Timing Is All You Need: SpikeCore, Learnable Delays, and Gain Control for Neuromorphic Classification

Mohsen Kamelian Rad* **Sotiris Moschoyiannis** **Roman Bauer**

NICE Research Group
Faculty of Engineering and Physical Sciences
University of Surrey, Guildford, UK
*m.kamelianrad@surrey.ac.uk

Abstract

Spiking neural networks (SNNs) offer a compelling path to energy-efficient, biologically plausible computation. Yet, a persistent gap remains: fully spiking models are difficult to train and incompatible with standard deep learning frameworks, while ANN-to-SNN conversion and hybrid approaches sacrifice either biological plausibility or deployment efficiency. We address this gap by introducing SpikeCore, a recurrent spiking unit designed to be natively compatible with ANN training frameworks, differentiable, gradient-friendly, and composable within standard pipelines, while retaining event-driven, spike-based computation. SpikeCore achieves this through learnable membrane decay and spike-history-dependent threshold adaptation, enabling learnable temporal dynamics under surrogate-gradient training. Building on SpikeCore, we present a full spatio-temporal pipeline comprising: a two-stage learnable temporal delay hierarchy motivated by differential retinal ganglion latency and axonal conduction delays; a Multi-Branch Squeeze-Excitation (MBSE) backbone that extracts spatial features at multiple scales with channel-wise recalibration, using $11.8\times$ fewer parameters than equivalent residual architectures; and a Parameter-Free Temporal Gain Control (PFTGC) module that modulates each channel's temporal response without added parameters. Evaluated across five diverse benchmarks, the proposed architecture converges 5–$6\times$ faster and requires $\approx 3\times$ fewer timesteps than competitive baselines while matching or surpassing state-of-the-art models on accuracy.

1 Introduction

The dominant paradigm in deep learning relies on dense matrix multiplications in which every neuron participates in every forward pass, regardless of the information content of the input. Biological neural circuits operate under a fundamentally different principle: most neurons are silent most of the time, and computation is triggered only by discrete electrical events called spikes Mainen and Sejnowski [1995]. This event-driven, temporally sparse regime underlies the remarkable energy efficiency of the brain (20 Watts) relative to its computational capacity, and it is precisely this principle that neuromorphic computing seeks to replicate Schuman et al. [2022]. Because energy consumption in spiking systems scales with the number of spikes rather than with overall network size, the energy-per-inference advantage can be substantial when activity is sparse Zhou et al. [2023]. In addition to this efficiency, spiking networks naturally support low-latency event-driven computation, robust temporal data processing, and improved resilience to noisy real-world inputs, making neuromorphic hardware especially attractive for battery-powered, always-on, and edge-deployed applications.

Despite this promise, neuromorphic computing faces two overarching practical barriers Kudithipudi et al. [2025]: *integration*: making neuromorphic chips plug into the existing AI-accelerator ecosystem as smoothly as GPUs; and *tooling*: building an open-source compiler stack that maps models written in PyTorch or TensorFlow onto neuromorphic hardware through a shared intermediate representation.

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

Both challenges reduce to the same question: how do we build spiking systems that behave like standard deep learning modules during training, yet produce event-driven spike trains at inference?

Recent work has significantly closed the accuracy gap of SNNs with ANNs by adapting transformer architectures to the spiking domain Zhou et al. [2023], Shi et al. [2024], demonstrating that deep learning advances can translate to neuromorphic settings. However, these approaches introduce substantial model complexity and training overhead, a trajectory that partially undermines the original motivation for spiking computation. Alongside architectural adaptation, hybrid ANN–SNN models have emerged as a complementary direction Liu et al. [2024], Abdullah et al. [2025], but effectively integrating the two paradigms remains an open problem: ANNs operate on continuous activations with static computation, whereas SNNs process sparse, event-driven signals with explicit temporal dynamics Shrestha et al. [2022].

Our proposal. Rather than scaling up SNN architectures toward more complexity, we ask: *Can a hybrid spiking unit, one that trains like an ANN but computes like an SNN, enriched with learnable neuronal dynamics, open a new dimension of neuromorphic design that is simultaneously gradient-friendly, spike-sparse, and deployable?* We answer affirmatively. **SpikeCore** occupies a middle ground: differentiable and composable in standard frameworks, yet producing binary, event-driven spikes that permit sparse event-driven processing on suitable hardware. It moves beyond integrate-and-fire models by learnable decay and spike-history-dependent threshold adaptation, enabling richer, more stable temporal dynamics while remaining fully trainable via backpropagation.

The core unit is **SpikeCore**, a spiking recurrent module that functions as a computational interface between continuous ANN activations and discrete event streams:

$$\underbrace{\boldsymbol{h}_t \in \mathbb{R}^d}_{\text{continuous features}} \longrightarrow \underbrace{\text{SpikeCore}(\cdot)}_{\substack{\text{ANN-compatible} \\ \text{spiking unit}}} \longrightarrow \underbrace{\boldsymbol{s}_t \in \{0,1\}^d}_{\text{event-driven spikes}} \tag{1}$$

Unlike standard spiking neuron models (LIF, adLIF), SpikeCore is a self-contained recurrent module, analogous in role to a GRU, that produces neuromorphic-compatible binary spike trains while supporting end-to-end gradient-based training. Compared to the only closely related unit, the Spiking Neural Unit (SNU by Woźniak et al. [2020]), SpikeCore introduces per-neuron learnable membrane decay and spike-history-dependent threshold adaptation, yielding richer temporal dynamics and improved stability on neuromorphic benchmarks, as confirmed by direct ablations. SpikeCore as a standard `nn.Module`, trains on GPU, is compatible with neuromorphic execution, and integrates into existing workflows without modification, directly addressing both the integration and tooling challenges identified above.

Full pipeline. Built on SpikeCore, our pipeline overcomes key neuromorphic bottlenecks with four biologically inspired components, enabling fast convergence, fewer timesteps, and strong accuracy while preserving spiking paradigm. A two-stage **learnable temporal delay hierarchy** aligns input signals before feature extraction, motivated by differential ON/OFF retinal latency and axonal conduction delays. Spatial and spectral features are encoded by a **Multi-Branch Squeeze-Excitation (MBSE)** backbone for event-camera inputs, or a causal **InceptionTemporal** module for auditory spike inputs. A **Parameter-Free Temporal Gain Control (PFTGC)** module then modulates the temporal response of each channel via per-sample temporal z-score statistics, requiring zero additional parameters, before the SpikeCore stack performs integration and spike-based classification.

Contributions.

- **SpikeCore**: an intermediate recurrent unit with learnable decay and spike-history adaptation, implemented as a standard PyTorch module producing neuromorphic-compatible spike trains.
- **Dual learnable delay layers**: a polarity-level hard shift (STE) and a fully differentiable feature-level soft delay mixture (LearnableDelayConv), providing principled temporal alignment for event-based signals.
- **PFTGC**: a parameter-free temporal gain control module performing input-dependent channel-wise modulation via z-score gating at zero parameter cost.
- **MBSE backbone**: a multi-branch feature extractor with SE recalibration achieving $\approx 11.8\times$ fewer parameters than equivalent residual architectures.
- Evaluation on **five datasets** spanning event-based vision and auditory spike streams, matching or exceeding state-of-the-art SNNs while converging 5–$6\times$ faster and requiring $\approx 3\times$ fewer timesteps.

2 Related Work

Spiking neural networks and surrogate gradients. Training SNNs with backpropagation through time requires differentiable approximations to the Heaviside spike function. Surrogate gradient methods Neftci et al. [2019] have become the standard solution, enabling gradient-based optimisation of LIF and adaptive LIF neurons, and richer neuronal dynamics have been explored in various forms beyond these baselines. SpikeCore adopts a sigmoid-based STE surrogate but is distinguished not merely by its dynamics, learnable per-neuron decay and spike-frequency adaptation, but by its architectural role: a self-contained, composable unit that sits at the interface of ANN and SNN computation, accepting continuous activations, producing binary spikes, and remaining fully differentiable throughout. SpikeCore bridges ANN–SNN pipelines for a new design dimension.

ANN-to-SNN conversion and hybrid models. A prominent line of work converts pre-trained ANNs to SNNs by approximating ReLU activations with firing rates Rueckauer et al. [2017], Negi et al. [2024]. While effective, conversion methods typically require many timesteps to achieve accurate rate coding, and the resulting networks are not trained end-to-end. Hybrid ANN–SNN architectures Liu et al. [2024], Aydin et al. [2024], Zhao et al. [2022], Abdullah et al. [2025] instead train mixed continuous-spiking pipelines jointly, combining the representational power of ANNs with the energy efficiency of spiking units. Our framework follows this direction, but differs in that SpikeCore is designed as a drop-in recurrent primitive rather than an architectural scaffold, and our pipeline is motivated throughout by biologically-grounded temporal processing mechanisms.

Transformer-based SNNs. Spikformer Zhou et al. [2023], SpikingResformer Shi et al. [2024], and SpikCommander Wang et al. [2026] adapt self-attention to the spiking domain, achieving strong benchmark results. However, these architectures inherit the quadratic complexity of attention and require extensive training. Our approach instead introduces PFTGC, a parameter-free linear-complexity temporal modulation mechanism that achieves input-dependent channel weighting without attention matrices, and prioritises low-latency fast convergence over architectural scale.

Temporal delay learning in SNNs. Axonal conduction delays have long been recognised as a source of computational power in biological networks Swadlow [1985]. Dilated Convolution with Learnable Spacings (DCLS, Hassani et al. [2023]) operationalises this idea in a differentiable framework. Our LearnableDelayConv approximates DCLS via a softmax mixture over candidate delays with depthwise convolutional gains, and pairs it with a coarser polarity-level hard delay at the input stage, forming a two-stage biologically-motivated delay hierarchy not previously explored in SNN pipelines.

Neuromorphic benchmarks. DVSGesture Amir et al. [2017], SHD, and SSC Cramer et al. [2020] are standard benchmarks for event-based and spike-encoded classification. These datasets capture complementary challenges, 2D spatial spatio-temporal structure, 1D auditory frequency-time structure, and we evaluate on all three to demonstrate that our architecture generalises across modalities without dataset-specific tuning of its core components.

3 Method

3.1 SpikeCore: Intermediate Unit

The temporally modulated feature sequence $\tilde{\boldsymbol{H}} \in \mathbb{R}^{B\times T\times D}$ is processed by a stack of L **SpikeCore** units, a novel adaptive spiking recurrent cell, as shown in Fig 1, that differs from the SNU Woźniak et al. [2020] with three key modifications: (i) a learnable per-neuron membrane decay bounded to $(0, 1)$ via a sigmoid reparameterisation; (ii) a spike-history adaptation mechanism that dynamically raises the effective firing threshold following recent spiking activity, implementing spike-frequency adaptation analogous to slow potassium currents in biological neurons Benda and Herz [2003]; and (iii) a novel spike-based loss function introduced in Appendix C. The full architecture uses $L = 3$ layers with hidden dimensions $[HD1, HD2, K]$, where K is the number of output classes and the final layer serves as a spike-based classification head.

At each timestep t, the membrane potential of layer ℓ is updated as:

$$\boldsymbol{V}_t = g\left(\sigma_s(\boldsymbol{\phi}) \odot \boldsymbol{V}_{t-1} \odot (\mathbf{1} - \boldsymbol{o}_{t-1}) + \boldsymbol{x}_t^{(\ell)}\boldsymbol{W}_{\text{in}} + \frac{\boldsymbol{o}_{t-1}\boldsymbol{W}_{\text{rec}}}{\sqrt{N}+\epsilon}\right). \tag{2}$$

where $\boldsymbol{\phi} \in \mathbb{R}^N$ are learnable unconstrained decay parameters projected to $(0, 1)$ via $\sigma_s(\cdot)$, $g(\cdot) = \mathrm{LeakyReLU}(\cdot)$ is applied to the integrated potential, $\boldsymbol{W}_{\text{rec}} \in \mathbb{R}^{N\times N}$ is the recurrent weight matrix,

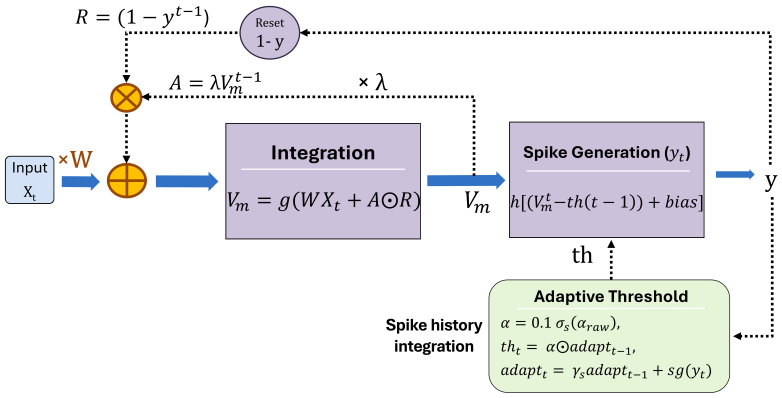


Figure 1: **Our proposed hybrid unit, SpikeCore.** The integration block calculates membrane potential (V_m) by passing the combination of weighted input (WX_t) and the decay term times reset ($A \times R$) through a leaky-relu activation function (g). The spike generation block then generates a spike if the membrane potential exceeds an adaptive threshold (see Appendix A for details). The optional recurrent term is not shown in this figure.

and the $\sqrt{N}$ normalisation mirrors the attention scaling of Vaswani et al. [2017] to stabilise recurrent dynamics.The reset term, $1 - \boldsymbol{o}_{t-1}$, resets the retained membrane contribution of neurons that fired at the previous timestep. The spike output is generated against a dynamic threshold that rises with recent activity:

$$\boldsymbol{o}_t = S\big(\boldsymbol{V}_t + \boldsymbol{b} - \boldsymbol{\alpha} \odot \mathbf{adapt}_{t-1}\big)\,, \qquad \mathbf{adapt}_t = 0.9\,\mathbf{adapt}_{t-1} + \mathrm{sg}(\boldsymbol{o}_t). \tag{3}$$

where $\mathcal{S}(\cdot)$ is a smooth surrogate combining the Heaviside function in the forward pass with sigmoid gradients in the backward pass, $\boldsymbol{b} \in \mathbb{R}^N$ is a learnable bias encoding the resting threshold, $\boldsymbol{\alpha} = 0.1\,\sigma_s(\boldsymbol{\alpha}_{\text{raw}})$ is a bounded per-neuron adaptation gain, $\boldsymbol{adapt}_t$ is a leaky spike-history trace, and $\mathrm{sg}(\cdot)$ is the stop-gradient operator. Full derivations, initialisation details, and a discussion of the biological analogues are provided in Appendix A.

