SSM (Zhong et al., 2026) further incorporate history-dependent spiking dynamics into SSM-based architectures. Rather than directly using the standard LIF dynamics, they introduce modified reset-refractory neuron models and integrate them into an SSM backbone, with parallel processing enabled by a max–min boundary compression (PMBC) scheme. In contrast, our method retains the standard subtractive-reset LIF dynamics and derives a pre-threshold formulation that enables Picard iteration with an affine parallel scan, without an SSM backbone or modified neuron dynamics.

Parallelism for spiking cells. Efforts to parallelise spiking-cell computation include implementation optimisation, architectural simplification, and iterative solvers. (i) *Faster sequential rollout*: SLAYER (Shrestha & Orchard, 2018), snnTorch (Eshraghian et al., 2023), and SpikingJelly (Fang et al., 2023a) reduce overhead but retain $\Theta(T)$ sequential depth. (ii) *Architectural simplification*: Parallel Spiking Neurons (Fang et al., 2023b) remove recurrent reset, while Parallel Spiking Units (Li et al., 2024) decouple integration, firing, and reset via probabilistic reset, at the cost of increased firing activity and reduced sparsity. (iii) *Iterative solvers*: FPT (Feng et al., 2025) reformulates LIF dynamics as fixed-point iteration for parallel training without modifying the network architecture. Our approach follows this perspective but uses a different formulation: a pre-threshold state exposes the spike-dependent forcing term, which is frozen at each iteration to reduce the recurrence to an affine parallel scan, together with a fused CUDA implementation.

3 METHOD

3.1 PRELIMINARIES

3.1.1 THE SNN CELL

We use a reset-by-subtraction leaky integrate-and-fire (LIF) cell augmented with a learnable neuron-wise self-feedback pathway. Inspired by biological autapses, which are thought to support persistent neural activity and longer temporal integration, each neuron feeds its previous spike back only to itself when updating its membrane state. This provides a lightweight internal memory without introducing dense recurrent interactions. Let N be the number of neurons, T the sequence length, and $\mathbf{W} \in \mathbb{R}^{N \times D_{\text{in}}}$ the dense input projection. The vectors $\mathbf{a}_{\text{rec}}, \mathbf{a}_{\text{leak}}, \mathbf{b}, \boldsymbol{\theta} \in \mathbb{R}^N$ are learnable parameters. With membrane potential $\mathbf{v}_t$ and binary spike vector $\mathbf{s}_t$, the cell is

$$\begin{aligned} \mathbf{m}_t &= \mathbf{a}_{\text{leak}} \odot \mathbf{v}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{a}_{\text{rec}} \odot \mathbf{s}_{t-1} + \mathbf{b}, \\ \mathbf{s}_t &= H(\mathbf{m}_t - \boldsymbol{\theta}), \qquad \mathbf{v}_t = \mathbf{m}_t - \boldsymbol{\theta} \odot \mathbf{s}_t, \end{aligned} \tag{1}$$

where $\mathbf{v}_0 = \mathbf{s}_0 = \mathbf{0}$, H is the right-continuous Heaviside function. The self-feedback coefficients are stored as a vector; equivalently, they represent the diagonal matrix $\operatorname{diag}(\mathbf{a}_{\text{rec}})$, not a dense recurrent matrix. Thus the temporal self-feedback is neuron-wise, while the dense input projection performs channel mixing at the current time step.

The input matrix $\mathbf{W}$ is Xavier-initialised, $\mathbf{b}$ is initialised to zero, $\mathbf{a}_{\text{leak}}$ and $\boldsymbol{\theta}$ are initialised from task-specific values, and the vector $\mathbf{a}_{\text{rec}}$ is initialised near zero with small random perturbations unless a non-zero initial value is specified. During training, $\mathbf{a}_{\text{leak}}$ is constrained to $[0, 0.99]$ and $\boldsymbol{\theta}$ to $[0.05, 5.0]$. These constraints are post-update dynamical safeguards.

3.1.2 PICARD FIXED-POINT ITERATION AND PARALLEL REDUCTION

We recall two classical tools in task-agnostic form; their application to our work is given in §3.2.

Picard iteration. A fixed-point problem $z = \mathcal{F}(z)$ can be solved by Picard iteration $z^{(k+1)} = \mathcal{F}(z^{(k)})$ (Parker & Sochacki, 1996). If $\mathcal{F}$ is a contraction on a complete metric space—$\|\mathcal{F}(z) - \mathcal{F}(z')\| \le L\|z - z'\|$ with $L < 1$—the Banach fixed-point theorem guarantees a *unique* fixed point $z^\star$ and *geometric* convergence, $\|z^{(k)} - z^\star\| \le L^k \|z^{(0)} - z^\star\|$. Hence a target error is reached in iterations that depends on L but *not* on the dimension of z. Unlike Newton's method, Picard iteration uses only evaluations of $\mathcal{F}$ and needs no Jacobian, which is attractive when $\mathcal{F}$ is non-smooth.

Parallel reduction of a linear recurrence. A first-order linear affine recurrence $z_t = \lambda_t z_{t-1} + \beta_t$ can be written as the composition of per-step affine maps $\Phi_t(z) = \lambda_t z + \beta_t$. Because affine maps are closed under composition and composition is associative, the prefix compositions $\Phi_t \circ \cdots \circ \Phi_1$ can be evaluated by a balanced-tree parallel scan (Blelloch, 1990; Hillis & Steele Jr, 1986) in $\Theta(\log T)$ depth instead of the naive $\Theta(T)$. This mechanism gives linear SSMs their speed; the difficulty for spiking cells is that Eq. 1 is *not* affine, which the next section resolves.

3.2 PARALLEL PICARD TRAINING

3.2.1 PRE-THRESHOLD REFORMULATION.

Define the pre-threshold variable as the potential immediately before thresholding:

$$\text{pre}_t \;:=\; m_t \;=\; \mathbf{a}_{\text{leak}} \odot \mathbf{v}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{a}_{\text{rec}} \odot \mathbf{s}_{t-1} + \mathbf{b}. \tag{2}$$

The reset rule of Eq. 1 gives the identity

$$\mathbf{v}_{t-1} = \text{pre}_{t-1} - \boldsymbol{\theta} \odot \mathbf{s}_{t-1},$$

and, by definition,

$$\mathbf{s}_{t-1} = H(\text{pre}_{t-1} - \boldsymbol{\theta}).$$

Substituting both identities into Eq. 2 yields a recurrence closed in pre alone:

$$\begin{aligned} \text{pre}_t = \mathbf{a}_{\text{leak}} \odot \text{pre}_{t-1} + \big(\mathbf{W}\mathbf{x}_t + \mathbf{b}\big) \\ + \boldsymbol{\kappa} \odot H(\text{pre}_{t-1} - \boldsymbol{\theta}). \end{aligned} \tag{3}$$

The forcing coefficient is $\boldsymbol{\kappa} := \mathbf{a}_{\text{rec}} - \mathbf{a}_{\text{leak}} \odot \boldsymbol{\theta}$.

Spikes and membrane potentials are read back as

$$\mathbf{s}_t = H(\text{pre}_t - \boldsymbol{\theta}), \qquad \mathbf{v}_t = \text{pre}_t - \boldsymbol{\theta} \odot \mathbf{s}_t.$$

Three properties make Eq. 3 amenable to parallelization.

(P1) Exact reformulation. The substitution introduces no approximation: in exact arithmetic, Eq. 3 reproduces the same spike train and membrane-potential trajectory as Eq. 1. Any approximation arises only from the finite-iteration solver, not from the reformulation itself.

(P2) A single continuous state. The spike appears only through an explicit forcing term,

$$c_t := \mathbf{W}\mathbf{x}_t + \mathbf{b} + \boldsymbol{\kappa} \odot H(\text{pre}_{t-1} - \boldsymbol{\theta}), \tag{4}$$

so Eq. 3 is an affine recurrence $\text{pre}_t = \mathbf{a}_{\text{leak}} \odot \text{pre}_{t-1} + c_t$. For a fixed forcing sequence $\{c_t\}_{t=1}^{T}$, the recurrence is linear in pre; the only nonlinearity lies in constructing c_t.

(P3) A unique state-to-spike mapping. Unlike a sequential rollout, which computes the spike explicitly at every time step, a parallel fixed-point solver must recover the spike decision directly from the current state iterate. This is precisely why we solve for pre rather than v. The post-reset membrane potential is not injective: $v_t = m_t - \theta \odot H(m_t - \theta)$ maps both firing and non-firing histories to the same value, so v_t alone does not uniquely determine s_t. Near a spike-flip event, this ambiguity causes fixed-point iterations on v to oscillate between the two branches, preventing stable convergence. By contrast, pre_t records the membrane potential before reset, so $s_t = H(\text{pre}_t - \theta)$ is uniquely determined for every $\text{pre}_t \in \mathbb{R}$. Consequently, each pre-threshold trajectory implies exactly one spike train, enabling the Picard iteration to converge to a unique fixed point.

3.2.2 The Picard Iteration Step by Step

Why not Newton. One might instead solve the T-equation system of Eq. 1 with Newton iteration, as in smooth recurrent cells (Danieli et al., 2025). However, the Heaviside threshold makes the Jacobian ill-defined: setting $H' \equiv 0$ reduces Newton to fixed-point iteration, while surrogate substitutes introduce bias into the forward solution. Moreover, the spike-decision ambiguity in *(P3)* causes residual discontinuities near spike transitions. We therefore apply Picard iteration to the reformulated Eq. 3, which requires no Jacobian and preserves the exact spiking dynamics (Paniconi & Putti, 1994).