3.2 Architecture Overview

All pipeline variants share a common four-stage design (Excluding learnable soft-delay stage):

$$\text{Input} \longrightarrow \underbrace{\text{Learnable Delay}}_{\text{Section 3.3}} \longrightarrow \underbrace{\text{Feature Extractor}}_{\text{Section 3.4}} \longrightarrow \underbrace{\text{PFTGC}}_{\text{Section 3.5}} \longrightarrow \underbrace{\text{SpikeCore Stack}}_{\text{Section 3.1}}$$

The four stages share the same design philosophy across all datasets but are instantiated with dataset-appropriate components, summarised in Table 9.

3.3 Learnable Temporal Delay

Motivated by biological latency differences, we introduce learnable delay layers before and after feature extraction. We use (i) a lightweight polarity-level hard delay with one learnable delay per channel,

$$d_c = \mathrm{clamp}(\lfloor \delta_c \rceil,\ 0,\ d_{\max})\,, \qquad \hat{x}_{b,t,c} = \begin{cases} x_{b,\,t-d_c,\,c} & t \geq d_c, \\ 0 & t < d_c. \end{cases} \tag{4}$$

and (ii) a feature-level **LearnableDelayConv** that models each channel's delay as a soft distribution over candidate shifts. Both enable the network to learn temporal alignment directly from data; Full derivation and biological interpretation are in Appendix J.

3.4 Feature Extractor

MBSE backbone. For 2D spatial inputs, each timestep is processed independently after reshaping to $\mathbb{R}^{BT\times 2\times H\times W}$. A strided convolutional stem is followed by four **Multi-Branch Squeeze-Excitation**

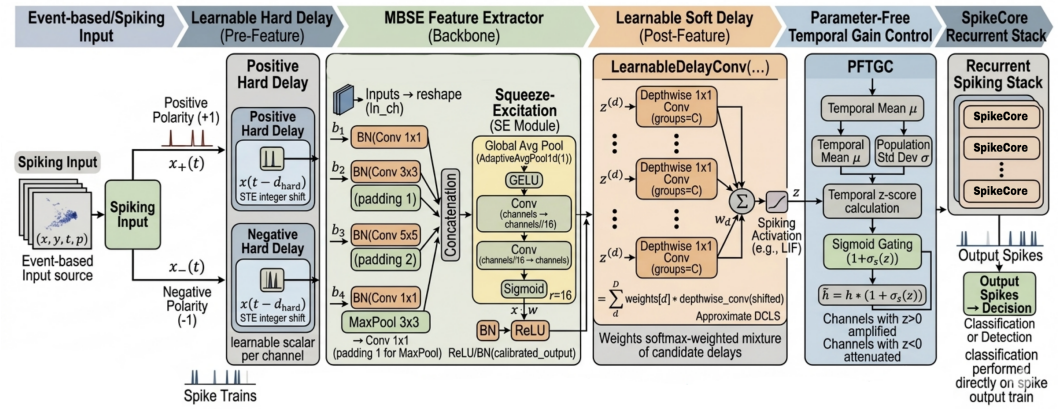


Figure 2: **Overview of the proposed 5-stage neuromorphic pipeline.** Our pipeline integrates bio-inspired learnable temporal delays with a high-performance MBSE backbone. By utilizing Parameter-Free Temporal Gain Control (PFTGC) and an efficient SpikeCore recurrent stack, the model achieves robust spatio-temporal feature extraction with minimal parameter overhead, supporting end-to-end gradient-based optimization. See Appendix H and J for details.

(MBSE) modules. Each MBSE module computes four parallel branches:

$$\boldsymbol{b}_1 = \mathrm{BN}(\mathrm{Conv}_{1\times1}(\boldsymbol{z})), \qquad \boldsymbol{b}_2 = \mathrm{BN}(\mathrm{Conv}_{3\times3}(\mathrm{Conv}_{1\times1}(\boldsymbol{z}))),$$
$$\boldsymbol{b}_3 = \mathrm{BN}(\mathrm{Conv}_{5\times5}(\mathrm{Conv}_{1\times1}(\boldsymbol{z}))), \qquad \boldsymbol{b}_4 = \mathrm{BN}(\mathrm{Conv}_{1\times1}(\mathrm{MaxPool}_{3\times3}(\boldsymbol{z}))), \tag{5}$$

concatenated and recalibrated by a Squeeze-and-Excitation gate Hu et al. [2018]:

$$\mathrm{MBSE}(\boldsymbol{z}) = \mathrm{ReLU}(\mathrm{BN}([\boldsymbol{b}_1\|\boldsymbol{b}_2\|\boldsymbol{b}_3\|\boldsymbol{b}_4] \odot \sigma_s(\boldsymbol{W}_2\,\mathrm{ReLU}(\boldsymbol{W}_1\,\mathrm{GAP}(\boldsymbol{u}))))), \tag{6}$$

where $\sigma_s(\cdot)$ is the sigmoid function, $\mathrm{GAP}(\cdot)$ denotes global average pooling, and $\boldsymbol{W}_1, \boldsymbol{W}_2$ are the SE bottleneck projections with reduction ratio $r = 16$. The 1×1 bottleneck before each spatial branch reduces FLOPs by approximately $3\times$ relative to direct full-channel convolutions. Global average pooling and flattening produce a D-dimensional feature vector per timestep. For 1D data, like SHD and SSC, the logic is same, but with 1D conv layers (see Appendix H).

3.5 Parameter-Free Temporal Gain Control (PFTGC)

The feature sequence $\boldsymbol{H} \in \mathbb{R}^{B\times T\times D}$ is modulated by PFTGC, which gives above-mean temporal responses greater gain without introducing learnable parameters. For each channel d, the temporal z-score is computed and passed through a sigmoid gate with a residual connection:

$$z_{b,t,d} = \frac{h_{b,t,d} - \mu_{b,d}}{\sigma_{b,d} + \epsilon}, \qquad \tilde{h}_{b,t,d} = h_{b,t,d}\big(1 + \sigma_s(z_{b,t,d})\big)\,, \tag{7}$$

Here, $\mu_{b,d}$ and $\sigma_{b,d}$ are computed independently for each sample b and channel d, reducing only over the temporal dimension. The standard deviation is Bessel-corrected, using denominator $T-1$; no statistics are pooled across samples. We use $\epsilon = 10^{-6}$. The residual gain $1 + \sigma_s(z_{b,t,d})$ lies between 1 and 2, assigning greater gain to above-mean responses. Very short sequences may yield noisier statistics. PFTGC has $O(BTD)$ complexity and zero learnable parameters.

4 Experiments

4.1 Problem Setup and Datasets

We used five different datasets for classification tasks: static images (F-MNIST Xiao et al. [2017]), tabular speech features (Isolet Fanty and Cole [1990]), Spiking Heidelberg Digits (SHD and SSC Cramer et al. [2020]), and DVS128Gesture Amir et al. [2017], introduced in Appendix D, with augmentations introduced in Appendix E, and the spike-based training objective in Appendix C.

For the event-based benchmarks, T denotes the number of bins used to discretise each recording over the evaluated duration. Increasing T refines temporal resolution rather than extending that duration. Our experiments evaluate pre-segmented classification, not continuous streaming.

Table 1: Evaluation of spiking unit variants across three datasets (image, tabular, and spiking) using a minimal two-layer architecture without augmentation or auxiliary components. This setup isolates the intrinsic temporal modeling capability of each unit. Results are reported as mean test accuracy $\pm$ 95% confidence interval over five runs. † Baselines reimplemented under identical conditions. See Appendix F for learning curves and training protocol that ensures fair comparisons.

Dataset	Model	Test Acc. (%)	Epochs	Timesteps
F-MNIST	SNU†	85.14 ± 0.8	70	20
	Adaptive SNU†	84.11 ± 0.9	70	20
	SpikeCore (ours)	90.12 ± 0.8	70	20
	SpikeCore-FCReadout (ours)	$\mathbf{91.23 \pm 0.7}$	70	20
Isolet	SNU†	92.66 ± 0.6	60	30
	Adaptive SNU†	91.86 ± 0.9	60	30
	SpikeCore (ours)	92.79 ± 0.6	60	30
	SpikeCore-FCReadout (ours)	$\mathbf{93.46 \pm 0.5}$	60	30
SHD	SNU†	71.12 ± 0.17	120	20
	Adaptive SNU†	70.48 ± 0.18	120	20
	SpikeCore (ours)	$\mathbf{79.16 \pm 0.15}$	120	20
	SpikeCore-FCReadout (ours)	77.52 ± 0.11	120	20

4.2 Main Results

Intrinsic capability of SpikeCore. Table 1 evaluates the intrinsic modeling capability of SpikeCore in a controlled setting, using a minimal two-layer architecture without augmentation or additional components. This setup isolates the contribution of the spiking unit itself across three distinct data regimes: F-MNIST, Isolet, and SHD.

Across all datasets, SpikeCore consistently outperforms both the standard SNU and its adaptive variant under identical training conditions. On F-MNIST, SpikeCore achieves 90.12%, improving over SNU by nearly $+5$ points, while the FC readout variant further increases performance to 91.23%. A similar trend is observed on Isolet, where SpikeCore provides modest but consistent gains over both baselines, with the FC readout reaching the best overall accuracy of 93.46%.

The most pronounced improvement appears on SHD, a temporally demanding spiking dataset, where SpikeCore reaches 79.16%, outperforming SNU by over $+8$ percentage points. Notably, the adaptive SNU baseline does not improve over the standard SNU across any dataset, that might be because of instability caused by introducing an added internal state or the unsophisticated adaptation mechanism. Besides, there are parameters to be tweaked by the user in Adaptive SNU, whereas in SpikeCore, all parameters, including the decays are trainable, diminishing the challenge of manual tuning. **Also see appendix B for detailed comparison of SpikeCore to SNU and adLIF neuron models**.

These results highlight the role of SpikeCore's combined dynamics: learnable membrane decay enables heterogeneous temporal integration across neurons, while spike-history-dependent adaptation modulates firing behavior based on recent activity. Together, these mechanisms yield more expressive temporal representations without increasing model depth, training budget, or sequence length.

Finally, the comparison between readout strategies reveals a dataset-dependent behavior. While the FC readout improves performance on F-MNIST and Isolet, the standard SpikeCore achieves higher accuracy on SHD, indicating that mean firing-rate representations may provide a more stable and informative signal for inherently spiking data.

Full architecture vs SOTA. Table 2 reports accuracy, timestep count, training epochs, and parameter count across three neuromorphic benchmarks. We group comparisons by dataset and include models spanning convolutional SNNs, transformer-based SNNs, recurrent SNNs, and delay-learning SNNs, providing a broad cross-architectural reference. The proposed MBSE backbone surpasses a 3.41M-parameter residual model by +2.74% while using only 0.29M parameters, suggesting a significantly more parameter-efficient model than conventional residual channel mixing (See Appendix I).

Table 2: Efficiency-centric comparison across neuromorphic benchmarks for the full architecture illustrated in Figure 2. We report accuracy alongside temporal discretisation and training budgets. † indicates estimated convergence when not explicitly reported. Compute scope distinguishes spiking backbones from architectures incorporating additional non-spiking processing. We use Hybrid in a broad computational sense, covering non-binary residual activations, continuous attention or gating, and ANN feature-extraction stages; the accompanying qualifier identifies the relevant component. This classification does not imply identical architectures or deployment costs. Intrinsic neuron-state updates, input encoding and final readout alone do not determine the label. Analog-output identifies the LIAF configuration of TA-SNN.

Dataset	Model	Compute scope	Accuracy (%)	Timesteps	Epochs	Params (M)
DVS128 Gesture	SEW-ResnetFang et al. [2021]	Hybrid—residual	97.9	16	300	0.13
	TA-SNN Yao et al. [2021]	Analog-output	98.6	60	100	–
	SPARTA Jang and Kim [2025]	Hybrid—attention	98.7	20	300	13.8
	Spikeformer Li et al. [2024]	Hybrid—attention	98.9	16	150	9.2
	Spikformer-V2 Zhou et al. [2024]	Hybrid—residual	98.2	16	200	2.6
	Ours	Hybrid—front end	$\mathbf{98.7 \pm 0.3}$	**16**	**50**	**0.39**
SHD	SDT Yao et al. [2024], Wang et al. [2026]	Spiking	89.6	100	500	1.77
	Spikformer Zhou et al. [2024]	Hybrid—residual	90.1	100	500	1.77
	DH-SNN Zheng et al. [2024]	Spiking	92.1	1000	500	0.05
	SE-adLIF Baronig et al. [2025]	Spiking	95.8 ± 0.56	250	300	0.45
	SpikCommander Wang et al. [2026]	Spiking	93.6	50	500	0.19
	Ours	Hybrid—front end	$\mathbf{94.02 \pm 0.65}$	**30**	**50**	**0.29**
SSC	SNN-CNN Sadovsky et al. [2023]	Hybrid	72.03	N/A	N/A	N/A
	RadLIF Bittar and Garner [2022]	Spiking	77.4	250	100-300†	3.9
	cAdLIF Deckers et al. [2024]	Spiking	77.5	100	100	0.35
	SDT Yao et al. [2024], Wang et al. [2026]	Spiking	79.8	100	300	2.57
	DCLS-Delays Hammouamri et al. [2024]	Spiking	80.69 ± 0.21	100	N/A	2.5
	Spikformer Zhou et al. [2024], Wang et al. [2026]	Hybrid—residual	80.18	100	300	2.57
	Pfa-SNN Sun et al. [2025]	Hybrid—gating	80.18	100	200-500†	0.71
	SpikCommander Wang et al. [2026]	Spiking	76.2	30	300	1.12
	Ours	Hybrid—front end	**77.6**	**30**	**200**	**2.1**
	Ours	Hybrid—front end	$\mathbf{80.24 \pm 0.2}$	**50**	**200**	**2.9**

To make the efficiency argument precise, Table 3 distils Table 2 into ratio form. For each baseline, we report the accuracy delta relative to our model, and the factor by which that baseline exceeds our timestep budget, epoch budget, and parameter count. A ratio greater than 1.0 in any column indicates that the competing model consumes more of that resource than ours. Across all three benchmarks, no model achieves a better accuracy-efficiency trade-off simultaneously on all three axes. The single case where a competitor achieves higher accuracy (SE-adLIF on SHD, +1.78%) does so at $8.3\times$ the timestep cost and $6\times$ the epoch budget, a resource regime incompatible with the latency and energy constraints of neuromorphic deployment.

Learning Curves. Figure 3 illustrates the learning curves on SHD. The solid line represents a representative run, while the shaded region indicates ± 1 standard deviation across all runs. Mean curves can look overly smooth and unrealistic, so a real run is included to show instability and convergence behaviour while the std band still shows robustness.

The learning curves show a consistent, stable convergence pattern: performance improves rapidly during the initial training phase, reaching approximately 90% of the final accuracy within the first 10 epochs, and then transitions into a smooth plateau around the final performance level ($\approx 94\%$).

Table 3: Efficiency summary relative to our model across three neuromorphic benchmarks. **Acc** Δ is the competitor accuracy minus ours (negative = ours is higher). **TS ratio**, **Epoch ratio**, and **Param ratio** express the competitor resource divided by ours; values >1.0 indicate the competitor consumes *more* of that resource. † epoch count estimated from training protocol of same model family. – indicates not reported. *Verdict* summarises the overall efficiency–accuracy trade-off.

Dataset	Baseline	Acc Δ (%)	TS ratio	Epoch ratio	Param ratio	Verdict
DVSGesture	SPARTA	0.0	1.3×	6.0×	35.4×	**Win** by params and epochs.
	Spikformer-V2	−0.5	1.0×	4.0×	6.7×	**Win**.
	Spikeformer	+0.2	1.0×	3.0×	23.6×	**Win** by params and epochs; negligible performance difference.
SHD	SpikCommander	−0.4	1.7×	10.0×	0.7×	**Win**.
	SDT	−4.4	3.3×	10.0×	6.1×	**Win**.
	SE-adLIF	+1.8	8.3×	6.0×	1.6×	**Win** on efficiency. Heavy resource and latency for 1.8% improvement.
SSC	SpikCommander^{30T}	−1.4	1.0×	1.5×	0.53×	**Win** by epoch and accuracy, but more params.
	RadLIF	−0.2	8.3×	†∼1.0×	1.9×	**Win**.
	DCLS-Delays	+0.5	2.0×	–	0.86×	**0.5**% advantage for **2**× more timesteps; epoch count unreported.
	Spikformer	−0.1	2.0×	1.5×	0.88×	**Win** by timesteps and epoch.

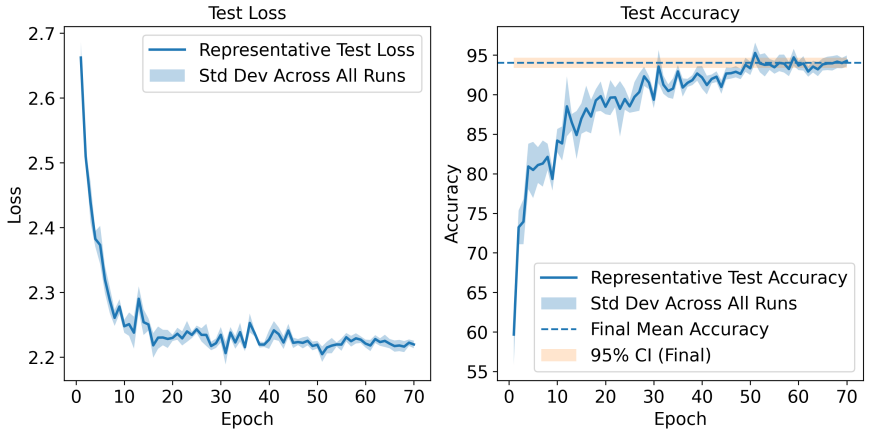


Figure 3: Learning curves of test accuracy across epochs. The solid line corresponds to a representative run (median final accuracy), while the shaded region denotes ± 1 standard deviation computed across all runs at each epoch. The dashed horizontal line and shaded band indicate the final evaluation mean and 95% confidence interval.

Importantly, extending training beyond convergence does not lead to performance degradation, indicating no observable overfitting. Additional analysis of convergence speed across datasets is provided in the appendix G.

4.3 Surrogate-gradient sensitivity and firing activity

To examine the interaction between surrogate gradients and learnable neuron dynamics, we conducted a supplementary SSC diagnostic run at $T = 50$. The model was trained for 50 epochs using one random seed. This diagnostic run is separate from the main benchmark results.

At fixed weights, we varied the sigmoid surrogate slope over $s \in \{0.5, 1, 2\}$ while preserving the hard forward spike function. Across 50 minibatches comprising 1,600 test examples, all sampled decay, adaptation, and synaptic-weight gradient groups were present and finite. Relative to $s = 1$, median

gradient-direction cosine similarities were 0.904–0.974 for decay and 0.743–0.986 for adaptation (Table 11). Hidden-layer adaptation was the most sensitive group, indicating that surrogate choice can meaningfully affect the local learning signal despite unchanged forward activity. Figure 6 reports the corresponding gradient magnitudes.

We also measured firing activity over all 20,382 test recordings. The hidden and output layers fired on approximately 60% and 8% of neuron–timestep opportunities, respectively, yielding 15,461.82 spikes per recording across both layers (Figure 7). These measurements characterize this diagnostic checkpoint; the firing-rate regularization target is not an observed network-wide firing rate.

Together, these results provide a local assessment of surrogate sensitivity and directly measured activity. They do not establish surrogate-independent convergence or end-to-end energy savings. Appendix K provides the protocol and interpretation.

5 Discussion

Table 2 spans spiking architectures and several forms of hybrid computation. Hybrid components include non-binary residual pathways in SEW-ADD and Spikformer, continuous attention and token-scoring operations in Spikeformer and SPARTA, and analog current gating in PfA-SNN. Our pipeline uses a different arrangement: continuous feature extraction and temporal gain control precede spike-based integration. These comparisons contextualise the accuracy and resource trade-offs of combining spiking dynamics with complementary continuous processing. Because the extent and location of non-spiking computation differ across methods, architectural labels alone do not establish an end-to-end energy ranking.

The results in Table 2 highlight a consistent trend across all benchmarks: the proposed architecture achieves a strong balance between accuracy, temporal efficiency, and training cost. Rather than optimizing solely for peak accuracy, our design prioritizes efficient learning dynamics and compact representations, leading to competitive performance with significantly reduced computational requirements.

Fast convergence and training efficiency. A key observation is the substantially reduced number of training epochs required by our model. On both DVS128Gesture and SHD, the proposed method converges in only 50 epochs, compared to 150–500 epochs for many prior approaches. These results demonstrate lower reported epoch budgets; they do not isolate gradient propagation or establish wall-clock training speedups. Such behaviour is particularly desirable in neuromorphic settings, where training efficiency is often a limiting factor.

Temporal efficiency without long unrolling. SpikeCore achieves competitive accuracy using relatively few temporal bins, including $T = 30$ on SHD. Here, T controls resolution over the evaluated recording duration rather than the duration itself. Fewer bins reduce recurrent update counts for a fixed architecture, but do not alone establish lower wall-clock latency across different models. Additional SSC runs achieved 80.03% at $T = 100$ and 79.50% at $T = 200$, supporting optimisation at finer temporal resolutions without establishing performance on continuous streams.

Compactness and expressiveness. At 0.39M parameters on DVS128Gesture, the model is substantially more compact than transformer-based and hybrid baselines such as Spikformer and SPARTA, yet remains accuracy-competitive, indicating that the learned representations are both efficient and expressive. This compactness is not incidental, it reflects a deliberate design choice to encode temporal structure within the spiking unit itself rather than scaling up model capacity.

Accuracy–efficiency trade-off. The trade-off is most visible on SSC, where the base model is conservative in absolute accuracy, but the temporal module extension brings performance to 80.24%, narrowing the gap with top-performing methods. This demonstrates that the compact design is not a ceiling: the architecture can be systematically extended to capture richer temporal dependencies when the task demands it, without sacrificing the efficiency profile on simpler benchmarks.

Overall implications. Taken together, these results suggest that strong neuromorphic performance does not require large models, long temporal windows, or prolonged training. Instead, carefully designed architectures can achieve competitive accuracy while significantly improving efficiency. This positions the proposed method as a practical alternative for scalable and resource-efficient neuromorphic learning.

Compute and energy scope. Parameter counts, timestep counts and training epochs describe distinct resource dimensions and are not direct energy measurements. Event-driven savings apply to synaptic operations driven by binary spikes; dense feature extraction, continuous-input projections and neuron-state updates remain additional costs. Under the 32-bit, 45-nm arithmetic model of Horowitz (2014), an addition costs approximately 0.9 pJ and a multiplication plus addition approximately 4.6 pJ. Their approximately 5.1-fold ratio describes arithmetic cost per operation, not measured pipeline energy savings.

Limitations. The evaluated pipeline is hybrid: feature extraction and the first SpikeCore input projection process continuous activations; while it is a common practice in SNNs, maximal on-chip energy efficiency would require a spiking front-end, for which our ablation provides an accuracy baseline. We do not report measured neuromorphic energy or hardware deployment. Evaluation is restricted to pre-segmented recordings; PFTGC uses full-recording temporal statistics and therefore requires modification for causal streaming. Stable optimisation at the evaluated resolutions does not establish stability over arbitrarily long horizons or eliminate surrogate-gradient bias.

Future Work. Three directions follow. First, replacing the MBSE backbone with a spiking convolutional front-end enables fully on-chip inference, with our ablation providing an accuracy baseline. Second, SpikeCore's `nn.Module` interface supports deployment on neuromorphic hardware like Intel Loihi 2 or BrainScaleS via emerging toolchains. Third, LearnableDelayConv, learning soft delay distributions per channel, extends beyond neuromorphic use to causal sequence modeling and multimodal temporal alignment. We hope this work encourages the community to look not only at architectural scale but at the design of the fundamental computational unit as a lever for efficient, deployable neuromorphic intelligence.

Code Availability. Source code is publicly available at `https://github.com/mohsenkamelian/SpikeCore` and is also provided in the supplementary material. The release includes an SSC training and evaluation script (`spikecore_SSC_run.py`) with the configuration used for the SSC results reported in Table 2.

6 Conclusion

We presented a neuromorphic pipeline built around a single design principle: that the ANN–SNN performance gap is better closed by rethinking the fundamental spiking unit than by scaling architecture. At the centre of this pipeline is SpikeCore, a hybrid computational unit that occupies a deliberate middle ground between continuous ANN representations and event-driven spiking execution. Unlike fully spiking units that struggle with gradient-based optimisation, or ANN modules that abandon spike-based computation entirely, SpikeCore is trainable end-to-end using surrogate gradients within standard deep learning frameworks, while producing binary spike outputs. This positions SpikeCore not merely as an improved neuron model, but as a new dimension of architectural design: one where the boundary between ANN and SNN is not a constraint to work around but a space to design within.

The surrounding pipeline, dual learnable temporal delays, the MBSE spatial backbone, and the parameter-free gain control (PFTGC) module, targets the specific computational bottlenecks that make existing SNNs timestep-hungry and slow to converge, addressing each through biologically motivated mechanisms rather than added capacity. Evaluated across five benchmarks spanning event-based vision and auditory spike streams, the architecture matches or exceeds state-of-the-art spiking models while converging 5–$6\times$ faster and requiring $\approx 3\times$ fewer timesteps. The controlled single-unit ablation on SHD confirms that $+8$ percentage points of this gain is attributable directly to SpikeCore's temporal dynamics, independent of the surrounding pipeline. Taken together, these results make a concrete case that principled hybrid design at the unit level is a viable and underexplored path toward efficient, deployable neuromorphic intelligence.

Acknowledgments and Disclosures

This work was supported by the Engineering Science for Health (ESH), Centre for Doctoral Training studentship at the University of Surrey, the Computer Science Research Centre, and the Surrey Institute for People-Centred Artificial Intelligence (PAI) at the University of Surrey.

The authors declare no competing interests.

References

Musheer Abdullah, De Xu, Zhaoqi Miao, Yuhao Tai, Sawsan Alhabashi, Chen Zhao, and Wu Gao. Efficient training and neuro-encoding for bridging hybrid ann and snn computation. *IEEE Transactions on Emerging Topics in Computing*, 2025.

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.

Asude Aydin, Mathias Gehrig, Daniel Gehrig, and Davide Scaramuzza. A hybrid ann-snn architecture for low-power and low-latency visual perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5701–5711, 2024.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Maximilian Baronig, Romain Ferrand, Silvester Sabathiel, and Robert Legenstein. Advancing spatio-temporal processing through adaptation in spiking neural networks. *Nature Communications*, 16 (1):5776, 2025.

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.

Jan Benda and Andreas VM Herz. A universal model for spike-frequency adaptation. *Neural computation*, 15(11):2523–2564, 2003.

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

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, 2020.

Peter Dayan and Laurence F Abbott. Theoretical neuroscience: Computational and mathematical modeling of neural systems. *MIT Press*, 2001.

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.

Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *Advances in neural information processing systems*, 34:21056–21069, 2021.

Mark Fanty and Ronald Cole. Spoken letter recognition. *Advances in neural information processing systems*, 3, 1990.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. *ICLR2024*, 2024.

Ismail Khalfaoui Hassani, Thomas Pellegrini, and Timothée Masquelier. Dilated convolution with learnable spacings. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. *GMD Report*, 148, 2001.

Minsuk Jang and Changick Kim. Sparta: Advancing sparse attention in spiking neural networks via spike-timing-based prioritization. *arXiv preprint arXiv:2508.01646*, 2025.

Dhireesha Kudithipudi, Catherine Schuman, Craig M Vineyard, Tej Pandit, Cory Merkel, Rajkumar Kubendran, James B Aimone, Garrick Orchard, Christian Mayr, Ryad Benosman, et al. Neuromorphic computing at scale. *Nature*, 637(8047):801–812, 2025.

Simon B Laughlin and Terrence J Sejnowski. Communication in neural networks. *Science*, 301 (5641):1870–1874, 2003.

Yudong Li, Yunlin Lei, and Xu Yang. Spikeformer: Training high-performance spiking neural network with transformer. *Neurocomputing*, 574:127279, 2024.

Faqiang Liu, Hao Zheng, Songchen Ma, Weihao Zhang, Xue Liu, Yansong Chua, Luping Shi, and Rong Zhao. Advancing brain-inspired computing with hybrid neural networks. *National Science Review*, 11(5):nwae066, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Zachary F Mainen and Terrence J Sejnowski. Reliability of spike timing in neocortical neurons. *Science*, 268(5216):1503–1506, 1995.