The iteration. Following §3.1.2, we treat the pre-threshold recurrence Eq. 3 as a fixed-point problem $\text{pre} = \mathcal{F}(\text{pre})$ and solve it by Picard iteration. Initialise $\text{pre}_t^{(0)} = \mathbf{W}\mathbf{x}_t + \mathbf{b}$ (the trajectory with no spikes) and for $k = 0, \ldots, K-1$,

$$c_t^{(k)} = (\mathbf{W}\mathbf{x}_t + \mathbf{b}) + \boldsymbol{\kappa} \odot H(\text{pre}_{t-1}^{(k)} - \boldsymbol{\theta}), \tag{5}$$

$$\text{pre}_t^{(k+1)} = \mathbf{a}_{\text{leak}} \odot \text{pre}_{t-1}^{(k+1)} + c_t^{(k)}, \quad t = 1, \ldots, T. \tag{6}$$

The key move is that the forcing $c_t^{(k)}$ is frozen at the previous iterate: it depends on $\text{pre}^{(k)}$, not on $\text{pre}^{(k+1)}$. Consequently Eq. 6 is, in the unknown $\text{pre}^{(k+1)}$, a plain first-order affine recurrence with the known, per-step coefficients $\lambda_t = a_{\text{leak}}$ and $\beta_t = c_t^{(k)}$ of §3.1.2. It is therefore solved in $\Theta(\log T)$ depth by a single Hillis–Steele parallel scan—no Jacobian assembly and no surrogate inside the forward pass.

One update in slow motion. Starting from a whole candidate trajectory $\text{pre}^{(k)} = (\text{pre}_1^{(k)}, \ldots, \text{pre}_T^{(k)})$: (i) threshold each step to read off its spikes, $s_t^{(k)} = H(\text{pre}_t^{(k)} - \theta)$; (ii) assemble and freeze the forcing, $c_t^{(k)} = (Wx_t + b) + \kappa \odot s_{t-1}^{(k)}$, which is now a fixed input sequence; (iii) scan the affine recurrence $\text{pre}_t^{(k+1)} = a_{\text{leak}} \odot \text{pre}_{t-1}^{(k+1)} + c_t^{(k)}$ in parallel to obtain the next whole trajectory. The data flow of one update is

$$\text{pre}^{(k)} \xrightarrow{\text{threshold}} s^{(k)} \xrightarrow{\text{freeze forcing}} c^{(k)} \xrightarrow{\text{parallel scan}} \text{pre}^{(k+1)}.$$

A worked scalar example. We now compare the direct sequential rollout with the Picard sweeps for one neuron. Let $a_{\text{leak}} = 0.9$, $\theta = 1$, $a_{\text{rec}} = 0.5$, let the external input drive be $u_t = Wx_t + b = 0.6$ for $t \in \{1, 2, 3\}$, and initialise $\text{pre}_0 = v_0 = s_0 = 0$.

Sequential rollout. The first step has no previous membrane potential or spike:

$$\text{pre}_1 = 0.9 \times 0 + 0.6 = 0.600, \qquad s_1 = H(0.600 - 1) = 0,$$

so $v_1 = \text{pre}_1 - 1 \times s_1 = 0.600$. At the second step,

$$\begin{aligned}\text{pre}_2 &= 0.9 \times v_1 + 0.6 + 0.5 \times s_1 \\ &= 0.9 \times 0.600 + 0.6 + 0.5 \times 0 = 1.140,\end{aligned}$$

so $s_2 = H(1.140 - 1) = 1$ and $v_2 = 1.140 - 1 = 0.140$. At the third step,

$$\begin{aligned}\text{pre}_3 &= 0.9 \times v_2 + 0.6 + 0.5 \times s_2 \\ &= 0.9 \times 0.140 + 0.6 + 0.5 \times 1 \\ &= 0.126 + 0.600 + 0.500 = 1.226.\end{aligned}$$

Hence $s_3 = 1$ and $v_3 = 1.226 - 1 = 0.226$. The sequential trajectory is therefore

t	1	2	3
pre_t	0.600	1.140	1.226
s_t	0	1	1
v_t	0.600	0.140	0.226

Picard initialisation. The pre-threshold reformulation uses

$$\kappa = a_{\text{rec}} - a_{\text{leak}}\theta = 0.5 - 0.9 \times 1 = -0.4.$$

The initial candidate ignores previous spikes and uses only the external drive:

$$\text{pre}^{(0)} = (0.600, 0.600, 0.600).$$

Since no values reaches $\theta = 1$, the spike sequence is

$$s^{(0)} = (0, 0, 0).$$

The forcing for the first sweep is consequently

$$c_t^{(0)} = u_t + \kappa s_{t-1}^{(0)}, \qquad c^{(0)} = (0.600, 0.600, 0.600),$$

where the boundary spike $s_0^{(0)}$ is zero.

Picard sweep 1. The forcing is frozen at $c^{(0)}$. The parallel scan solves the affine recurrence, which we write sequentially here only to expose the arithmetic:

$$\begin{aligned}\text{pre}_1^{(1)} &= 0.9 \times 0 + 0.600 = 0.600, \\ \text{pre}_2^{(1)} &= 0.9 \times 0.600 + 0.600 = 0.540 + 0.600 = 1.140, \\ \text{pre}_3^{(1)} &= 0.9 \times 1.140 + 0.600 = 1.026 + 0.600 = 1.626.\end{aligned}$$

Thus $\text{pre}^{(1)} = (0.600, 1.140, 1.626)$, $s^{(1)} = (0, 1, 1)$.

The third pre-value is temporarily high because the second-step spike has not yet been included in the frozen forcing.

Picard sweep 2. Only the previous time step can contribute to the current forcing, so $s^{(1)}$ is shifted by one position:

$$(s_0^{(1)}, s_1^{(1)}, s_2^{(1)}) = (0, 0, 1).$$

Therefore

$$\begin{aligned}c_1^{(1)} &= 0.600 + (-0.4) \times 0 = 0.600, \\ c_2^{(1)} &= 0.600 + (-0.4) \times 0 = 0.600, \\ c_3^{(1)} &= 0.600 + (-0.4) \times 1 = 0.200.\end{aligned}$$

The next parallel scan is

$$\begin{aligned}\mathrm{pre}_1^{(2)} &= 0.9 \times 0 + 0.600 = 0.600,\\ \mathrm{pre}_2^{(2)} &= 0.9 \times 0.600 + 0.600 = 1.140,\\ \mathrm{pre}_3^{(2)} &= 0.9 \times 1.140 + 0.200 = 1.026 + 0.200 = 1.226.\end{aligned}$$

Consequently, $\mathrm{pre}^{(2)}$ and $s^{(2)}$ exactly matches the sequential rollout above.

Picard sweep 3 and fixed point. Since $s^{(2)} = s^{(1)}$, the forcing remains unchanged, and the third sweep returns the same trajectory $(0.600, 1.140, 1.226)$. The fixed point is therefore reached: Picard sweeps recover the sequential rollout while replacing step-by-step execution with affine time scans. For a sequence of length T, the total parallel depth is determined by the number of sweeps K rather than the rollout length.

How many iterations. Each Picard sweep requires one $\Theta(\log T)$ affine scan, giving total depth $\Theta(K \log T)$. The next theorem shows that K is a constant independent of T.

Theorem 1 (Contraction of Picard-pre)**.** *Let H_σ approximate H with slope bounded by $\beta < \infty$, and set $\alpha := \|a_{leak}\|_\infty$ and $\rho := \|\kappa\|_\infty \beta$. If $\alpha + \rho < 1$, the Picard-pre operator defined by Eqs. 5–equation 6 is a strict contraction on $\ell^\infty(\{1, \ldots, T\})$ with rate $\alpha + \rho$, and its unique fixed point is the exact sequential rollout of Eq. 3.*

The proof in Appendix applies the mean-value theorem to H_σ and telescopes Eq. 6. It shows geometric convergence with error $\mathcal{O}((\alpha + \rho)^K)$, making the iteration count independent of sequence length. Proofs of smooth contraction and hard-threshold fixed-point consistency are also provided in Appendix. By *(P1)*, the fixed point exactly recovers the sequential SNN rollout rather than approximating it. Empirically, $K = 4$ matches 85–90% of spike positions for sequences up to $T = 65{,}535$ (§4.2), with remaining mismatches caused by floating-point-sensitive metastable spikes and having negligible impact on end-to-end accuracy (§4.3).

3.2.3 Temporal Parallel Complexity

Direct evaluation of Eq. 1 incurs a $\Theta(T)$ temporal dependency depth. PPT freezes the nonlinear forcing in each Picard sweep and solves the resulting affine recurrence with a balanced parallel scan, reducing the depth of each sweep to $\Theta(\log T)$. Therefore, K sweeps require $\Theta(K \log T)$ depth, where K is independent of T under the contraction condition of Theorem 1 (fixed to four in our experiments). The same principle applies to both forward and backward passes: PPT replaces sequential rollout and time-direction gradient recursion with parallel reductions, reproducing the original dynamics and BPTT gradients while reducing sequential depth.