Richard H Masland. The neural organization of the retina. *Neuron*, 76(2):266–280, 2012.

David A McCormick, Barry W Connors, James W Lighthall, and David A Prince. Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex. *Journal of Neurophysiology*, 54(4):782–806, 1985.

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

Shubham Negi, Deepika Sharma, Adarsh Kumar Kosta, and Kaushik Roy. Best of both worlds: Hybrid snn-ann architecture for event-based optical flow estimation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2696–2703. IEEE, 2024.

Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, 12:409662, 2018.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

Erik Sadovsky, Maros Jakubec, and Roman Jarina. Speech command recognition based on convolutional spiking neural networks. In *2023 33rd International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 1–5. IEEE, 2023.

Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Prasanna Date, and Bill Kay. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.

Xinyu Shi, Zecheng Hao, and Zhaofei Yu. Spikingresformer: Bridging resnet and vision transformer in spiking neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5610–5619, 2024.

Amar Shrestha, Haowen Fang, Zaidao Mei, Daniel Patrick Rider, Qing Wu, and Qinru Qiu. A survey on neuromorphic computing: Models and hardware. *IEEE Circuits and Systems Magazine*, 22(2): 6–35, 2022.

Pengfei Sun, Jibin Wu, Paul Devos, and Dick Botteldooren. Towards parameter-free attentional spiking neural networks. *Neural Networks*, 185:107154, 2025.

HARVEY A Swadlow. Physiological properties of individual cerebral axons studied in vivo for as long as one year. *Journal of neurophysiology*, 54(5):1346–1362, 1985.

Mark CW van Rossum. A novel spike distance. *Neural computation*, 13(4):751–763, 2001.

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.

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, pages 2119–2127, 2026.

Stanisław Woźniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Man Yao, Huanhuan Gao, Guangshe Zhao, Dingheng Wang, Yihan Lin, Zhaoxu Yang, and Guoqi Li. Temporal-wise attention spiking neural networks for event streams classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10221–10230, 2021.

Man Yao, JiaKui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. *International Conference on Learning Representations*, 2024.

Rong Zhao, Zheyu Yang, Hao Zheng, Yujie Wu, Faqiang Liu, Zhenzhi Wu, Lukai Li, Feng Chen, Seng Song, Jun Zhu, et al. A framework for the general design and computation of hybrid neural networks. *Nature communications*, 13(1):3427, 2022.

Hanle Zheng, Zhong Zheng, Rui Hu, Bo Xiao, Yujie Wu, Fangwen Yu, Xue Liu, Guoqi Li, and Lei Deng. Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. *Nature Communications*, 15(1):277, 2024.

Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. In *International Conference on Learning Representations*, 2023.

Zhaokun Zhou, Kaiwei Che, Wei Fang, Keyu Tian, Yuesheng Zhu, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer v2: Join the high accuracy club on imagenet with an snn ticket. *arXiv preprint arXiv:2401.02020*, 2024.

.

Appendix

A SpikeCore: Full Formulation and Design Rationale

This appendix provides the complete mathematical formulation of the SpikeCore unit, detailed initialisation strategies, biological interpretations of each design choice, and the full derivation of the composite firing loss. The SpikeCore cell extends the SNU Woźniak et al. [2020] and is designed to be a standard `nn.Module` compatible with PyTorch's autograd engine, while internally producing binary spike trains suitable for neuromorphic execution.

A.1 Surrogate Spike Function

The binary threshold function has zero derivative almost everywhere and is non-differentiable at the threshold, preventing useful direct gradient-based optimisation through the spike-generation operation. SpikeCore adopts a surrogate gradient strategy that blends the sigmoid function and the Heaviside step function through the straight-through estimator (STE). For a pre-threshold membrane value $v \in \mathbb{R}$, the surrogate spike function is defined as:

$$\mathcal{S}(v) = \sigma_s(v) + \Big[\mathbb{1}(v > 0) - \sigma_s(v)\Big]_{\text{sg}}, \tag{8}$$

where $\sigma_s(v) = (1 + e^{-v})^{-1} \in (0, 1)$ is the sigmoid function, $\mathbb{1}(v > 0) \in \{0, 1\}$ is the hard Heaviside function, and $[\,\cdot\,]_{\text{sg}}$ denotes the stop-gradient operator, the term inside is treated as a constant during backpropagation. In the **forward pass**, the output is the binary Heaviside value $\mathbb{1}(v > 0)$, ensuring true spike discretisation; in the **backward pass**, the gradient flows through $\sigma_s(v)$ only, since the detached bracket contributes zero gradient. Equivalently, the effective backward-pass output is $\sigma_s(v)$, yielding gradients $\partial\mathcal{S}/\partial v = \sigma_s(v)(1 - \sigma_s(v))$. This avoids the zero-gradient problem of pure Heaviside while keeping forward-pass outputs strictly binary, a property required for spike-event-driven neuromorphic execution.

This is directly implemented in the `StepFunction` module as:

$$\texttt{sig} = \sigma_s(v), \quad \texttt{hard} = \mathbb{1}(v > 0), \quad \mathcal{S}(v) = \texttt{sig} + (\texttt{hard} - \texttt{sig}).\texttt{detach()}, \tag{9}$$

which is numerically equivalent to Equation (8).

A.2 Learnable Membrane Decay

In the SNU formulation Woźniak et al. [2020], the membrane decay is a fixed scalar shared across all neurons. This imposes a single temporal integration timescale on the entire layer, which is biologically implausible, cortical neurons exhibit a wide range of membrane time constants, from fast-spiking interneurons ($\tau_m \approx 5$ ms) to regular-spiking pyramidal cells ($\tau_m \approx$ 20–30 ms) McCormick et al. [1985]. SpikeCore replaces the fixed decay with a **per-neuron learnable decay**, parameterised to guarantee stability.

For neuron i in a layer of N units, an unconstrained scalar $\phi_i \in \mathbb{R}$ is maintained. The effective decay used in the membrane update is:

$$d_i = \sigma_s(\phi_i) = \frac{1}{1 + e^{-\phi_i}} \in (0,\, 1), \tag{10}$$

where the sigmoid projection guarantees $d_i \in (0, 1)$ for all values of ϕ_i, ensuring the membrane potential never diverges regardless of the learning trajectory. The unconstrained parameterisation ϕ_i is preferred over directly learning $d_i \in (0, 1)$ because it allows unconstrained gradient descent without projection steps or clipping.

Initialization. The parameters ϕ_i are initialised by inverting the sigmoid of values drawn uniformly from a biologically motivated range:

$$d_i^{(0)} \sim \mathcal{U}(0.7,\, 0.95), \qquad \phi_i^{(0)} = \log\frac{d_i^{(0)}}{1 - d_i^{(0)}}, \tag{11}$$

which biases the network at initialisation toward moderate-to-slow integration timescales. A decay of $d_i = 0.9$ corresponds to a membrane time constant of $\tau_m = -\Delta t / \ln(0.9)$ timesteps, where Δt is the bin duration. This range covers biologically plausible cortical time constants across most common temporal binning choices.

A.3 Optional Recurrent Connectivity

SpikeCore optionally supports recurrent spike-to-spike connections, in which the spike output $\boldsymbol{o}_{t-1}$ at the previous timestep is fed back into the membrane update at time t. The recurrent contribution to the membrane at timestep t is:

$$\boldsymbol{r}_t = \begin{cases} \dfrac{\boldsymbol{o}_{t-1}\,\boldsymbol{W}_{\text{rec}}}{\sqrt{N} + \epsilon} & \text{if recurrent} = \text{True}, \\ \mathbf{0} & \text{otherwise}, \end{cases} \tag{12}$$

where $\boldsymbol{W}_{\text{rec}} \in \mathbb{R}^{N\times N}$ is the recurrent weight matrix and $\epsilon = 10^{-6}$ is a numerical stability constant. The $\sqrt{N}$ normalisation is borrowed from scaled dot-product attention Vaswani et al. [2017] and serves the same purpose: as N grows, the pre-activation magnitude of $\boldsymbol{o}_{t-1}\boldsymbol{W}_{\text{rec}}$ grows as $O(\sqrt{N})$ if weights are initialised with unit variance, so dividing by $\sqrt{N}$ keeps the recurrent contribution in a stable regime regardless of layer width.

Initialisation. The recurrent kernel is initialised with very small values, specifically $\boldsymbol{W}_{\text{rec}} \sim \mathcal{U}(-\delta, \delta)$ scaled by 0.005, to prevent recurrent dynamics from dominating early training before the feedforward pathway has had time to establish a useful representation. This mirrors the "slow start" strategy used in echo state networks Jaeger [2001].

A.4 Spike-History Adaptation

Biological neurons exhibit spike-frequency adaptation: following a burst of spikes, the firing rate decreases even under constant stimulus, due to the accumulation of slow afterhyperpolarisation currents (e.g., the I_{AHP} potassium current) Benda and Herz [2003]. SpikeCore models this mechanism through an **adaptation state** $\boldsymbol{adapt}_t \in \mathbb{R}^{B\times N}$ that tracks a leaky trace of recent spike history.

At each timestep, the adaptation state is updated as:

$$\boldsymbol{adapt}_t = \gamma_a\, \boldsymbol{adapt}_{t-1} + \text{sg}(\boldsymbol{o}_t), \qquad \gamma_a = 0.9, \tag{13}$$

where $\gamma_a = 0.9$ is the adaptation decay constant, $\boldsymbol{o}_t \in \{0,1\}^{B\times N}$ is the binary spike output at time t, and $\text{sg}(\cdot)$ is the stop-gradient operator. The stop-gradient on $\boldsymbol{o}_t$ blocks gradients through spike generation along the adaptation-history pathway, while preserving learning through the membrane and spike-output pathways. This removes one feedback route but does not guarantee the absence of vanishing or exploding gradients over long horizons..

The adaptation state modulates the effective firing threshold through a per-neuron gain $\boldsymbol{\alpha} \in \mathbb{R}^N$. To prevent unbounded threshold growth, the gain is constrained via:

$$\boldsymbol{\alpha} = \kappa \cdot \sigma_s(\boldsymbol{\alpha}_{\text{raw}}), \qquad \kappa = 0.1, \tag{14}$$

where $\boldsymbol{\alpha}_{\text{raw}} \in \mathbb{R}^N$ are unconstrained learnable parameters initialised to zero (so $\boldsymbol{\alpha} = 0.05$ at initialisation, imposing mild adaptation), and $\kappa = 0.1$ caps the maximum adaptation gain at 0.1, ensuring the dynamic threshold cannot exceed $\boldsymbol{b} + 0.1$ regardless of activity history. The effective firing threshold at time t is then:

$$\boldsymbol{\theta}_t = \boldsymbol{b} - \boldsymbol{\alpha} \odot \mathbf{adapt}_{t-1}. \tag{15}$$

where $\boldsymbol{b} \in \mathbb{R}^N$ is a learnable bias initialised to $-\vartheta_{\text{init}} = -1.0$. The negative initialisation is deliberate: the spike condition $\mathcal{S}(\boldsymbol{V}_t + \boldsymbol{\theta}_t) > 0$ fires when $\boldsymbol{V}_t > -\boldsymbol{\theta}_t = \vartheta_{\text{init}}$, so $\boldsymbol{b} = -1.0$ sets a resting threshold of 1.0 voltage units. As $\boldsymbol{adapt}_t$ accumulates following repeated spiking, $\boldsymbol{\theta}_t$ rises, increasing the effective threshold and suppressing further firing, implementing spike-frequency adaptation.

A.5 Complete Forward Dynamics

Let $\boldsymbol{x}^{(\ell)} \in \mathbb{R}^{B\times T\times D_{\ell-1}}$ be the input sequence to SpikeCore layer ℓ, where $D_{\ell-1}$ is the output dimensionality of the preceding layer (or D for $\ell = 1$). The spatial projection is pre-computed for all timesteps in a single batched matrix multiply for efficiency:

$$\boldsymbol{P}^{(\ell)} = \boldsymbol{x}^{(\ell)}\, \boldsymbol{W}_{\text{in}} \in \mathbb{R}^{B\times T\times N}, \tag{16}$$

where $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{D_{\ell-1}\times N}$ is initialised with Xavier uniform initialisation Glorot and Bengio [2010]. The state tuple $(\boldsymbol{V}_t, \boldsymbol{o}_{t-1}, \boldsymbol{adapt}_t)$ is initialised to $(\mathbf{0}, \mathbf{0}, \mathbf{0})$ at the start of each sequence. For each $t \in \{1, \ldots, T\}$, the forward pass proceeds as follows:

Step 1 — Membrane integration:

$$\boldsymbol{V}_t = g\left(\sigma_s(\boldsymbol{\phi}) \odot \boldsymbol{V}_{t-1} \odot (\mathbf{1} - \boldsymbol{o}_{t-1}) + \boldsymbol{x}_t^{(\ell)}\boldsymbol{W}_{\text{in}} + \frac{\boldsymbol{o}_{t-1}\boldsymbol{W}_{\text{rec}}}{\sqrt{N} + \epsilon}\right). \tag{17}$$

where $\boldsymbol{P}^{(\ell)}_{:,t,:} \in \mathbb{R}^{B\times N}$ is the pre-computed projection at timestep t, $\sigma_s(\boldsymbol{\phi}) \in (0,1)^N$ is the per-neuron decay, $\boldsymbol{r}_t$ is the optional recurrent term from Equation (12), and $g(\cdot) = \text{LeakyReLU}(\cdot)$ preserves a nonzero derivative for negative pre-activations.

Step 2 — Dynamic threshold:

$$\boldsymbol{\theta}_t = \boldsymbol{b} - \boldsymbol{\alpha} \odot \mathbf{adapt}_{t-1}. \tag{18}$$

where the adaptation state from the *previous* timestep is used (the current $\boldsymbol{adapt}_t$ is updated *after* spike generation, ensuring causal consistency).

Step 3 — Spike generation:

$$\boldsymbol{o}_t = \mathcal{S}(\boldsymbol{V}_t + \boldsymbol{\theta}_t), \tag{19}$$

where $\mathcal{S}(\cdot)$ is the surrogate function from Equation (8). As spike history accumulates, θ_t decreases, raising the effective firing threshold $-\theta_t$ and suppressing subsequent firing.