3.3 Fused CUDA Kernel

A naive Picard sweep launches multiple GPU operators for thresholding, forcing construction, time reduction, and state propagation, while materialising (B, N, T) intermediate tensors. For long sequences, the resulting HBM traffic becomes the dominant cost. We therefore implement a fused CUDA kernel (Muller & Hoffmann, 2021) that combines the entire sweep into a single operator (Algorithm A.2). Each thread block handles one (b, n) pair and processes time in parallel chunks, keeping intermediate values in registers or shared memory. Cross-block propagation and affine recurrence reduction are handled by a hierarchical parallel scan, substantially reducing memory movement and kernel-launch overhead.

4 Experiments

4.1 Setup

PicardSNN comprises an input encoder, L stacked SNN layers based on Eq. 1 with linear projections and layer normalisation, and a task-specific head. Unless stated otherwise, experiments use width 512, $L = 3$ ($\approx$1.3M parameters), AdamW with cosine warm-up scheduling and EMA (0.999). We use fast-sigmoid surrogate gradients except for the longest-range tasks, where Gaussian surrogates are adopted. All Picard results use $K = 4$ iterations. Baselines are shown in Appendix A.1.

Table 1: Forward and full-step (forward + backward) speed-up. "Spike Acc" is the fraction of (b, n, t) spike positions matching the sequential rollout under identical parameters.

T	SEQUENTIAL	**Ours**			Spike
	fwd (ms)	fwd (ms)	fwd spd	total spd	Acc
128	20.84	0.64	32×	12×	87.7%
256	41.85	0.65	65×	24×	86.6%
512	85.22	0.64	132×	45×	85.7%
1 024	169.34	0.65	262×	96×	86.5%
2 048	341.52	0.69	493×	190×	87.2%
4 096	707.45	1.11	635×	274×	87.8%
8 192	1 379.45	2.09	659×	294×	88.5%
16 384	2 871.04	3.97	723×	325×	88.8%
32 768	5 819.40	7.63	763×	348×	88.0%
65 535	11 615.01	15.47	751×	342×	89.3%

4.2 Training Speed-Up and Spike Fidelity

We benchmark on a single data-center GPU (CUDA 12.1, fp32) with $B = 8$, $N = 128$, $D_{\text{in}} = 64$, $K = 4$, and $T \in \{128, \ldots, 65535\}$; timings are averaged over 5 runs after 3 warm-ups. We evaluate sequences up to $T = 65,535$, the maximum length supported by our current fused kernel due to implementation-level indexing constraints, rather than an algorithmic limitation.

As shown in Table 1, the forward speed-up increases with sequence length and saturates at long T due to HBM bandwidth limits. The fused kernel gains primarily from operator fusion and reduced intermediate memory traffic. With $K = 4$, spike recovery matches the sequential rollout at 85–90%, supporting the length-independent convergence predicted by Theorem 1. The remaining mismatches are metastable spikes sensitive to floating-point rounding and do not affect end-to-end accuracy (§4.3). We fix $K = 4$ throughout and analyse the speed–fidelity trade-off in §4.4.

4.3 End-to-End Accuracy Across Task Families

We group benchmarks by modality and compare with standard baselines for each family.

On SHD and SSC (Table 2 Left), PicardSNN achieves 92.27% and 82.77%, respectively, obtaining the best SSC accuracy while remaining competitive on SHD without task-specific mechanisms such as neural adaptation, structured memory, or learnable delays. On N-MNIST, sMNIST, and psMNIST (Table 2 Right), it consistently ranks among the best methods, achieving 99.72%, 99.17%, and 98.45%. On LRA Tasks (Table 3), PicardSNN achieves 84.39% average accuracy, outperforming Transformer by 30.70% and achieving performance comparable to specialised SSMs. It also achieves the best average performance among spiking models. Under matched depth and parameter budgets, PicardSNN consistently outperforms LSTM and GRU trained with ParaRNN or SMT across all six tasks, with a gain of 35.29%–44.08%, demonstrating the effectiveness of PPT for training nonlinear recurrent models.

4.4 Fidelity–Efficiency Trade-off in PPT

We first evaluate spike fidelity, since acceleration is meaningful only if the parallel solver preserves the sequential trajectory. Averaged over sequence lengths, spike accuracy is 90.8%, 87.6%, 87.3%, 89.8%, and 92.6% for $K \in \{2, 4, 8, 16, 32\}$, respectively. All settings remain within the same 85–95% range, indicating that additional Picard iterations provide no practical fidelity gain. The remaining mismatches mainly correspond to metastable threshold events sensitive to floating-point perturbations and do not affect downstream accuracy (§4.3).

The figure in Appendix A.3 shows that $K = 4$ achieves the best speed, reaching up to $763\times$ acceleration. Larger K incurs additional $\Theta(\log T)$ scans and therefore lower throughput. The slightly lower speed-up of $K = 2$ is due to fixed system overheads dominating when fewer sweeps are used, whereas $K \geq 4$ reaches the steady-state efficiency of the fused kernel. For fixed K, speed-up increases with sequence length as parallelism is better amortised, then saturates at very long sequences due to HBM bandwidth limits and cross-block synchronization overhead.

Table 2: Accuracy (%) on SHD / SSC and N-MNIST / sMNIST / psMNIST.

Model	SNN	SHD	SSC
LSTM	No	88.80	73.10
GRU	No	90.40	79.05
liBRU	No	89.61	78.70
LIF	Yes	49.80	41.50
RSNN	Yes	71.40	50.90
SpikGRU	Yes	87.80	77.00
Adaptive RSNN	Yes	84.40	74.20
RadLIF	Yes	94.62	77.40
TC-LIF	Yes	88.91	61.09
d-cAdLIF	Yes	94.85	80.23
SE-adLIF	Yes	**95.61**	80.44
Spikformer	Yes	90.10	80.18
SDT	Yes	89.61	79.82
Spiking S4	Yes	94.60	81.70
DCLS-Delays	Yes	95.07	80.16
PicardSNN (PPT)	Yes	92.27	**82.77**

Model	SNN	N-MNIST	sMNIST	psMNIST
LSTM	No	99.18	98.90	93.60
GRU	No	99.24	99.10	95.07
LMU	No	99.31	98.49	97.15
coRNN	No	99.08	99.00	97.80
Transformer	No	98.60	98.80	97.90
LMUFormer	No	99.46	98.95	98.15
S4	No	99.27	**99.63**	**98.70**
Adaptive SRNN	Yes	99.42	98.37	94.30
LIF	Yes	99.64	72.06	80.36
PLIF	Yes	**99.72**	87.92	86.76
GLIF	Yes	99.48	95.27	90.47
ALIF	Yes	97.71	98.70	94.30
FPTT-LTC-SNN	Yes	99.62	97.37	94.77
SpikingLMUFormer	Yes	99.51	98.16	97.92
Binary S4D	Yes	99.12	99.40	98.10
S6-based SNN	Yes	98.86	98.94	98.40
SpikingSSM-VT	Yes	99.35	99.05	98.40
PicardSNN (PPT)	Yes	**99.72**	99.17	98.45

Table 3: Accuracy (%) on the LRA tasks. * indicates failed training with chance-level performance.

Model	SNN	LISTOPS	TEXT	RETRIEVAL	IMAGE	PATHFINDER	Path-X	AVG
Transformer	No	36.37	64.27	57.46	42.44	71.40	50.20*	53.69
LMUFormer	No	34.43	68.27	78.65	54.16	69.90	50.20*	59.27
S4D-Lin	No	60.52	86.97	90.96	87.93	93.96	92.80	85.52
S4	No	58.35	86.82	90.90	87.26	94.20	96.35	85.65
S5	No	**62.15**	**89.31**	**91.40**	**88.65**	**95.33**	**98.58**	**87.57**
LSTM (BPTT)	No	18.25	51.36*	51.06*	56.62	50.30*	50.15*	46.29
GRU (BPTT)	No	38.30	53.24*	52.85*	64.85	50.46*	50.30*	51.67
LSTM (w/ ParaRNN)	No	18.55	50.26*	51.63*	20.82	50.30*	50.30*	40.31
GRU (w/ ParaRNN)	No	18.50	50.26*	52.12*	24.94	50.30*	50.30*	41.07
LSTM (w/ SMT)	No	18.30	50.44*	77.31	40.07	50.30 *	50.30*	47.79
GRU (w/ SMT)	No	18.45	50.34*	77.43	47.79	50.30*	50.30*	49.10
Spiking LMUFormer	Yes	37.30	65.80	79.76	55.65	72.68	50.30*	60.25
Binary S4D	Yes	54.80	82.50	85.03	82.00	82.60	61.20	74.69
S6-based SNN	Yes	55.70	77.62	88.48	80.10	83.41	50.10*	72.57
SpikingSSM-VF	Yes	57.82	83.05	88.05	85.95	**92.48**	92.98	83.39
SpikingSSM-VT	Yes	**60.10**	80.84	88.92	**87.11**	92.21	93.52	83.78
PicardSNN (PPT)	Yes	55.55	**85.58**	**90.32**	84.12	91.51	**99.28**	**84.39**

5 Conclusion

Spiking recurrence has long been treated as inherently sequential, and this paper shows that the fundamental obstacle lies not in the spikes themselves but in the state representation used for computation. By reformulating the dynamics through the pre-threshold state, an unmodified reset-by-subtraction SNN can be expressed as a tractable recurrence with an explicit spike-driven term. This reformulation enables parallel sequence evaluation while preserving the non-linear dynamics responsible for the expressive and energy-efficient properties of spiking networks. Our results demonstrate that non-linear recurrence and efficient long-sequence training are not fundamentally incompatible, narrowing the gap between spiking models and modern state-space sequence models. This work opens a path toward scalable spiking systems capable of combining the temporal efficiency of recurrent computation with the parallel training efficiency required for long-horizon intelligence.