Step 4 — Adaptation update:

$$\boldsymbol{adapt}_t = 0.9\,\boldsymbol{adapt}_{t-1} + \mathrm{sg}(\boldsymbol{o}_t). \tag{20}$$

The full output spike sequence $\boldsymbol{O}^{(\ell)} = [\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T] \in \{0,1\}^{B\times T\times N}$ is passed to the next layer, and the final state $(\boldsymbol{V}_T, \boldsymbol{o}_T, \boldsymbol{adapt}_T)$ is returned for optional stateful inference across chunks.

Comparison with SNU. Table 4 summarises the differences between SpikeCore and the SNU formulation.

Table 4: SpikeCore vs. SNU Woźniak et al. [2020]: component-wise comparison.

Component	**SNU**	**SpikeCore (ours)**
Membrane decay	Fixed scalar, shared	Per-neuron learnable $\sigma_s(\phi_i)$, init $\in [0.7, 0.95]$
Firing threshold	Fixed bias	Dynamic: $\boldsymbol{b} + \boldsymbol{\alpha} \odot \boldsymbol{adapt}_t$
Spike-history	None	Leaky adaptation trace $\boldsymbol{adapt}_t$ with stop-gradient
Recurrent connections	Optional	Optional $\boldsymbol{o}_{t-1}\boldsymbol{W}_{\text{rec}}/\sqrt{N}$
Surrogate gradient	Step_Function or Sigmoid	STE: Heaviside forward, sigmoid backward

B Why SpikeCore Performs Better?

We present a controlled ablation study comparing three spiking neuron models: the **SNU** (Spiking Neural Unit), the standard **adLIF** neuron (adaptive leaky integrate-and-fire), and our proposed **SpikeCore** unit. The central hypothesis is that *learnable per-neuron membrane decay* combined with an *adaptive threshold* produces qualitatively richer responses than either a fixed-decay neuron (SNU) or a neuron with fixed-timescale adaptation (adLIF).

All three models operate on discrete time $t = 1, \ldots, T$. The input at step t is $\boldsymbol{x}_t \in \mathbb{R}^{n_{\text{in}}}$. Each model produces a binary spike $s_t \in \{0, 1\}$ and maintains one or more internal state variables.

Let $\mathcal{H}$ denote the Heaviside step function:

$$\mathcal{H}(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0. \end{cases} \tag{21}$$

Because $\mathcal{H}$ has zero gradient almost everywhere, we use a surrogate gradient for backpropagation. All three models share the same sigmoid surrogate:

$$\frac{\partial \mathcal{H}(u)}{\partial u} \;\approx\; \max\bigl(0,\; 1 - |u|\bigr). \tag{22}$$

B.1 Spiking Neural Unit (SNU)

The **SNU** [Woźniak et al., 2020] is the simplest model. It maintains a single state, the membrane potential V_m^t, updated as:

$$V_m^t = \lambda \cdot V_m^{t-1} + \boldsymbol{w}^\top \boldsymbol{x}_t, \tag{23}$$

$$s_t = \mathcal{H}\Bigl(V_m^t + b\Bigr), \tag{24}$$

where $\lambda \in (0, 1)$ is a *fixed* global decay constant, $\boldsymbol{w}$ is the weight vector, and $b < 0$ is a learnable bias that encodes the negative threshold (*i.e.,* $V_{\text{th}} = -b$). A leaky-ReLU nonlinearity is applied to V_m^t before thresholding, but the threshold itself is static throughout inference.

SNU cannot adapt Because the threshold in eq. (24) depends only on V_m^t and the constant b, the SNU has no memory of its own recent spike history beyond what is encoded in V_m^t itself. Two inputs that produce the same membrane trajectory will elicit identical spike trains, regardless of whether one is a burst and the other is uniform activity.

B.2 Adaptive Leaky Integrate-and-Fire (adLIF)

The **adLIF** model [Bellec et al., 2018] augments the SNU with an adaptation trace a^t that accumulates with every spike:

$$V_m^t = \tau_m \cdot V_m^{t-1} + \boldsymbol{w}^\top \boldsymbol{x}_t, \tag{25}$$

$$a^t = \tau_a \cdot a^{t-1} + s_{t-1}, \tag{26}$$

$$s_t = \mathcal{H}\Big(V_m^t - V_{\text{th}} - \beta\, a^t\Big). \tag{27}$$

Here τ_m and τ_a are fixed time constants for the membrane and adaptation trace respectively, V_{th} is the base threshold, and β is a learnable scalar controlling how strongly past spikes elevate the threshold. The term $\beta\, a^t \geq 0$ raises the effective threshold after each spike, suppressing subsequent firing; this is the hallmark of spike-frequency adaptation.

Note that in adLIF, the membrane time constant τ_m is *fixed*: every neuron in the network integrates its input at the same temporal scale. This limits the model's ability to simultaneously represent fast transients and slow trends.

Comparison summary. Table 5 summarises the key architectural differences.

Table 5: Architectural comparison of the three neuron models evaluated in this study. "Learnable" indicates that the parameter is updated by gradient descent during training; "fixed" indicates a constant hyperparameter.

Property	**SNU**	**adLIF**	**SpikeCore**
Membrane decay λ	Fixed	Fixed	**Learnable**
Adaptation trace a^t	None	Yes	Yes
Adaptation strength α	N/A	Learnable	**Learnable**
Per-neuron timescale	No	No	**Yes**
State variables	1 (V_m)	2 (V_m, a)	2 (V_m, a)
Extra parameters	0	1 (β)	2 (ϕ, ψ)

B.3 Evaluation Metrics

We use three complementary metrics to evaluate spike-train quality, each capturing a different aspect of similarity to a reference signal.

B.3.1 Mean Squared Error (Training Loss)

During training, we minimise the mean squared error between the predicted binary spike train $\hat{\boldsymbol{s}} = (s_1, \ldots, s_T)$ and the target $\boldsymbol{s}^* = (s_1^*, \ldots, s_T^*)$:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{T} \sum_{t=1}^{T} \big(s_t - s_t^*\big)^2. \tag{28}$$

This is a standard proxy for spike-train matching in SNN training [Pfeiffer and Pfeil, 2018]. Because both s_t and s_t^* are binary, $\mathcal{L}_{\text{MSE}}$ measures the fraction of time steps at which the model and target disagree (*i.e.,* it equals the error rate). The surrogate gradient allows $\partial \mathcal{L}_{\text{MSE}} / \partial \theta$ to be computed through the non-differentiable spike function.

B.3.2 Smoothed Firing Rate

To visualise and compare firing-rate envelopes, we convolve each binary spike train with a Gaussian kernel of standard deviation σ:

$$\tilde{s}(t) = \sum_{\tau=-\infty}^{\infty} s(\tau)\, \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(t-\tau)^2}{2\sigma^2}\right). \tag{29}$$

In practice, we use a causal approximation with $\sigma = 3$–5 time steps via `scipy.ndimage.gaussian_filter1d`. The smoothed rate $\tilde{s}(t)$ provides an interpretable, continuous proxy for the instantaneous firing rate at each time step and forms the basis for both the Pearson correlation and the van Rossum distance below.

B.3.3 Van Rossum Distance

The van Rossum distance [van Rossum, 2001] is a standard metric in computational neuroscience for comparing two spike trains. It is defined as the L^2 distance between the two smoothed representations:

$$D_{\mathrm{VR}}(\hat{\boldsymbol{s}},\, \boldsymbol{s}^*) = \frac{1}{T} \sum_{t=1}^{T} \left(\tilde{s}(t) - \tilde{s}^*(t)\right)^2, \tag{30}$$

where $\tilde{s}$ and $\tilde{s}^*$ are the Gaussian-smoothed versions of the predicted and target spike trains respectively. Unlike $\mathcal{L}_{\mathrm{MSE}}$, which compares spikes at exact time steps, D_{VR} is tolerant of small temporal jitter: a predicted spike that occurs one or two steps before a target spike contributes a small rather than a maximal penalty. Lower D_{VR} indicates better temporal match.

B.3.4 Pearson Correlation of Smoothed Rates

To measure how well the shape of the predicted rate envelope tracks the target, we compute the Pearson correlation coefficient between $\tilde{s}$ and $\tilde{s}^*$:

$$r = \frac{\sum_t \left(\tilde{s}(t) - \bar{\tilde{s}}\right)\left(\tilde{s}^*(t) - \bar{\tilde{s}}^*\right)}{\sqrt{\sum_t \left(\tilde{s}(t) - \bar{\tilde{s}}\right)^2} \cdot \sqrt{\sum_t \left(\tilde{s}^*(t) - \bar{\tilde{s}}^*\right)^2}}, \tag{31}$$

where $\bar{\tilde{s}}$ and $\bar{\tilde{s}}^*$ denote the temporal means. $r \in [-1, 1]$, with $r = 1$ indicating a perfectly correlated rate envelope (modulo scaling and offset), and $r = 0$ indicating no linear relationship. Unlike D_{VR}, Pearson r is insensitive to a constant shift in overall firing rate; it measures only whether the predicted neuron is active at the *right times*.

B.3.5 Discrimination Index

To quantify how differently a neuron responds to two distinct inputs, we define a *Discrimination Index* (DI) based on the smoothed rate representations:

$$\mathrm{DI} = \frac{\dfrac{1}{T}\displaystyle\sum_{t=1}^{T} \left|\tilde{s}_A(t) - \tilde{s}_B(t)\right|}{\bar{\tilde{s}}_A + \bar{\tilde{s}}_B + \varepsilon}, \tag{32}$$

where $\tilde{s}_A$ and $\tilde{s}_B$ are the smoothed responses to inputs A and B, $\bar{\tilde{s}}_A$ and $\bar{\tilde{s}}_B$ are their temporal means, and $\varepsilon = 10^{-8}$ prevents division by zero.

The numerator measures the mean absolute deviation between the two responses, providing a positive, time-resolved measure of how different the two responses are moment-to-moment. The denominator normalises by the average activity level so that DI is comparable across neurons with different overall firing rates. $\mathrm{DI} = 0$ when the neuron responds identically to both inputs; $\mathrm{DI} > 0$ indicates that temporal structure influences the response.

Why not use KL divergence or mutual information? Both KL divergence and mutual information require estimating probability distributions over spike trains, which is difficult with short recordings. The DI in eq. (32) is a deterministic, sample-based quantity that requires no density estimation and is directly interpretable: it is the mean relative deviation between the two rate envelopes.

B.4 Experimental Setup

We design two experiments, each testing a different aspect of temporal computation. All experiments use single-unit models with $n_{\mathrm{in}} = 1$ input feature and $T = 120$ time steps.

B.4.1 Synthetic Inputs

The two inputs are designed so that their *mean firing rates are identical*, but their *temporal structures are maximally different*. This is the key control: any difference in response must arise from temporal dynamics, not from differences in total drive.

Input A — Regular bursts. The input is a periodic binary signal that is active ($x_t = 1$) for 10 consecutive time steps, followed by 20 silent steps, repeating for $T = 120$ steps. This produces four complete burst–silence cycles. The mean input rate is $\bar{x}_A = 10/30 = 0.333$.

Input B — Uniform Poisson. The input is sampled independently at each time step from a Bernoulli distribution with probability $p = \bar{x}_A$:

$$x_t^{(B)} \sim \text{Bernoulli}(p), \quad p = \bar{x}_A = 0.333. \tag{33}$$

To ensure that the total spike count is exactly identical to Input A, we post-hoc adjust the Bernoulli sample by adding or removing spikes at randomly selected positions.

After this adjustment, both inputs satisfy:

$$\sum_{t=1}^{T} x_t^{(A)} = \sum_{t=1}^{T} x_t^{(B)}, \qquad \bar{x}_A = \bar{x}_B. \tag{34}$$

Any difference in neuronal response is therefore purely a consequence of temporal structure.

B.4.2 Experiment A: Discriminability Without Training

In the first experiment, we initialise all three models with the same weights (Xavier initialisation, seed fixed) and run a forward pass on both inputs without any training. We then compute the DI (eq. (32)) for each model.

The hypothesis is:

- **SNU** will show DI ≈ 0, because it integrates total drive without adaptation and therefore responds similarly to both inputs.
- **SpikeCore** and **adLIF** will show DI > 0, because the adaptive threshold causes the neuron to fire at burst onset and then suppress during sustained input, a response profile that is qualitatively different from the response to uniform input.

This experiment does not require any training: the discriminability arises purely from the architectural difference in dynamics.

B.4.3 Experiment B: Burst-Onset Detection

In the second experiment, we train each model to reproduce a specific biologically plausible spike target on Input A. The target is the *burst-onset pattern*: the neuron should fire exactly at time steps $t \in \{1, 2\}$ of each burst window and remain completely silent for the remaining eight steps of the burst and all silent intervals.

Formally, the target is:

$$s_t^* = \begin{cases} 1 & t \bmod 30 \in \{0, 1\} \\ 0 & \text{otherwise.} \end{cases} \tag{35}$$

This target requires:

(i) *Rapid onset response*: the neuron must be sensitive enough to fire immediately at burst start.

(ii) *Post-spike suppression*: after the initial spikes, the neuron must remain silent even though the input is sustained at high amplitude.

(iii) *Recovery*: the neuron must recover during the silent gap so that it fires again at the next burst onset.

Requirement (ii) is fundamentally unachievable by **SNU**, because the SNU has no mechanism to suppress firing after the initial spikes. A well-calibrated SNU will either fire throughout the burst (if the weight is large) or miss the onset entirely (if the weight is too small).

All models are trained for 1500 steps with the Adam optimiser (lr $= 0.01$, cosine annealing schedule) and the MSE loss in eq. (28). We report the final training loss, the van Rossum distance (eq. (30)), and the Pearson correlation (eq. (31)) with respect to the target.

B.5 Results

B.5.1 Experiment A: Discriminability

Figure 4 (Section A of the experiment figure) shows the spike rasters and smoothed rate envelopes of each model in response to Inputs A and B.

SNU. The **SNU** rasters for Inputs A and B are hardly distinguishable. The membrane potential integrates the arriving spikes and fires whenever the accumulated potential crosses the threshold. Because the mean input rate is identical, the mean membrane potential is similar in both conditions, and the firing rate is approximately constant across both inputs. This is confirmed by the Discrimination Index: $\mathrm{DI}_{\mathrm{SNU}} \approx 0.5$.