Limitations and Future work. Beyond the LIF dynamics, we are actively extending PPT to richer spiking neuron families, including adaptive LIF variants. Another promising direction is adaptive Picard iteration, where the number of sweeps is adjusted according to trajectory convergence.

AI USE STATEMENT

We used generative AI tools solely to check for grammatical errors and polish the academic expression of portions of the manuscript. No generative AI tools were used to develop the methods, formulate theoretical claims, design experiments, implement the proposed method, analyze or interpret experimental results, or generate research findings. All AI-assisted text was reviewed and verified by the authors. We take full responsibility for the final content of this work, including all claims, methods, results, and artifacts.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or sensitive personal data. The proposed method is intended for research on efficient training of spiking neural networks and does not introduce known risks to individuals or groups. We have followed standard research practices for reporting methods and experimental results, and have made reasonable efforts to ensure that the reported results are accurate, reproducible, and transparently documented.

REFERENCES

Ido Amos, Jonathan Berant, and Ankit Gupta. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. *arXiv preprint arXiv:2310.02980*, 2023.

Malyaban Bal and Abhronil Sengupta. Rethinking spiking neural networks as state space models. *arXiv preprint arXiv:2406.02923*, 2(3):4, 2024.

Maximilian Baronig, Romain Ferrand, Silvester Sabathiel, and Robert Legenstein. Advancing spatio-temporal processing in spiking neural networks through adaptation. *arXiv preprint arXiv:2408.07517*, 2024.

Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.

Alexandre Bittar and Philip N Garner. A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, 16:865897, 2022.

Guy E Blelloch. Prefix sums and their applications. 1990.

Guy E Blelloch. Scans as primitive parallel operations. *IEEE Transactions on computers*, 38(11): 1526–1538, 2002.

Guozhang Chen, Junrong Qi, Xingyu Liu, Qixuan Fan, Zhuoheng Gao, Songcheng Xu, Xuliang Wang, Yujie Wu, Dongping Yang, Douglas Zhou, et al. Redefining spiking neural networks through the lens of dynamical superspace. *Cognitive Neurodynamics*, 20(1):101, 2026.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

Federico Danieli, Pau Rodriguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. Pararnn: Unlocking parallel training of nonlinear rnns for large language models. *arXiv preprint arXiv:2510.21450*, 2025.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Lucas Deckers, Laurens Van Damme, Werner Van Leekwijck, Ing Jyh Tsang, and Steven Latré. Co-learning synaptic delays, weights and adaptation in spiking neural networks. *Frontiers in Neuroscience*, 18:1360300, 2024.

Yu Du, Xu Liu, and Yansong Chua. Spiking structured state space model for monaural speech enhancement. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 766–770. IEEE, 2024.

Jason K Eshraghian, Max Ward, Emre O Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023.

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*, pp. 2661–2671, 2021.

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, 2023a.

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, 2023b.

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. *arXiv preprint arXiv:2506.12087*, 2025.

Tobias Grantner, Emanuel Sallinger, and Martin Flechl. Linear-time and constant-memory text embeddings based on recurrent language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 41459–41481, 2026.

Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in neural information processing systems*, 35:35971–35983, 2022.

Ilyass Hammouamri, Ismail Khalfaoui Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *International conference on learning representations*, volume 2024, pp. 17890–17903, 2024.

W Daniel Hillis and Guy L Steele Jr. Data parallel algorithms. *Communications of the ACM*, 29 (12):1170–1183, 1986.

Akarsh Kumar and Phillip Isola. Pretraining recurrent networks without recurrence. *arXiv preprint arXiv:2606.06479*, 2026.

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)*, pp. 1–8. IEEE, 2024.

Zeyu Liu, Gourav Datta, Anni Li, and Peter Beerel. Lmuformer: Low complexity yet powerful spiking model with legendre memory units. In *International conference on learning representations*, volume 2024, pp. 12108–12122, 2024.

Changze Lv, Dongqi Han, Yansen Wang, Xiaoqing Zheng, Xuanjing Huang, and Dongsheng Li. Advancing spiking neural networks for sequential modeling with central pattern generators. *Advances in Neural Information Processing Systems*, 37:26915–26940, 2024.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

S Yu Makovkin, S Yu Gordleeva, and IA Kastalskiy. Toward a biologically plausible snn-based associative memory with context-dependent hebbian connectivity. *International journal of neural systems*, 35(11):2550027, 2025.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

Nicola Muca Cirone, Antonio Orvieto, Benjamin Walker, Cristopher Salvi, and Terry Lyons. Theoretical foundations of deep selective state-space models. *Advances in Neural Information Processing Systems*, 37:127226–127272, 2024.

Stefan K Muller and Jan Hoffmann. Modeling and analyzing evaluation cost of cuda kernels. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–31, 2021.

Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

Jane Oruh, Serestina Viriri, and Adekanmi Adegun. Long short-term memory recurrent neural network for automatic speech recognition. *IEEE Access*, 10:30069–30079, 2022.

Claudio Paniconi and Mario Putti. A comparison of picard and newton iteration in the numerical solution of multidimensional variably saturated flow problems. *Water Resources Research*, 30 (12):3357–3374, 1994.

G Edgar Parker and James S Sochacki. Implementing the picard iteration. *Neural, Parallel & Scientific Computations*, 4(1):97–112, 1996.

T Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. *arXiv preprint arXiv:2010.00951*, 2020.

Shubhabrata Sengupta, Mark J Harris, Michael Garland, and John D Owens. Efficient parallel scan algorithms for manycore gpus., 2010.

Shuaijie Shen, Chao Wang, Renzhuo Huang, Yan Zhong, Qinghai Guo, Zhichao Lu, Jianguo Zhang, and Luziwei Leng. Spikingssms: Learning long sequences with sparse and parallel spiking state space models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20380–20388, 2025.

Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. International Conference on Learning Representations (ICLR), 2023.

Matei Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *arXiv preprint arXiv:2401.00955*, 2023.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

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.

Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

Jiaqi Wang, Liutao Yu, Liwei Huang, Chenlin Zhou, Han Zhang, Zhenxi Song, Honghai Liu, Min Zhang, Zhengyu Ma, and Zhiguo Zhang. Efficient speech command recognition leveraging spiking neural networks and progressive time-scaled curriculum distillation. *Neural Networks*, pp. 108253, 2025.

Jiaqi Wang, Liutao Yu, Xiongri Shen, Sihang Guo, Chenlin Zhou, Leilei Zhao, Yi Zhong, Zhiguo Zhang, and Zhengyu Ma. Spikcommander: A high-performance spiking transformer with multi-view learning for efficient speech command recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 2119–2127, 2026.

Kashu Yamazaki, Viet-Khao Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *Brain sciences*, 12(7):863, 2022.

Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer. *Advances in neural information processing systems*, 36:64043–64058, 2023.

Xingting Yao, Fanrong Li, Zitao Mo, and Jian Cheng. Glif: A unified gated leaky integrate-and-fire neuron for spiking neural networks. *Advances in Neural Information Processing Systems*, 35: 32160–32171, 2022.

Bojian Yin, Federico Corradi, and Sander M Bohté. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nature Machine Intelligence*, 3(10): 905–913, 2021.

Bojian Yin, Federico Corradi, and Sander M Bohté. Accurate online training of dynamical spiking neural networks through forward propagation through time. *Nature Machine Intelligence*, 5(5): 518–527, 2023.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

Shimin Zhang, Qu Yang, Chenxiang Ma, Jibin Wu, Haizhou Li, and Kay Chen Tan. Tc-lif: A two-compartment spiking neuron model for long-term sequential modelling. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 16838–16847, 2024.

Zhu Zhang, Chang Zhou, Jianxin Ma, Zhijie Lin, Jingren Zhou, Hongxia Yang, and Zhou Zhao. Learning to rehearse in long sequence memorization. In *International Conference on Machine Learning*, pp. 12663–12673. PMLR, 2021.

Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do rnn and lstm have long memory? In *International Conference on Machine Learning*, pp. 11365–11375. PMLR, 2020.

Yan Zhong, Ruoyu Zhao, Chao Wang, Qinghai Guo, Jianguo Zhang, Zhichao Lu, and Luziwei Leng. Spike-ssm: A sparse, precise, and efficient spiking state space model for long sequences learning. *arXiv preprint arXiv:2410.17268*, 2024.

Yan Zhong, Ruoyu Zhao, Chao Wang, Jiaqi He, Qinghai Guo, Jianguo Zhang, Zhichao Lu, and Luziwei Leng. Dyn-ssm: Towards the efficient long sequence learning via bio-interpretable dynamics in spiking state space models. *IEEE Transactions on Cognitive and Developmental Systems*, 2026.

Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. *arXiv preprint arXiv:2209.15425*, 2022.

A Appendix

A.1 Baselines and comparison protocol.

All results are averaged over three independent random seeds. We compare three baseline families.**(i)** non-spiking sequence models, including RNNs and Transformers (Graves, 2012; Cho et al., 2014; Voelker et al., 2019; Rusch & Mishra, 2020; Vaswani et al., 2017; Liu et al., 2024; Gu et al., 2021; 2022; Smith et al., 2023). **(ii)** conventional and specialised SNNs (Yin et al., 2021; Bittar & Garner, 2022; Zhang et al., 2024; Deckers et al., 2024; Baronig et al., 2024; Fang et al., 2021; Yao et al., 2022; Bellec et al., 2018; Yin et al., 2023; Zhou et al., 2022; Yao et al., 2023; Hammouamri et al., 2024), while Spike-driven Transformer is abbreviated as SDT. **(iii)** parallel or SSM-based SNNs (Liu et al., 2024; Stan & Rhodes, 2023; Bal & Sengupta, 2024; Shen et al., 2025). RNN/SNN comparisons use matched parameter, while specialised SSMs use their strongest reported settings. We exclude speech-specific systems whose specialised modules do not isolate general sequence modeling (Wang et al., 2025; 2026). On LRA, we further compare training methods with ParaRNN and SMT (Danieli et al., 2025; Kumar & Isola, 2026) by applying them to parameter-matched LSTMs and GRUs, isolating the effect of the training algorithm.

A.2 Algorithm used in FUSED CUDA KERNEL

Algorithm 1 One Picard step for one (b, n) thread block.

Input: $\mathrm{pre}_{\mathrm{in}}, \mathbf{u} \in \mathbb{R}^{B\times N\times T}$;
$\mathbf{a}_{\mathrm{leak}}, \boldsymbol{\kappa}, \boldsymbol{\theta} \in \mathbb{R}^{N}$.
Output: $\mathrm{pre}_{\mathrm{out}} \in \mathbb{R}^{B\times N\times T}$.

Load $\mathbf{a}_{\mathrm{leak},n}$, $\boldsymbol{\kappa}_n$, $\boldsymbol{\theta}_n$ into registers; carry $\leftarrow 0$.
for segment $= 0, \ldots, \lceil T/(\mathrm{TPB}\cdot c_{\mathrm{chunk}})\rceil - 1$ **do**
  Read $\mathrm{pre}_{\mathrm{in}}[b, n, \mathrm{prev}]$ and $\mathbf{u}[b, n, \mathrm{cur}]$ for c_{chunk} time elements per thread.
  $\mathrm{s}_{\mathrm{prev}} \leftarrow H(\mathrm{pre}_{\mathrm{in}}[b, n, \mathrm{prev}] - \boldsymbol{\theta}_n)$ in registers.
  $\mathrm{rhs} \leftarrow \mathbf{u}[b, n, \mathrm{cur}] + \boldsymbol{\kappa}_n \cdot \mathrm{s}_{\mathrm{prev}}$; $\mathrm{coef} \leftarrow -\mathbf{a}_{\mathrm{leak},n}$.
  if first step of the segment **then**
    $\mathrm{coef}[0] \leftarrow 0$; $\mathrm{rhs}[0] \mathrel{+}= \mathbf{a}_{\mathrm{leak},n} \cdot \mathrm{carry}$.
  end if
  ParallelReductionShared $(\mathrm{coef}, \mathrm{rhs})$.
  Write rhs to $\mathrm{pre}_{\mathrm{out}}[b, n, \mathrm{cur}]$; carry $\leftarrow \mathrm{rhs}[\mathrm{last}]$.
end for

A.3 Figures showing speed-up across various Picard iterations

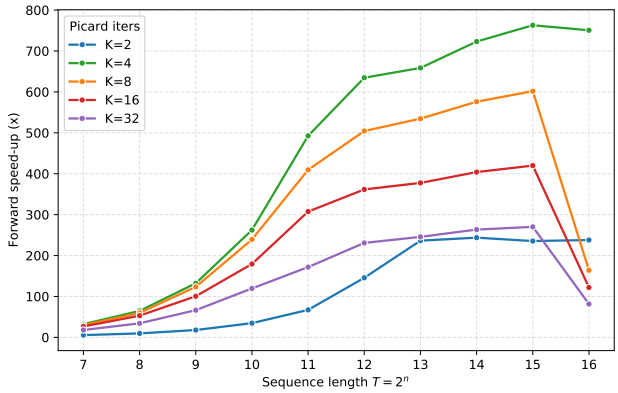

Figure 2: Forward speed-up across various Picard iterations.

A.4 Scope and Notation

This supplement provides the details referenced by Theorem 1 of the main paper. In particular, it gives two separate results: (i) a contraction proof for the Picard operator with a smooth threshold approximation, including the mean-value and telescoping arguments stated in the main text; and (ii) a direct fixed-point consistency proof for the right-continuous hard Heaviside operator used by the forward implementation. We also record the finite-precision qualification, the parallel backward recurrence, and implementation details of the fused CUDA kernel.

Let

$$\boldsymbol{u}_t := \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}, \qquad \boldsymbol{\kappa} := \boldsymbol{a}_{\mathrm{rec}} - \boldsymbol{a}_{\mathrm{leak}} \odot \boldsymbol{\theta}.$$

For a threshold function G applied elementwise, define the Picard operator Φ_G as follows. Given a candidate trajectory $\mathbf{pre} = (\mathbf{pre}_1, \ldots, \mathbf{pre}_T)$ and the fixed boundary state $\mathbf{pre}_0$, the output $\mathbf{pre}^+ = \Phi_G(\mathbf{pre})$ is the unique solution of

$$\begin{aligned} \mathbf{pre}_t^+ &= \boldsymbol{a}_{\mathrm{leak}} \odot \mathbf{pre}_{t-1}^+ + \boldsymbol{u}_t \\ &\quad + \boldsymbol{\kappa} \odot G(\mathbf{pre}_{t-1} - \boldsymbol{\theta}), \qquad t = 1, \ldots, T. \end{aligned} \tag{7}$$

with $\mathbf{pre}_0^+ = \mathbf{pre}_0$. Equation 7 is precisely one Picard sweep in Eqs. (5)–(6) of the main paper. The hard-threshold dynamics in Eq. (3) correspond to $G = H$, where $H(z) = \mathbf{1}\{z \geq 0\}$; the smooth analysis uses $G = H_\sigma$.

We equip trajectories with

$$\|\mathbf{pre}\|_\infty := \max_{1 \leq t \leq T} \max_{1 \leq n \leq N} |\mathrm{pre}_{t,n}|,$$

and set

$$\alpha := \|\boldsymbol{a}_{\mathrm{leak}}\|_\infty, \qquad \rho := \|\boldsymbol{\kappa}\|_\infty \beta, \qquad r := \alpha + \rho.$$

The boundary state is not part of the variable trajectory; hence two candidate trajectories always share the same $\mathbf{pre}_0$.

A.5 Proof of Smooth Picard Contraction

We first prove the contraction claim behind Theorem 1. This result concerns the smooth operator Φ_{H_σ}; the hard operator is treated separately in Section A.6.

Theorem 2 (Smooth Picard contraction)**.** *Assume that H_σ is β-Lipschitz and that $r = \alpha + \rho < 1$. Then Φ_{H_σ} is a strict contraction on $\ell^\infty(\{1, \ldots, T\}; \mathbb{R}^N)$. More precisely,*

$$\begin{aligned} &\|\Phi_{H_\sigma}(\mathbf{pre}) - \Phi_{H_\sigma}(\widetilde{\mathbf{pre}})\|_\infty \\ &\qquad \leq q\, \|\mathbf{pre} - \widetilde{\mathbf{pre}}\|_\infty, \\ &q := \frac{\rho}{1 - \alpha} \leq r < 1. \end{aligned} \tag{8}$$

Consequently, the Picard iterates converge to a unique fixed point $\mathbf{pre}_\sigma^\star$ and satisfy

$$\begin{aligned} &\|\mathbf{pre}^{(K)} - \mathbf{pre}_\sigma^\star\|_\infty \\ &\quad \leq q^K \|\mathbf{pre}^{(0)} - \mathbf{pre}_\sigma^\star\|_\infty \\ &\quad \leq r^K \|\mathbf{pre}^{(0)} - \mathbf{pre}_\sigma^\star\|_\infty. \end{aligned} \tag{9}$$

Thus the error is $\mathcal{O}((\alpha + \rho)^K)$. In particular, if $E_0 := \|\mathbf{pre}^{(0)} - \mathbf{pre}_\sigma^\star\|_\infty$, then any

$$K \geq \left\lceil \frac{\log(E_0/\varepsilon)}{-\log r} \right\rceil$$

is sufficient for error at most ε; this iteration bound does not depend on T.