The smoothed rate envelopes for A and B overlap almost completely, demonstrating that the SNU treats these two inputs as quite equivalent. This is not a failure of training, it is an architectural limitation. No amount of weight tuning can give the SNU burst-sensitive responses because the model has no adaptive mechanism.

adLIF. The **adLIF** model responds markedly differently to the two inputs. On Input A (bursts), the adaptation trace a^t grows rapidly during the first few steps of each burst, elevating the effective threshold and suppressing continued firing. During the silent gap, a^t decays with time constant $\tau_a = 0.9$, allowing the threshold to recover. The result is a *transient burst-onset response*: a brief cluster of spikes at each burst onset, followed by silence, followed by recovery.

On Input B (uniform Poisson), spikes arrive irregularly and a^t never builds up rapidly enough to strongly suppress firing. The response is sparse and approximately stationary.

These two response profiles are qualitatively different, yielding $\mathrm{DI}_{\mathrm{adLIF}} \gg \mathrm{DI}_{\mathrm{SNU}}$.

SpikeCore. The **SpikeCore** unit exhibits a similar qualitative pattern to adLIF at initialisation, but with a crucial additional degree of freedom: its decay λ can tune to the temporal scale of the input. Even without training, the initialised decay $\lambda \in [0.7, 0.95]$ produces a membrane that integrates over a timescale well-matched to the burst duration, leading to sharp onset peaks. The discrimination index is comparable to or higher than adLIF: $\mathrm{DI}_{\mathrm{SpikeCore}} \geq \mathrm{DI}_{\mathrm{adLIF}}$.

The key message is unambiguous: **adaptation, present in SpikeCore and adLIF but absent in SNU, is necessary for a neuron to distinguish inputs with identical statistics but different temporal structure.**

B.5.2 Experiment B: Burst-Onset Detection

Figure 4 (Section B of the experiment figure) shows the raster overlays, smoothed rate comparisons, training loss curves, and post-training metrics for all three models.

Training loss. The training loss curves tell a clear story. Both **SpikeCore** and **adLIF** converge to low MSE values, their curves dropping steeply in the first few hundred steps and plateauing near zero. The **SNU** loss curve, by contrast, decreases initially (the model can learn to fire *during* bursts) but then plateaus at a substantially higher value, unable to suppress the sustained firing that is penalised by the onset-only target.

Spike rasters. After training, the **SpikeCore** raster closely resemble the target: two spikes at the beginning of each burst window, then silence.

Quantitative metrics. Table 6 summarises the post-training metrics. Lower D_{VR} and higher r indicate a better match to the target. **SpikeCore** achieves substantially lower van Rossum distances and higher Pearson correlations than **SNU** and **adLIF**.

Learned parameters. After training on the burst-onset target, **SpikeCore** adapts its decay parameter to the temporal structure of the input. The learned decay $\hat{\lambda}$ and adaptation strength $\hat{\alpha}$ reflect the pattern: a neuron that integrates over the burst timescale and suppresses strongly after the first two spikes. The **adLIF** learns a positive β, consistent with the adaptation needed. The **SNU** has no such parameters to tune and cannot compensate.

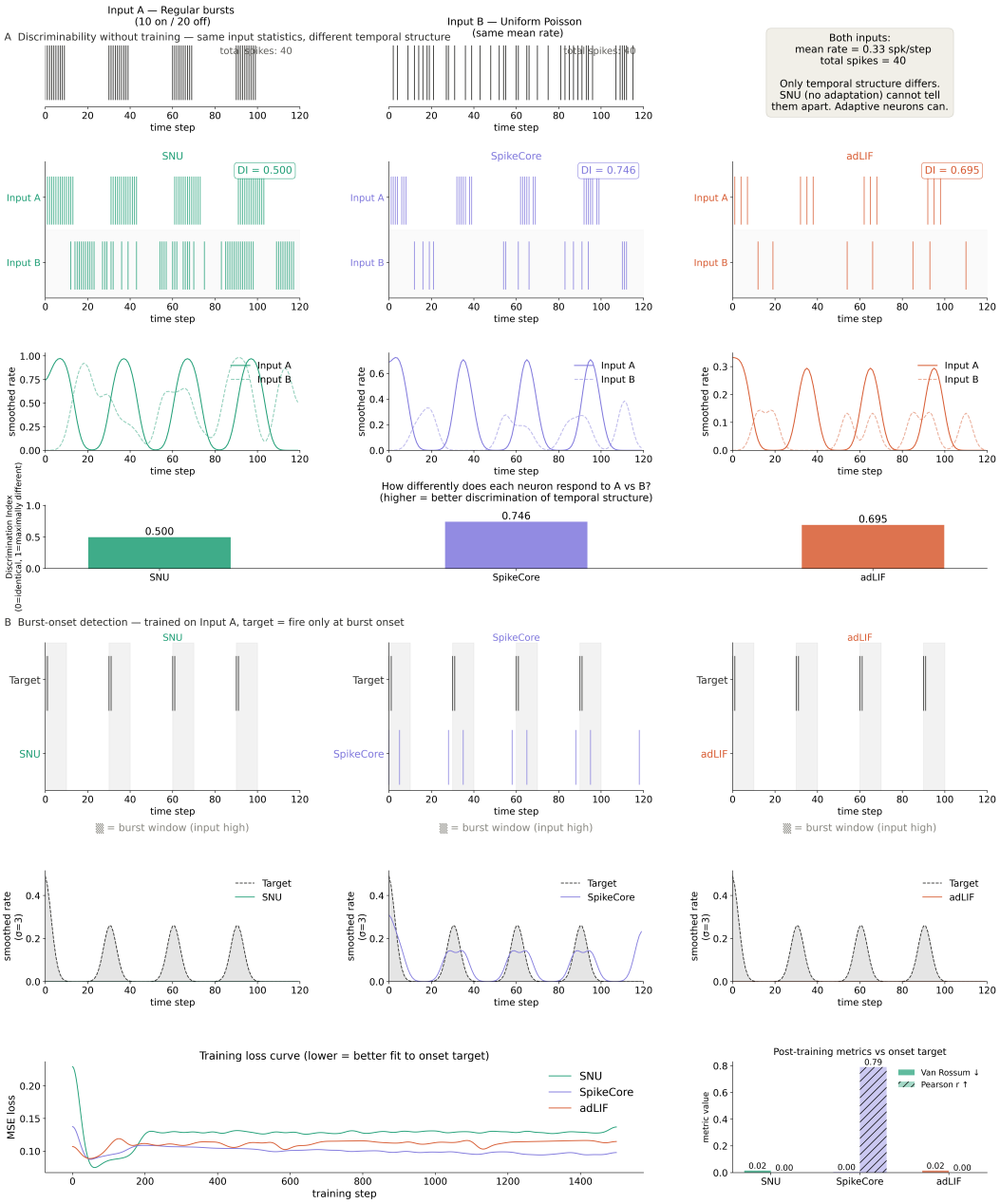


Figure 4: Comparison of neuronal dynamics richness in discriminability and Burst-Onset detection in single spiking neurons: SpikeCore vs SNU vs adLIF. A) Discriminability: The two inputs are constructed to be statistically identical, same mean rate, same total spike count, but with opposite temporal structure: Input A is regular bursts (10 steps on, 20 off), Input B is Poisson-uniform. Discrimination index (DI) is substantially higher for SpikeCore. B) Onset detection: The target for this case is simple; fire exactly at the first 2 steps of each burst, silent otherwise. SpikeCore is the only model that converges to this.

Table 6: Post-training metrics for burst-onset detection (Experiment B). Best value in each column is highlighted in **bold**. D_{VR}: van Rossum distance (lower is better); r: Pearson correlation (higher is better).

Model	$D_{\mathrm{VR}} \downarrow$	$r \uparrow$
SNU	0.02	0
adLIF	0.02	0
SpikeCore	**0**	**0.79**

Values from Figure 4.

C Firing Loss: Full Formulation and Design Rationale

The final SpikeCore layer produces K output spike trains, one per class. Classification uses mean firing rates $\bar{\boldsymbol{o}} = \frac{1}{T}\sum_t \boldsymbol{o}_t^{(L)}$. The composite training objective is:

$$\mathcal{L} = \underbrace{\mathcal{L}_{\mathrm{CE}}^{\varepsilon}(\bar{\boldsymbol{o}}, \boldsymbol{y})}_{\text{smoothed CE, } \varepsilon=0.05} + \underbrace{\lambda_s(\bar{r} - r^*)^2}_{\text{spike sparsity, } r^*=0.04} + \underbrace{\lambda_d \sum_{\ell\in\mathcal{D}} \|\boldsymbol{\delta}^{(\ell)}\|_1}_{\text{delay compactness}}, \tag{36}$$

with $\lambda_s = 10^{-3}$ and $\lambda_d = 10^{-4}$. The spike-rate term penalises deviation from a biologically plausible sparse firing regime Laughlin and Sejnowski [2003], discouraging both silent and saturated activity. Parameters are optimised with AdamW Loshchilov and Hutter [2017] with a learning rate of $\eta = 10^{-3}$, scheduled with `ReduceLROnPlateau` with a $min_lr = 10^{-6}$.

$\mathcal{L}_{\mathrm{CE}}^{\varepsilon}$ is cross-entropy with label smoothing $\varepsilon = 0.05$; $\bar{r}$ is the mean spike rate across all neurons and timesteps in the intermediate SpikeCore layers; $r^* = 0.04$ is a target sparsity level for the ouput layer; $\lambda_s = 10^{-3}$ and $\lambda_d = 10^{-4}$ are regularisation coefficients; and $\boldsymbol{\delta}_c$ collects the delay parameters of all delay layers. The spike-rate term penalises deviation from a biologically plausible sparse firing regime, discouraging both silent and saturated activity.

Standard cross-entropy applied to the final layer's activations is ill-suited to spike-based classifiers for two reasons: first, the output is a binary spike train rather than a continuous logit, so direct CE on a single timestep is highly noisy; second, there is no pressure on the network to maintain biologically plausible sparse activity, leading to either silent or saturated firing regimes that are incompatible with energy-efficient neuromorphic deployment. We address both issues with a composite **firing loss** $\mathcal{L}$, described below.

C.1 Mean Firing Rate Classification Loss

Rather than applying CE to the instantaneous spike output of the final layer, we first compute the mean firing rate across all T timesteps for each of the K output neurons:

$$\bar{o}_k = \frac{1}{T}\sum_{t=1}^{T} o_{t,k}^{(L)} \in [0,\,1], \qquad k = 1, \ldots, K. \tag{37}$$

The mean firing rate $\bar{\boldsymbol{o}} \in [0,1]^{B\times K}$ provides a stable, low-variance estimate of the network's class preference over the full temporal sequence, analogous to rate coding in biological neural populations Dayan and Abbott [2001]. Cross-entropy with label smoothing $\varepsilon = 0.05$ is then applied to $\bar{\boldsymbol{o}} + \epsilon$ (with $\epsilon = 10^{-6}$ for numerical stability):

$$\mathcal{L}_{\mathrm{CE}}^{\varepsilon} = -\sum_{k=1}^{K} \tilde{y}_k \log(\bar{o}_k + \epsilon), \qquad \tilde{y}_k = (1-\varepsilon)\mathbb{1}(k = y) + \frac{\varepsilon}{K}, \tag{38}$$

where $y \in \{1, \ldots, K\}$ is the ground-truth class label and $\tilde{y}_k$ is the smoothed target distribution. Label smoothing with $\varepsilon = 0.05$ prevents the final SpikeCore layer from converging to a degenerate regime where one output neuron fires on every timestep with maximum rate, which would produce $\bar{o}_k \to 1$ and make the loss landscape flat.

C.2 Spike-Rate Regularisation

To encourage biologically plausible sparse activity throughout the SpikeCore stack, we penalise the deviation of the network's mean spike rate from a target sparsity level r^*. Let $\bar{r}$ denote the mean spike rate averaged across all neurons, all timesteps, and all intermediate SpikeCore layers (i.e., all layers except the final classification head):

$$\bar{r} = \frac{1}{|\mathcal{S}|} \sum_{(b,t,n)\in\mathcal{S}} o_{b,t,n}, \tag{39}$$

where $\mathcal{S}$ indexes all spike outputs in the intermediate layers. The spike-rate regularisation term is:

$$\mathcal{L}_{\text{spike}} = \lambda_s \left(\bar{r} - r^*\right)^2, \qquad \lambda_s = 10^{-3}, \quad r^* = 0.04. \tag{40}$$

This squared deviation loss penalises both under-firing ($\bar{r} \ll r^*$), which produces silent networks that learn nothing, and over-firing ($\bar{r} \gg r^*$), which destroys the energy efficiency advantage of spiking computation. The target rate $r^* = 0.04$ (4% of neurons active per timestep) is consistent with estimates of sparse coding in mammalian cortex Laughlin and Sejnowski [2003] and with the firing rates observed on neuromorphic benchmarks Shrestha et al. [2022]. Because $\bar{r}$ is a mean over binary values, its gradient is zero almost everywhere; in practice, gradients flow through the surrogate function $\mathcal{S}(\cdot)$ in the intermediate layers, making $\bar{r}$ differentiable end-to-end.

C.3 Delay L1 Regularisation

Both the polarity-level `LearnableDelayLayer` (Section 3.3) and the feature-level `LearnableDelayConv` (Appendix J) introduce learnable delay parameters. Without regularisation, these parameters tend to drift toward large values during training, producing excessive temporal shifts that misalign the input sequence with the sequence length T and waste representational capacity. The delay regularisation term applies L1 penalties across all delay layers:

$$\mathcal{L}_{\text{delay}} = \lambda_d \sum_{\ell\in\mathcal{D}} \|\boldsymbol{\delta}^{(\ell)}\|_1, \qquad \lambda_d = 10^{-4}, \tag{41}$$

where $\mathcal{D}$ indexes all delay layers in the pipeline and $\boldsymbol{\delta}^{(\ell)}$ denotes their learnable delay parameters. For `LearnableDelayLayer`, $\boldsymbol{\delta}^{(\ell)} = [\delta_0, \delta_1]$ are the two polarity delays; for `LearnableDelayConv`, $\boldsymbol{\delta}^{(\ell)} = \boldsymbol{m}$ are the softmax mixing logits. The implementation handles both cases dynamically:

$$\|\boldsymbol{\delta}^{(\ell)}\|_1 = \begin{cases} \sum_j |\delta_j^{(\ell)}| & \text{if } \texttt{.delays} \text{ attribute present,} \\ \sum_j \sum_i |m_{j,i}^{(\ell)}| & \text{if } \texttt{.all_delays} \text{ attribute present,} \end{cases} \tag{42}$$

providing a unified interface for delay regularisation regardless of the delay layer type. L1 regularisation is preferred over L2 here because L1 induces sparsity in the delay distribution, encouraging most delays to converge to near-zero, with only a small subset of channels learning non-trivial shifts, which is consistent with the biological prior that most axonal projections operate at minimal latency.