Proof. Fix two candidate trajectories $\mathbf{pre}$ and $\widetilde{\mathbf{pre}}$, and write $\mathbf{pre}^+ = \Phi_{H_\sigma}(\mathbf{pre})$ and $\widetilde{\mathbf{pre}}^+ = \Phi_{H_\sigma}(\widetilde{\mathbf{pre}})$. Subtracting their instances of Eq. 7 gives, coordinatewise,

$$\begin{aligned} &\mathrm{pre}_{t,n}^+ - \widetilde{\mathrm{pre}}_{t,n}^+ \\ &\quad = a_{\mathrm{leak},n}(\mathrm{pre}_{t-1,n}^+ - \widetilde{\mathrm{pre}}_{t-1,n}^+) \\ &\qquad + \kappa_n \big[H_\sigma(\mathrm{pre}_{t-1,n} - \theta_n) - H_\sigma(\widetilde{\mathrm{pre}}_{t-1,n} - \theta_n) \big]. \end{aligned} \tag{10}$$

By the mean-value theorem, for some point $\xi_{t-1,n}$ between the two threshold arguments,

$$\begin{aligned} &H_\sigma(\mathrm{pre}_{t-1,n} - \theta_n) - H_\sigma(\widetilde{\mathrm{pre}}_{t-1,n} - \theta_n) \\ &\quad = H'_\sigma(\xi_{t-1,n})(\mathrm{pre}_{t-1,n} - \widetilde{\mathrm{pre}}_{t-1,n}), \end{aligned} \tag{11}$$

where $|H'_\sigma(\xi_{t-1,n})| \le \beta$. The same inequality follows directly from Lipschitz continuity when H_σ is not differentiable everywhere.

Let

$$D_t := \|\mathbf{pre}_t^+ - \widetilde{\mathbf{pre}}_t^+\|_\infty, \qquad E := \|\mathbf{pre} - \widetilde{\mathbf{pre}}\|_\infty.$$

Equations 10–equation 11 imply

$$D_t \le \alpha D_{t-1} + \rho E, \qquad D_0 = 0.$$

Telescoping the affine recurrence in Eq. (6) of the main paper yields

$$D_t \le \rho E \sum_{j=0}^{t-1} \alpha^j = \rho E \frac{1-\alpha^t}{1-\alpha} \le \frac{\rho}{1-\alpha} E. \tag{12}$$

Taking the maximum over t proves the first inequality in Eq. 8. Moreover,

$$(\alpha + \rho) - \frac{\rho}{1-\alpha} = \frac{\alpha(1-\alpha-\rho)}{1-\alpha} \ge 0, \tag{13}$$

so $q \le r < 1$. Hence Φ_{H_σ} is a strict contraction; Banach's fixed-point theorem gives a unique fixed point and geometric convergence. Applying the contraction inequality repeatedly gives Eq. 9. Because neither q nor its conservative upper bound r contains T, the number of iterations needed to reduce a fixed initial error below a fixed tolerance is independent of sequence length. □

Corollary 1 (A posteriori error bound)**.** *Under the assumptions of Theorem 2,*

$$\begin{aligned} &\|\mathbf{pre}^{(K)} - \mathbf{pre}^\star_\sigma\|_\infty \\ &\quad \le \frac{q^K}{1-q} \|\mathbf{pre}^{(1)} - \mathbf{pre}^{(0)}\|_\infty. \end{aligned}$$

Proof. The triangle inequality and contractivity give

$$\begin{aligned} &\|\mathbf{pre}^{(K+j+1)} - \mathbf{pre}^{(K+j)}\|_\infty \\ &\quad \le q^{K+j} \|\mathbf{pre}^{(1)} - \mathbf{pre}^{(0)}\|_\infty. \end{aligned}$$

Summing this geometric tail and taking the limit proves the claim. □

What is the smooth fixed point? Substituting $\mathbf{pre}^\star_\sigma = \Phi_{H_\sigma}(\mathbf{pre}^\star_\sigma)$ into Eq. 7 shows that it is exactly the sequential rollout of the recurrence with H_σ in place of H. This statement alone does not identify it with the hard-threshold trajectory. Equality between the smooth and hard fixed points additionally requires the threshold-consistency condition in Proposition 1; the next section separately establishes the exact fixed-point result for the hard operator itself.

A.6 Hard-Threshold Fixed-Point Consistency

The actual forward pass uses the right-continuous hard threshold H, not H_σ. Since H is discontinuous, a global Lipschitz contraction argument is unavailable. Exact fixed-point consistency instead follows from causality.

Theorem 3 (Hard-threshold fixed point and finite-step prefix recovery)**.** *Let Φ_H be Eq. 7 with $H(z) = \mathbf{1}\{z \ge 0\}$. Then:*

1. *Φ_H has a unique fixed point $\mathbf{pre}^\star_H$;*
2. *$\mathbf{pre}^\star_H$ is exactly the sequential hard-threshold rollout of Eq. (3) in the main paper; and*
3. *for any initial candidate trajectory, after K Picard sweeps the first $\min\{K, T\}$ states equal the sequential rollout in exact arithmetic.*

In particular, the complete trajectory is recovered after at most T sweeps.

Proof. Let $\widehat{\mathbf{pre}}_1, \ldots, \widehat{\mathbf{pre}}_T$ denote the sequential solution of Eq. (3), with the same fixed boundary state $\mathbf{pre}_0$. Consider arbitrary Picard iterates $\mathbf{pre}^{(k+1)} = \Phi_H(\mathbf{pre}^{(k)})$.

For $t = 1$, Eq. 7 depends only on the fixed boundary $\mathbf{pre}_0$ and the input $\boldsymbol{u}_1$. Therefore $\mathbf{pre}_1^{(1)} = \widehat{\mathbf{pre}}_1$. Suppose inductively that after k sweeps,

$$\mathbf{pre}_t^{(k)} = \widehat{\mathbf{pre}}_t, \qquad t = 1, \ldots, k.$$

The associated hard spike decisions are then also equal on this prefix because H is a single-valued, right-continuous function. During sweep $k+1$, the forcing at time $t \leq k+1$ uses only the candidate state at time $t-1 \leq k$. It is consequently identical to the sequential forcing, and the affine scan produces $\mathbf{pre}_t^{(k+1)} = \widehat{\mathbf{pre}}_t$ for $t \leq k+1$. Induction proves finite-step prefix recovery and the T-sweep bound.

It remains to prove fixed-point uniqueness. Let $\mathbf{pre}^\star = \Phi_H(\mathbf{pre}^\star)$. At $t = 1$, the fixed-point equation forces $\mathbf{pre}_1^\star = \widehat{\mathbf{pre}}_1$. If $\mathbf{pre}_j^\star = \widehat{\mathbf{pre}}_j$ for all $j < t$, then the hard spike at $t-1$, the forcing at t, and hence $\mathbf{pre}_t^\star$ all equal their sequential counterparts. Induction over time gives $\mathbf{pre}^\star = \widehat{\mathbf{pre}}$. Thus every fixed point equals the sequential rollout, proving both existence and uniqueness. □

Proposition 1 (Transfer from a smooth to the hard fixed point)**.** *Let $\mathbf{pre}_H^\star$ be the hard sequential rollout. If the smooth threshold is hard-consistent on that trajectory,*

$$H_\sigma(\mathrm{pre}_{H,t,n}^\star - \theta_n) = H(\mathrm{pre}_{H,t,n}^\star - \theta_n) \quad \textit{for all } t, n, \tag{14}$$

then $\mathbf{pre}_H^\star$ is a fixed point of Φ_{H_σ}. Under the contraction assumptions of Theorem 2, it is therefore the unique smooth fixed point as well.

Proof. Under Eq. 14, the forcing assembled by Φ_{H_σ} at $\mathbf{pre}_H^\star$ equals the hard sequential forcing at every time step. Hence $\Phi_{H_\sigma}(\mathbf{pre}_H^\star) = \mathbf{pre}_H^\star$. Uniqueness follows from Theorem 2. □

Proposition 1 applies, for example, to a clipped smooth threshold that equals H outside a transition band, provided the hard trajectory never enters that band. For an ordinary sigmoid that only approaches H, the smooth and hard fixed points need not be exactly identical; the smooth theorem then supplies an analytical convergence result, while Theorem 3 supplies the exact hard-dynamics statement. This separation makes explicit the respective scopes of the smooth-convergence and hard-dynamics claims summarized in Theorem 1 of the main paper.

Finite-Precision Qualification Define the hard trajectory's threshold margin

$$\gamma := \min_{t,n} |\mathrm{pre}_{H,t,n}^\star - \theta_n|. \tag{15}$$

If a computed trajectory $\widetilde{\mathbf{pre}}$ satisfies $\|\widetilde{\mathbf{pre}} - \mathbf{pre}_H^\star\|_\infty < \gamma$, then it induces exactly the same spike train as the hard sequential rollout. More generally, the same conclusion holds whenever the combined numerical error in every threshold argument is smaller than its local margin. If a state lies on, or very near, a threshold, an arbitrarily small rounding or finite-iteration error may flip the spike. This explains why exact arithmetic fixed-point consistency and empirical finite-precision spike accuracy are distinct notions.

The right-continuous convention $H(0) = 1$ is used by both the reference rollout and the fused kernel. Matching this boundary convention is necessary: replacing ≥ 0 by > 0 would create a systematic discrepancy exactly at threshold.

A.7 Parallel Backward Pass

The forward hard threshold is paired with a surrogate derivative in the backward pass. Once a forward trajectory has been selected, let $\boldsymbol{g}_t$ collect the local loss derivative at time t, and let $\boldsymbol{J}_t$ denote the diagonal surrogate Jacobian of the recurrent state update. For the neuron-wise recurrence used in the main paper, the BPTT adjoint has the form

$$\boldsymbol{\lambda}_{t-1} = \boldsymbol{g}_{t-1} + \boldsymbol{J}_t \odot \boldsymbol{\lambda}_t, \qquad \boldsymbol{\lambda}_T = \boldsymbol{g}_T. \tag{16}$$

Equation 16 is an affine first-order recurrence in reverse time. Applying the same associative affine-scan operator to the time-reversed sequence computes all adjoints in $\Theta(\log T)$ parallel depth. The scan changes only the evaluation order; it computes the same surrogate-BPTT recurrence as a sequential reverse-time loop, up to floating-point reduction order.

After the adjoints are available, parameter gradients are reductions over batch and time. For the pre-threshold update, representative terms are

$$\nabla_{\boldsymbol{W}} L = \sum_{b,t} \boldsymbol{\lambda}_{b,t} \boldsymbol{x}_{b,t}^{\mathsf{T}}, \qquad \nabla_{\boldsymbol{b}} L = \sum_{b,t} \boldsymbol{\lambda}_{b,t}, \tag{17}$$

$$\nabla_{\boldsymbol{a}_{\mathrm{rec}}} L = \sum_{b,t} \boldsymbol{\lambda}_{b,t} \odot \boldsymbol{s}_{b,t-1}, \qquad \nabla_{\boldsymbol{a}_{\mathrm{leak}}} L = \sum_{b,t} \boldsymbol{\lambda}_{b,t} \odot \boldsymbol{v}_{b,t-1}. \tag{18}$$

The implementation uses detach-reset surrogate training: the hard reset is retained in the forward pass, while its spike-dependent derivative is detached to avoid repeatedly amplifying a large surrogate slope over long horizons. This choice affects the surrogate Jacobian $\boldsymbol{J}_t$, but not the associative scan structure of Eq. 16.

A.8 Fused CUDA Kernel Details

A.8.1 Data Layout and Work Assignment

The fused kernel consumes the previous Picard trajectory $\mathrm{pre}_{\mathrm{in}}$, the precomputed input drive $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, and the neuron-wise parameters; it writes $\mathrm{pre}_{\mathrm{out}}$. All trajectory tensors use the contiguous layout (B, N, T). The CUDA grid is $(1, N, B)$, so one thread block owns one independent batch–neuron pair (b, n) and scans its complete time axis. The implementation uses 1024 threads per block. Each thread holds a short contiguous chunk in registers: two fp32 elements or four fp64/fp16/bf16 elements in the current implementation.

For sequences longer than one block's temporal capacity, the same CUDA block loops over

$$S = \left\lceil \frac{T}{\mathrm{TPB}\, c_{\mathrm{chunk}}} \right\rceil$$

segments. The last solved state of one segment is stored in shared memory and inserted as the boundary condition of the next segment. This is a *cross-segment carry within one CUDA block*; no second global kernel is required for the boundary propagation.

A.8.2 One Fused Picard Sweep

For each resident time element, the kernel performs the following operations:

1. read the shifted previous trajectory and input drive;
2. compute the right-continuous hard spike $s_{t-1} = \mathbf{1}\{\mathrm{pre}_{t-1} \geq \theta\}$;
3. assemble $c_t = (\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}) + \boldsymbol{\kappa} \odot \boldsymbol{s}_{t-1}$ in registers;
4. solve $\mathrm{pre}_t = \boldsymbol{a}_{\mathrm{leak}} \odot \mathrm{pre}_{t-1} + c_t$ by the shared-memory affine scan; and
5. write only the new pre-threshold trajectory.

Thus the full-length spike tensor and forcing tensor are never materialized in global memory. The old and new pre-threshold trajectories are the only full-length state tensors required by a sweep.

Within a segment, each affine equation is represented by a pair (a, c) for the map $z \mapsto az + c$. Composition is associative:

$$(a_2, c_2) \circ (a_1, c_1) = (a_2 a_1,\ a_2 c_1 + c_2), \tag{19}$$

which permits a Hillis–Steele-style balanced scan. Local chunk work is kept in registers, warp-level partial results use shuffle operations, and inter-warp partial results use shared memory. Tail elements are replaced by the neutral affine element so that non-power-of-two sequence lengths remain valid.

Table 4: Temporal dependency depth and principal trajectory storage. The table reports parallel depth.

Method	Temporal depth	Full-length temporaries
Sequential rollout	$\Theta(T)$	state/output
One Picard sweep	$\Theta(\log T)$	old/new pre
K Picard sweeps	$\Theta(K \log T)$	old/new pre
Backward affine scan	$\Theta(\log T)$	adjoint/output

Table 5: Spike accuracy averaged over the tested sequence lengths. The $K = 4$ setting is selected as the practical fidelity–speed operating point.

Picard iterations K	2	4	8	16	32
Mean SpikeAcc (%)	90.8	87.6	87.3	89.8	92.6

A.8.3 Complexity and Memory

The implementation supports arbitrary lengths up to $T = 65{,}535$ in the reported benchmark. At very long lengths, arithmetic depth remains logarithmic, but throughput can saturate because the working set leaves the cache-friendly regime, HBM traffic dominates, and more temporal segments require carry propagation. Non-power-of-two tails can additionally reduce utilization.

A.9 Measurement and Reproducibility Details

A.9.1 Speed and Fidelity Protocol

The speed experiment in the main paper uses a single data-center GPU with CUDA 12.1 and fp32. The fixed dimensions are $B = 8$, $N = 128$, and $D_{\text{in}} = 64$; sequence lengths range from 128 to $65{,}535$. Each timing is averaged over five measured runs after three warm-up runs. The reference is the same cell evaluated by sequential BPTT. Forward speed-up and full-step speed-up are respectively

$$\frac{t_{\text{sequential,fwd}}}{t_{\text{PPT,fwd}}} \quad \text{and} \quad \frac{t_{\text{sequential,fwd+bwd}}}{t_{\text{PPT,fwd+bwd}}}.$$

CUDA synchronization brackets each measured region.

Spike accuracy compares the binary trajectories from PPT and the sequential hard-threshold rollout:

$$\text{SpikeAcc} = \frac{100}{BNT} \sum_{b,n,t} \mathbf{1}\{s^{\text{PPT}}_{b,n,t} = s^{\text{seq}}_{b,n,t}\}. \tag{20}$$

The average spike accuracies reported for $K \in \{2, 4, 8, 16, 32\}$ are shown in Table 5. Their non-monotonicity reflects the discontinuity of the hard threshold and finite-precision threshold crossings; it should not be interpreted as the smooth contraction error of Theorem 2.

A.9.2 Training Configuration

Unless otherwise specified in the main paper, PicardSNN uses three recurrent layers of width 512, $K = 4$ Picard sweeps, AdamW, cosine learning-rate decay with warm-up, and exponential moving average with decay 0.999. Fast-sigmoid surrogate derivatives are used for most tasks; Gaussian surrogates are used for the longest-range settings. The pre-threshold reformulation changes how the trajectory is evaluated, not the hard threshold or subtractive-reset rule in the forward model.

The benchmarked recurrence is neuron-wise: $\boldsymbol{a}_{\text{rec}}$ is a vector, equivalently the diagonal matrix $\text{diag}(\boldsymbol{a}_{\text{rec}})$. This is why both the forward affine solve and its reverse-time surrogate adjoint admit diagonal scans. The submitted results use this neuron-wise recurrence. Section A.11 shows that dense recurrent spike mixing can still be placed in the forcing term without changing the diagonal temporal scan.

A.10 Extensions to LIF Variants

PPT is not tied to a single LIF equation. Its essential requirement is that, after the spike-dependent quantities from the previous Picard iterate are frozen, the remaining temporal state update is an affine recurrence that admits an associative scan. This section gives three representative extensions.

A.10.1 Adaptive-Threshold LIF

Let $\boldsymbol{b}_t^{\mathrm{ad}}$ be a spike-triggered adaptation state,

$$\boldsymbol{b}_t^{\mathrm{ad}} = \boldsymbol{\rho}_{\mathrm{ad}} \odot \boldsymbol{b}_{t-1}^{\mathrm{ad}} + \boldsymbol{\beta}_{\mathrm{ad}} \odot \boldsymbol{s}_{t-1}, \tag{21}$$

$$\boldsymbol{s}_t = H(\mathbf{pre}_t - \boldsymbol{\theta} - \boldsymbol{b}_t^{\mathrm{ad}}). \tag{22}$$

Here adaptation changes the firing threshold, while subtractive reset uses the base threshold $\boldsymbol{\theta}$ as in Eq. 7. For Picard iteration $k+1$, first evaluate the spikes from $(\mathbf{pre}^{(k)}, \boldsymbol{b}^{\mathrm{ad},(k)})$ and hold them fixed. The adaptation trajectory in Eq. 21 and the membrane trajectory in Eq. 7 then become two diagonal affine recurrences. Each is solved by a logarithmic-depth scan, after which the spike sequence is updated. Thus adaptive LIF admits an alternating Picard design rather than a return to sequential temporal evaluation. Our codebase contains a non-fused forward path for this alternating solve; a complete fused two-state forward/backward kernel and its evaluation are outside the submitted results.

A.10.2 Delayed Recurrent LIF

A delayed LIF cell may receive spikes from a set of delays $\mathcal{D}$:

$$\begin{aligned}\mathbf{pre}_t = \boldsymbol{a}_{\mathrm{leak}} \odot \mathbf{pre}_{t-1} + \boldsymbol{u}_t \\ - \boldsymbol{a}_{\mathrm{leak}} \odot \boldsymbol{\theta} \odot \boldsymbol{s}_{t-1} + \sum_{d \in \mathcal{D}} \boldsymbol{A}_d \boldsymbol{s}_{t-d}.\end{aligned} \tag{23}$$

At a Picard step, every delayed spike $\boldsymbol{s}_{t-d}^{(k)}$ is known, so the entire sum is part of the forcing sequence. The only unresolved temporal coefficient remains the diagonal leak. Consequently, fixed delays, multiple delay taps, and differentiable interpolations of learnable delays all preserve the same diagonal affine scan; only forcing assembly changes.