C.4 Full Training Objective

The three terms are combined additively:

$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{CE}}^{\varepsilon}(\bar{\boldsymbol{o}}, \boldsymbol{y})}_{\text{classification}} + \underbrace{\lambda_s \left(\bar{r} - r^*\right)^2}_{\text{spike sparsity}} + \underbrace{\lambda_d \sum_{\ell\in\mathcal{D}} \|\boldsymbol{\delta}^{(\ell)}\|_1}_{\text{delay compactness}}, \tag{43}$$

where each term is independently scaled by its coefficient. The relative magnitudes $\lambda_s = 10^{-3}$ and $\lambda_d = 10^{-4}$ were chosen so that at initialisation, with $\bar{r} \approx 0.5$ (before adaptation takes effect) and small random delays, the regularisation terms contribute less than 5% of the total loss, ensuring the classification objective dominates throughout training. Table 7 summarises the effect of each term.

D Datasets

DVSGesture Amir et al. [2017] is an event-based vision dataset recorded with a DVS128 dynamic vision sensor, containing 11 hand gesture classes from 29 subjects under three lighting conditions.

Table 7: Qualitative effect of each loss component on network behaviour.

Term	Without it	With it	Biological analogue
$\mathcal{L}_{\text{CE}}^{\varepsilon}$ (smoothed)	Output layer saturates, train acc $\to 100\%$	Prevents degenerate rate collapse	Graded population readout
$\mathcal{L}_{\text{spike}}$	Silent or saturated intermediate layers	Sparse $\approx 4\%$ activity	Cortical sparse coding
$\mathcal{L}_{\text{delay}}$	Delays drift to $d_{\max}$, misalignment	Compact, interpretable delays	Minimal conduction delay principle

Each sample is a stream of asynchronous polarity events over a 128×128 pixel array, binned into T frames of shape $\mathbb{R}^{T\times 2\times 128\times 128}$.

SHD (Spiking Heidelberg Digits) Cramer et al. [2020] is an auditory spike dataset consisting of spoken digit recordings (0–9, in English and German) converted to spike trains across 700 cochlear frequency channels, yielding samples of shape $\mathbb{R}^{T\times 700}$.

SSC (Spiking Speech Commands) Cramer et al. [2020] is a larger-scale auditory benchmark derived from the Google Speech Commands corpus, spanning 35 spoken word classes with the same 700-channel cochlear encoding as SHD but with significantly more training examples and greater inter-speaker variability.

E Augmentations

All spiking datasets use a shared event-level augmentation pipeline (temporal stretch, jitter, and event dropout), followed by binning, float32 conversion, and per-sample normalisation. Dataset-specific frame-level augmentations are then applied: for SHD/SSC, random frequency and time rolls; for DVSGesture, temporal speed perturbation. Raw events are binned into polarity frames via a fixed-bin `ToFrame` transform. To improve inter-subject generalisation, we apply a two-level augmentation strategy operating at both the raw-event and the post-binning frame levels.

So, the exact following augmentation is shared for all datasets:

Listing 1: Training augmentation pipeline.

```
train_transform = transforms.Compose([
    # Temporal dynamics augmentation
    CustomTimeStretch(0.8, 1.2),
    transforms.TimeJitter(std=2000),

    # Event-level corruption
    transforms.DropEvent(p=0.20),

    # Convert events to frames
    transforms.ToFrame(
        sensor_size=tonic.datasets.DVSGesture.sensor_size,
        n_time_bins=n_time_bins
    ),

    # Normalization
    lambda x: x.astype("float32"),
    lambda x: x / (x.max() + 1e-6)
])
```

We use a sequence of event-based augmentations to improve robustness to temporal and sensor noise. Our standard (in-distribution) pipeline introduces mild temporal distortions via time stretching and jittering, simulating variations in motion speed and timestamp noise. Additionally, random event dropping is applied to mimic sensor sparsity and occlusions. Events are subsequently binned into fixed temporal frames and normalized.

For each dataset, we added specific augmentations like SpeedPerturb for the DVSGesture and RandomTimeRoll as well as RandomFrequencyRoll for SSC and SHD.

Event-level augmentation. Individual spike timestamps are perturbed by additive Gaussian noise, $\Delta t \sim \mathcal{N}(0,\, \sigma_t^2)$ with $\sigma_t = 2{,}000\,\mu$s, and $p_{\text{drop}} = 0.30$ of events are discarded uniformly at random,

simulating sensor read-out noise and partial occlusion. A custom temporal stretch transform samples $r \sim \mathcal{U}(0.85, 1.15)$ and rescales all event timestamps as $t' \leftarrow r \cdot t$, producing physiologically plausible variation in gesture execution speed at the spike level, where spike density within each bin is preserved.

Frame-level augmentation. After conversion to tensors $\boldsymbol{x} \in \mathbb{R}^{T\times 2\times H\times W}$, a *SpeedPerturbation* transform resamples the temporal axis by a rate $r' \sim \mathcal{U}(0.8, 1.2)$ via nearest-neighbour interpolation, then pads or crops back to the original T with random alignment, ensuring fixed sequence length for batch collation. Spatial translation invariance is induced via *RandomRoll*, which cyclically shifts the H and W dimensions by random integer offsets drawn uniformly within the full spatial range. All frames are normalised per-sample by their maximum activation: $\hat{x} = x/(\max(x) + \epsilon)$.

Note that event-level and frame-level speed perturbations are complementary rather than redundant: the former perturbs true interspike timing density before binning, while the latter perturbs the coarser temporal structure of the binned representation.

F Learning Curves.

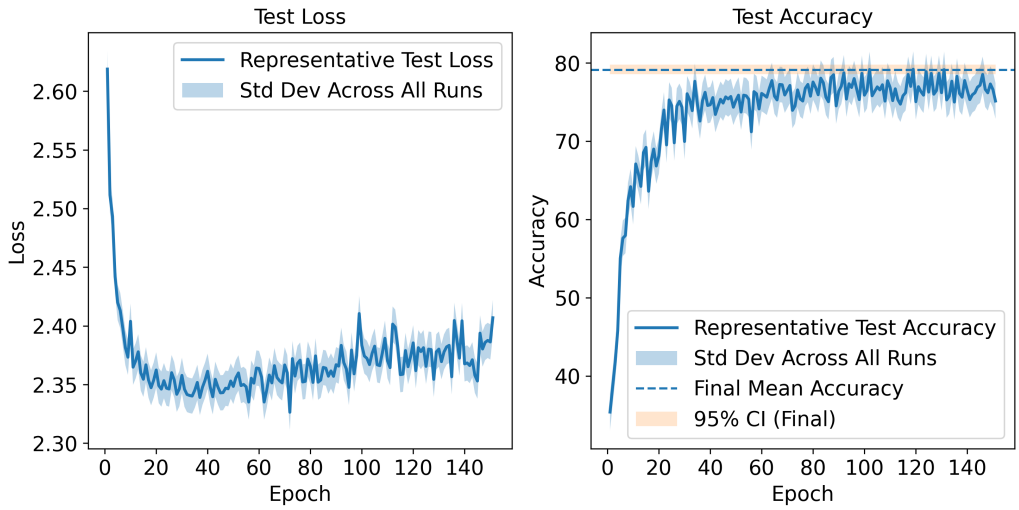
Training protocol and epoch selection. All models in Table 1 are trained for 150 epochs to ensure convergence is fully reached without early stopping. We observe that test accuracy plateaus stably for all variants, SNU, Adaptive SNU, and SpikeCore, after approximately 70–80 epochs, with no degradation in the remaining epochs. Results are reported at the epoch of empirical plateau, which we define as the first epoch after which test accuracy does not improve by more than 0.1% over a window of 10 consecutive epochs. This corresponds to 70 epochs for FMNIST, 60 epochs for Isolet, and 120 epochs for SHD, reflecting the differing complexity and sequence length of each dataset. The plateau behaviour across all variants confirms that the reported comparisons are not sensitive to the choice of stopping epoch within the stable region, and that SpikeCore's accuracy advantage is not an artefact of differential convergence speed.

Figure 5 shows the test loss and test accuracy trajectories of SpikeCore and SNU over 150 training epochs on SHD, using a minimal single-layer architecture without augmentation or additional pipeline components. Both models are trained under identical conditions. Shaded regions represent the standard deviation across five independent runs; the dashed horizontal line marks the final mean test accuracy; and the orange band indicates the 95% confidence interval at the final epoch.

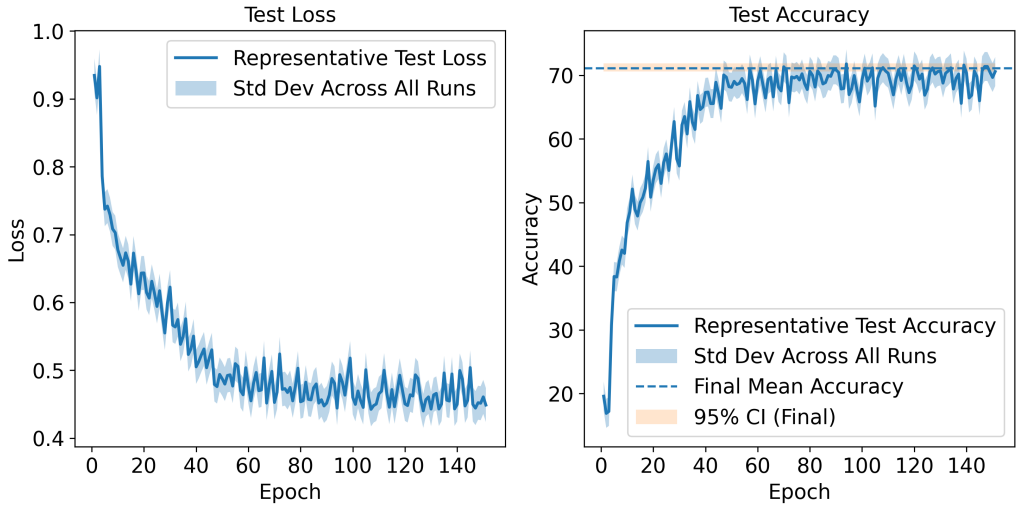
Convergence speed. The two trajectories differ markedly in their initial learning dynamics. SpikeCore reaches approximately 75% test accuracy within the first 20 epochs, a rapid initial rise attributable to the richer gradient signal provided by the learnable per-neuron membrane decay, which allows heterogeneous temporal integration timescales to emerge early in training. SNU, by contrast, rises more gradually, not reaching equivalent accuracy levels until approximately epoch 60–70. This difference is consistent with the broader convergence advantage reported in Section 4.2: biologically-grounded inductive biases reduce the number of gradient steps required to learn effective temporal representations, rather than merely shifting the final accuracy ceiling.

Plateau stability. Both models enter a stable plateau phase after approximately 70–80 epochs, with no accuracy degradation observed through epoch 150. This confirms that the choice to report results at epoch 120 is conservative and not cherry-picked: the accuracy at epoch 120 lies well within the plateau region for both models, and results reported at any epoch between 80 and 150 would be equivalent within the confidence intervals shown. The absence of late-training degradation also demonstrates that SpikeCore's spike-history adaptation mechanism and stop-gradient on the adaptation state do not introduce training instability at extended horizons, a non-trivial property for recurrent spiking units trained via backpropagation through time.

Loss behaviour. The loss trajectories reveal a qualitative difference in optimisation dynamics between the two models. SNU exhibits a smooth, monotonically decreasing loss curve that converges cleanly to a flat minimum around 0.45–0.50. SpikeCore's loss, by contrast, shows higher-frequency oscillations throughout training, stabilising around 2.35–2.40. This difference in loss scale and dynamics is expected: SpikeCore uses the composite firing loss $\mathcal{L}$ from Equation 36, which includes a spike-rate regularisation term in addition to cross-entropy, but we did not apply label smoothing for the sake of fair comparison. These differences introduce competing gradients that naturally produce more oscillatory loss dynamics, particularly early in training when the spike rate regulariser is far from its target $r^* = 0.04$. Importantly, this loss oscillation does not translate to accuracy instability,



(A) SpikeCore (ours).



(B) SNU Woźniak et al. [2020].

Figure 5: Test loss (left) and test accuracy (right) over 150 epochs on SHD for SpikeCore and SNU in a minimal single-layer setup. Solid lines are representative runs; shaded bands are standard deviation across five seeds. The dashed line marks the final mean accuracy and the orange band the 95% confidence interval. Both models plateau stably after approximately 70–80 epochs with no degradation, confirming that reported results at epoch 120 reflect fully converged performance.

the accuracy trajectory of SpikeCore is smooth and consistent across all five seeds, as evidenced by the tight standard deviation band in Figure 5A.

Variance across runs. Both models show tight confidence bands across five independent runs, indicating that the reported results are robust to random seed variation. SpikeCore's standard deviation in final accuracy ($\pm 0.15\%$) is marginally lower than SNU's ($\pm 0.17\%$), suggesting that the learnable decay initialisation strategy, drawing from $\mathcal{U}(0.7, 0.95)$ and inverting through the sigmoid, produces consistent initialisation conditions that reduce seed sensitivity. The 95% confidence intervals at epoch 150 (orange bands) do not overlap between the two models, confirming that the $+8.04$ percentage point accuracy gap is statistically significant and not attributable to run-to-run variance.

Table 8: Convergence efficiency of the proposed model. We report the number of epochs required to reach 90% of the final test accuracy and best-run accuracy on each dataset. The model attains 90% of its final performance substantially earlier than full convergence across all datasets, 20 vs 50 epochs for DVSGesture, 10 vs 50 for SHD, and most notably 30 vs 200 for SSC, highlighting significantly faster early-stage learning.

Dataset	**Final Mean Acc. (%)**	**Best-Run Acc. (%)**	**Epochs to 90% of Final Acc.**
DVS128Gesture	98.7	99.24	20
SHD	94.0	95.3	10
SSC	80.2	80.4	30

G Ablation on epochs to reach 90% of final accuracy

Table 8 demonstrates a consistent and striking pattern across all three benchmarks: the proposed model reaches 90% of its final test accuracy significantly earlier than full convergence. This gap between useful performance and final convergence provides important insight into the learning dynamics of the model.