A.10.3 PLIF and Multi-Trace GLIF

PLIF learns the membrane time constant, equivalently the entries of $\boldsymbol{a}_{\mathrm{leak}}$. Since these entries remain neuron-wise, PPT applies without structural modification. A broader GLIF-style cell can include additional linear traces

$$\boldsymbol{z}_t^{(\ell)} = \boldsymbol{a}_\ell \odot \boldsymbol{z}_{t-1}^{(\ell)} + \boldsymbol{c}_\ell \odot \boldsymbol{s}_{t-1}, \qquad \ell = 1, \ldots, L, \tag{24}$$

that enter the membrane or threshold through fixed affine couplings. Once $\boldsymbol{s}^{(k)}$ is frozen, each trace and the membrane are affine recurrences. They can be scanned separately when their transitions are diagonal, or jointly as a small block-affine state. This provides a direct route from LIF to richer multi-timescale neuron models while retaining Picard-parallel temporal evaluation. Non-affine trace interactions can instead be frozen in an outer Picard step, provided the resulting iteration remains stable.

A.11 Dense Recurrent Spike Mixing and Applicability

A.11.1 Dense $\boldsymbol{W}_{\mathrm{rec}}$ Remains a Forcing Term

Let $\boldsymbol{A}_{\mathrm{leak}} = \mathrm{diag}(\boldsymbol{a}_{\mathrm{leak}})$ and $\boldsymbol{\Theta} = \mathrm{diag}(\boldsymbol{\theta})$. Replacing neuron-wise self-feedback by a dense recurrent matrix gives the exact pre-threshold recurrence

$$\begin{aligned}\mathbf{pre}_t = \boldsymbol{A}_{\mathrm{leak}} \mathbf{pre}_{t-1} + \boldsymbol{u}_t \\ + (\boldsymbol{W}_{\mathrm{rec}} - \boldsymbol{A}_{\mathrm{leak}} \boldsymbol{\Theta}) \boldsymbol{s}_{t-1}.\end{aligned} \tag{25}$$

During Picard sweep $k+1$, define

$$\begin{aligned} \boldsymbol{c}_t^{(k)} := \boldsymbol{u}_t \\ + (\boldsymbol{W}_{\mathrm{rec}} - \boldsymbol{A}_{\mathrm{leak}}\boldsymbol{\Theta})\boldsymbol{s}_{t-1}^{(k)}. \end{aligned} \tag{26}$$

Although computing $\boldsymbol{c}_t^{(k)}$ mixes neurons densely, it does not introduce an unresolved temporal dependency: it is fully known before the affine solve. The remaining recurrence is

$$\mathbf{pre}_t^{(k+1)} = \boldsymbol{A}_{\mathrm{leak}}\mathbf{pre}_{t-1}^{(k+1)} + \boldsymbol{c}_t^{(k)}, \tag{27}$$

which has the same diagonal temporal coefficient as the submitted model. Therefore, the dense matrix multiplication is used to assemble the forcing, after which the existing diagonal CUDA scan can solve the complete time axis. The reverse-time surrogate adjoint admits the same decomposition. An outer adjoint Picard iteration freezes the dense $\boldsymbol{W}_{\mathrm{rec}}^{\mathsf{T}}$ contribution as forcing, and each iteration uses one reverse diagonal scan for the leak term. Thus this extension requires convergence of both the forward and adjoint Picard iterations; it is not reduced to a single reverse scan.

A.11.2 Boundary of the Diagonal CUDA Kernel

The diagonal CUDA kernel described in Section A.8 takes the coefficient vector $\boldsymbol{a}_{\mathrm{leak}}$ and a forcing tensor $\boldsymbol{C} \in \mathbb{R}^{B\times N\times T}$, and solves all states of $\mathbf{pre}_t = \boldsymbol{a}_{\mathrm{leak}} \odot \mathbf{pre}_{t-1} + \boldsymbol{c}_t$ by an associative affine scan. Its use therefore has two stages: assemble $\boldsymbol{C}$ from the current Picard iterate, then invoke the scan with $\boldsymbol{a}_{\mathrm{leak}}$. The kernel requires only the coefficient multiplying the unknown previous state to be diagonal. The forcing may contain dense recurrent mixing, delayed spikes, adaptation, external inputs, or any other quantity already fixed for the current Picard sweep. Hence a dense $\boldsymbol{W}_{\mathrm{rec}}$ does not by itself invalidate the kernel.

If $\boldsymbol{A}_{\mathrm{leak}}$ itself is dense, the present diagonal kernel is no longer the correct solver. The algebraic PPT formulation still extends to this case because affine maps $(\boldsymbol{A}_t, \boldsymbol{c}_t)$ compose associatively as

$$\begin{aligned} (\boldsymbol{A}_2, \boldsymbol{c}_2) \circ (\boldsymbol{A}_1, \boldsymbol{c}_1) \\ = (\boldsymbol{A}_2\boldsymbol{A}_1,\ \boldsymbol{A}_2\boldsymbol{c}_1 + \boldsymbol{c}_2). \end{aligned} \tag{28}$$

A separately designed matrix-valued or structured block-scan kernel could therefore replace the diagonal scan. Such an extension must re-evaluate its arithmetic cost and numerical stability, but it does not alter the underlying Picard-parallel formulation.

A.11.3 Boundary of PPT

A useful general form for PPT is

$$\boldsymbol{z}_t = \boldsymbol{A}_t\boldsymbol{z}_{t-1} + \boldsymbol{c}_t(\mathcal{S}(\boldsymbol{z}_{1:T}))\,, \tag{29}$$

where $\mathcal{S}$ collects threshold, reset, or other nonlinear events. PPT freezes $\mathcal{S}$ at the previous iterate and scans the resulting affine recurrence. It is therefore applicable whenever (i) this frozen update has an associative parallel solver and (ii) the induced fixed-point iteration is stable, either by a contraction condition, causal finite-step recovery, or a damped iteration satisfying an appropriate local spectral condition. Diagonal, block-diagonal, and dense affine transitions differ in kernel cost rather than in this principle. For additional nonlinear states, one may augment $\boldsymbol{z}_t$ or alternate several affine scans, as in the adaptive LIF construction above. This criterion covers a broad class of hybrid spiking dynamics while making explicit which component must be redesigned when the state transition changes.

A.12 Adaptive Picard Iteration Budget

A fixed K is convenient for benchmarking, but different samples, layers, and training stages need not converge at the same rate. Let

$$R_k := \frac{\|\mathbf{pre}^{(k)} - \mathbf{pre}^{(k-1)}\|_\infty}{\|\mathbf{pre}^{(k)}\|_\infty + \varepsilon_0}, \tag{30}$$

$$F_k := \frac{1}{BNT}\sum_{b,n,t} \mathbf{1}\{s_{b,n,t}^{(k)} \neq s_{b,n,t}^{(k-1)}\} \tag{31}$$

measure trajectory change and spike-decision change, respectively. An adaptive policy performs at least $K_{\min}$ sweeps and stops once

$$R_k \le \tau_{\mathrm{pre}} \quad \text{and} \quad F_k \le \tau_s \tag{32}$$

for a small number of consecutive sweeps, with $K_{\max}$ as a deterministic fallback. The test may be applied per layer or per mini-batch; bucketing similar iteration counts retains regular GPU execution.

For the smooth contractive operator, the current-iterate increment also gives the certified bound

$$\begin{aligned} &\|\mathbf{pre}^{(k)} - \mathbf{pre}^{\star}_{\sigma}\|_\infty \\ &\quad \le \frac{q}{1-q}\|\mathbf{pre}^{(k)} - \mathbf{pre}^{(k-1)}\|_\infty. \end{aligned} \tag{33}$$

For the hard operator, F_k directly monitors whether the discrete decisions have stabilized, while R_k protects against hidden state drift. With nonzero tolerances these are practical stopping diagnostics, not a certificate that the hard fixed point has been reached. Adaptive K changes only how many identical Picard sweeps are executed; it does not change the fixed-point equations or the scan kernel. It therefore provides a natural path to spending fewer sweeps on easy trajectories and additional sweeps only where refinement is needed.

A.13 Interpretation and Limitations of the Guarantees

For clarity, the theoretical and empirical statements have different scopes:

- The algebraic pre-threshold reformulation is exact for the hard LIF recurrence in exact arithmetic.
- The sequence-length-independent geometric rate is a theorem for the smooth operator under $\alpha + \rho < 1$.
- The hard operator has the exact sequential rollout as its unique fixed point, and causality guarantees exact prefix growth and recovery within at most T sweeps; a global hard-threshold contraction rate is not claimed.
- The choice $K = 4$ and its nearly length-stable spike accuracy are empirical properties of the tested finite-precision implementation, not a corollary of smooth contraction alone.
- Exact spike equality in finite precision additionally requires a nonzero threshold margin larger than the accumulated numerical error.

These distinctions reconcile the smooth convergence analysis with the exact hard-threshold dynamics implemented by PicardSNN.