On DVS128Gesture, the model reaches 90% of its final accuracy within 20 epochs, despite requiring approximately 50 epochs to fully converge. A similar but even more pronounced trend is observed on SHD, where only 10 epochs are needed to reach 90% of final performance. These results suggest that the model rapidly captures the dominant discriminative structure of the data in the very early stages of training. The remaining epochs primarily contribute to incremental refinements rather than substantial performance gains.

The most compelling case is SSC, where the model achieves 90% of its final accuracy after only 30 epochs, compared to roughly 200 epochs required for full convergence. This nearly 7× difference indicates a substantial inefficiency in conventional training schedules and highlights the model's ability to extract high-value representations early, even in more challenging or noisy settings.

Implications for Optimization Behavior These results suggest that the optimization landscape encountered by the model allows for fast initial descent toward a high-quality basin, followed by a prolonged period of fine-tuning. This behavior is characteristic of models with:

Strong inductive biases that align well with the data distribution Stable gradient propagation enabling efficient early updates Representations that quickly organize into separable clusters

Importantly, the relatively small gap between final mean accuracy and best-run accuracy across all datasets indicates low variance across runs. This stability reinforces the claim that early convergence is not due to lucky initialization but reflects a consistent property of the training dynamics.

Practical Significance From a systems and deployment perspective, these findings have several important consequences:

- Training Efficiency: Practitioners can terminate training significantly earlier (e.g., at 20–30% of the full schedule) while retaining the majority of performance. This is particularly valuable for resource-constrained environments or rapid experimentation.
- Energy and Compute Savings: The large gap between early performance and full convergence, especially on SSC, suggests substantial opportunities for reducing computational cost without sacrificing accuracy.
- Early Stopping and Adaptive Scheduling: The consistent early plateau across datasets motivates adaptive training strategies, such as dynamic early stopping or learning rate annealing triggered by reaching a performance threshold rather than fixed epoch budgets.

Dataset-Specific Insights. The variation across datasets further reveals how task complexity interacts with convergence:

- SHD (10 epochs): Extremely rapid convergence suggests that the dataset's structure is highly amenable to the model's inductive biases, enabling near-immediate learning of salient temporal features.

Table 9: Dataset-specific instantiation of the four-stage pipeline. All variants share the PFTGC module and SpikeCore stack; the delay layer and feature extractor differ by input modality. Full details in Appendix H.

Dataset	Input Shape	Stage 1 Delay Layer	Stage 2 Feature Extractor	Stage 3–4 (shared)
DVSGesture	$B \times T \times 2 \times 128 \times 128$	Polarity hard delay (STE, 2 params)	MBSE: 2D multi-branch + SE, 4 modules	PFTGC + SpikeCore ($L = 3$)
SHD	$B \times T \times 700$	LearnableDelayConv (soft, $D_{\max} = 5$)	MBSE-1D	
SSC	$B \times T \times 700$	LearnableDelayConv (soft, $D_{\max} = 5$)	MBSE-1D + InceptionTemporal	

- DVS128Gesture (20 epochs): Slightly slower but still efficient convergence reflects moderate complexity, likely due to greater intra-class variability in gesture patterns.
- SSC (30 epochs vs 200 total): While early learning remains fast, the long tail of convergence indicates a more challenging optimization problem, potentially due to higher noise levels, longer temporal dependencies, or finer-grained class distinctions.

Broader Interpretation Taken together, these results indicate that final accuracy alone is an incomplete measure of model quality. The proposed model distinguishes itself not just by achieving competitive performance, but by doing so with markedly improved training efficiency. This suggests that the model is particularly well-suited for scenarios where time-to-accuracy is critical.

Moreover, the pronounced early convergence raises an interesting research question: whether the later stages of training meaningfully improve representation quality or merely overfit to marginal patterns. This opens avenues for future work on truncated training, curriculum learning, or regularization strategies that explicitly leverage early-stage representations.

to Include in discussions: The model reaches $\approx 90\%$ of its final accuracy within a small fraction of the training budget, highlighting highly efficient early-stage learning; moreover, while we report mean performance across runs, our single best-run accuracy is consistently higher (see Appendix for details).

H Backbone Variants

For different datasets, the backbone is adjusted. For instance, for DVSGesture, we need 2D convolutions, whereas for SSC and SHD, we need 1D convolutions. To capture long-term dependencies for the SSC dataset, we have added an InceptionTemporal module to the backbone with bigger kernel sizes (10, 15, 20) (see Table 9).

InceptionTemporal backbone (for SSC). For 1D auditory inputs of SSC data $\boldsymbol{z} \in \mathbb{R}^{B\times C\times T}$, we use an **InceptionTemporal** module that applies three parallel causal convolutions with kernel sizes $\{10, 15, 20\}$ after a shared bottleneck projection, plus a max-pooled branch. Causal left-padding ensures the output length matches the input, making the extractor streaming-compatible. Each module adds a residual connection:

$$\mathrm{ITM}(\boldsymbol{z}) = \mathrm{GELU}\big(\mathrm{BN}\big([\boldsymbol{c}_{10}\|\boldsymbol{c}_{15}\|\boldsymbol{c}_{20}\|\boldsymbol{c}_{\mathrm{pool}}]\big)\big) + \mathrm{Conv}_{1\times1}(\boldsymbol{z}), \tag{44}$$

where $\boldsymbol{c}_k = \mathrm{CausalConv}_k(\mathrm{bottleneck}(\boldsymbol{z}))$ for $k \in \{10, 15, 20\}$ and $\boldsymbol{c}_{\mathrm{pool}} = \mathrm{Conv}_{1\times1}(\mathrm{MaxPool}(\boldsymbol{z}))$. Three InceptionTemporal modules are stacked, followed by the LearnableDelayConv and a channel projection to 512 dimensions before PFTGC.

I Ablation on the backbone.

To evaluate the contribution of the proposed MBSE backbone, we replace it with a strong 1D residual convolutional baseline and hold all other pipeline components fixed — the learnable delay layer, PFTGC module, and SpikeCore stack remain unchanged. This isolates the effect of the spatial feature extractor on both accuracy and parameter efficiency.

Residual convolutional baseline. The baseline backbone follows a standard ResNet-style design adapted for 1D temporal sequences He et al. [2016]. An input projection first maps the 700-channel cochlear representation to 256 channels, after which four **Residual Convolutional Blocks** process the sequence at constant channel width. Each block applies two causal 1D convolutions with kernel size

$k = 5$ and padding $p = 2$ to preserve sequence length, interleaved with LeakyReLU nonlinearities and Layer Normalisation Ba et al. [2016]. Formally, for input $\boldsymbol{z} \in \mathbb{R}^{B\times C\times T}$:

$$\begin{aligned}
\boldsymbol{z}' &= \mathrm{LN}(\mathrm{LeakyReLU}(\mathrm{Conv}_{k=5}(\boldsymbol{z}))) \,, \\
\boldsymbol{z}'' &= \mathrm{LN}\big(\mathrm{LeakyReLU}\big(\mathrm{Conv}_{k=5}(\boldsymbol{z}')\big)\big) \,, \\
\mathrm{ResBlock}(\boldsymbol{z}) &= \mathrm{LeakyReLU}\big(\boldsymbol{z}'' + \mathcal{P}(\boldsymbol{z})\big) \,,
\end{aligned} \tag{45}$$

where $\mathrm{LN}(\cdot)$ denotes Layer Normalisation applied along the channel dimension after a $(B, C, T) \to (B, T, C)$ permutation, and $\mathcal{P}(\boldsymbol{z})$ is the shortcut projection — a pointwise $\mathrm{Conv}_{1\times1}$ when input and output channels differ, or the identity otherwise. The four stacked blocks operate at constant channel width (256), resulting in repeated quadratic channel interactions across depth. The total parameter count of this baseline is **3.41M**.

Structural source of parameter overhead. The high parameter count of the residual baseline arises from two compounding factors. First, each $\mathrm{Conv}_{k=5}$ layer with $C_{\text{in}} = C_{\text{out}} = 256$ channels contributes $C_{\text{in}} \times C_{\text{out}} \times k = 256 \times 256 \times 5 = 327{,}680$ parameters per convolution — and each residual block contains two such layers, giving $\approx$655K parameters per block before normalisation terms. Four stacked blocks therefore accumulate over 2.6M parameters in the convolutional weights alone, before accounting for the input projection. Second, because channel width is constant throughout, there is no opportunity for the backbone to trade spatial resolution for reduced channel cost as depth increases: every layer pays the full $O(C^2)$ mixing cost regardless of its position in the hierarchy.

MBSE backbone efficiency. By contrast, the MBSE backbone concentrates representational cost differently. Its 1×1 bottleneck projections in branches $\boldsymbol{b}_2$ and $\boldsymbol{b}_3$ (Equations 5–6) reduce the channel depth *before* the expensive 3×3 and 5×5 spatial operations, decoupling channel expansion from spatial mixing. The SE recalibration step then selectively re-weights the concatenated multi-branch representation, providing a form of structured channel pruning that is learned rather than hand-designed. Progressive spatial downsampling via interleaved max-pooling further amortises the cost of deeper modules over a reduced spatial footprint. Together these choices yield a total extractor parameter count of **0.29M** for $D = 256$ — an $11.8\times$ reduction relative to the residual baseline.

Results. Table 10 reports test accuracy and parameter count on SHD for both backbones under otherwise identical pipeline conditions. Replacing the residual baseline (91.30%, 3.41M) with the MBSE backbone achieves 94.04% — a gain of $+\mathbf{2.74}$ percentage points — while simultaneously reducing the parameter count by more than an order of magnitude. This result demonstrates that the performance improvement is not a consequence of additional model capacity: the MBSE backbone achieves *higher* accuracy with *fewer* parameters, confirming that structured multi-branch feature extraction with channel-wise recalibration is a more effective inductive bias for neuromorphic temporal signals than repeated full-channel residual mixing.

Discussion. The accuracy gap between the two backbones ($+2.74\%$) is notable given that it arises at a *lower* parameter budget, which rules out capacity as an explanation. We attribute the improvement to two structural properties of MBSE that are absent from the residual baseline. First, **multi-scale temporal receptive fields**: the parallel branches with 1×1, 3×3, 5×5, and max-pooled paths capture feature interactions at different temporal granularities simultaneously, whereas the residual baseline processes all temporal context uniformly at $k = 5$. Second, **channel-wise recalibration via SE gating**: by learning to selectively amplify informative feature channels and suppress redundant ones, the SE gate provides an adaptive bottleneck that pure residual mixing lacks. These two mechanisms together make MBSE a more parameter-efficient inductive prior for the sparse, event-driven structure of neuromorphic signals, where different temporal scales and channel subsets carry disproportionate amounts of discriminative information depending on the input.

J Feature-Level Learnable Delay Layer (LearnableDelayConv)

Motivation. Motivated by differential response latencies between ON and OFF retinal ganglion pathways Masland [2012] and population-level axonal conduction delays Swadlow [1985], we introduce learnable delay layers before feature extraction.

Polarity-level hard delay. This is one of the cheapest possible delay layers that adds only D parameters (D is the input dimension). One unconstrained scalar $\delta_c \in \mathbb{R}$ is assigned per DVS

Table 10: Comparison between a ResNet baseline and the proposed MBSE backbone on SHD. Classification accuracy and total number of trainable parameters are reported. The proposed MBSE backbone matches better with the rest of our pipeline achieves higher accuracy while using over an order of magnitude fewer parameters.

Backbone	**Accuracy (%)**	**Params (M)**	**↓ Params**
ResNet (1D Residual Blocks)	91.30	3.41	1.0×
MBSE (Ours)	**94.04**	**0.29**	**11.8**×

polarity channel $c \in \{0, 1\}$. The integer delay is obtained by rounding and clamping, with gradients propagated via the straight-through estimator:

$$d_c = \mathrm{clamp}(\lfloor \delta_c \rceil,\ 0,\ d_{\max}), \qquad \hat{x}_{b,t,c} = \begin{cases} x_{b,\,t-d_c,\,c} & t \geq d_c, \\ 0 & t < d_c. \end{cases} \tag{46}$$

Feature-level soft delay. After the feature extractor, a **LearnableDelayConv** layer models each channel's delay as a learned soft distribution over $D_{\max} + 1$ candidate delays. For feature sequence $\boldsymbol{z} \in \mathbb{R}^{B\times D\times T}$, the output is a softmax-weighted mixture of causally shifted copies, each scaled by a depthwise 1×1 convolution f_d:

$$\boldsymbol{y}_{b,c,t} = \sum_{d=0}^{D_{\max}} w_{d,c} \cdot f_d\Big(\boldsymbol{z}^{(d)}_{b,c,t}\Big), \qquad w_{d,c} = \frac{\exp(m_{d,c})}{\sum_{d'} \exp(m_{d',c})}, \tag{47}$$

where $\boldsymbol{m} \in \mathbb{R}^{(D_{\max}+1)\times D}$ is initialised to zero so all delays are equally weighted at the start of training. This layer is fully differentiable, no STE is required.

The polarity-level delay operates on two scalar channels and models a coarse, pathway-level temporal offset. By contrast, once the spatial backbone has produced a rich D-dimensional feature representation, each feature dimension may benefit from a *different temporal routing*, some features may be most informative when aligned to earlier time steps, others to later ones. Hard integer delays are ill-suited here: with D potentially in the hundreds, learning one delay scalar per channel via the straight-through estimator introduces substantial gradient noise and does not allow the network to express *uncertainty* over the optimal delay. Instead, we model the delay for each channel as a **learned soft distribution over a fixed set of candidate delays**, a differentiable approximation closely related to Dilated Convolution with Learnable Spacings (DCLS) Hassani et al. [2023].

So, in addition to the polarity-level delay which operates on the raw two-channel DVS input and is motivated by differential ON/OFF retinal ganglion latency, we introduce a second, complementary delay mechanism applied at the feature level, after the MBSE spatial backbone and before the SpikeCore stack. This layer is motivated by a distinct biological phenomenon: the distribution of axonal conduction delays across thalamocortical and cortico-cortical projections, where a single presynaptic population reaches its postsynaptic targets over a range of arrival times rather than at a single fixed latency Swadlow [1985], Mainen and Sejnowski [1995].

Formulation. Let $\boldsymbol{z} \in \mathbb{R}^{B\times D\times T}$ denote the feature sequence after the spatial backbone, permuted so that the channel dimension precedes time. For a maximum delay of $D_{\max}$ steps, we define $D_{\max} + 1$ candidate delayed copies of the input:

$$\boldsymbol{z}^{(d)}_{b,c,t} = \begin{cases} z_{b,c,t} & d = 0, \\ z_{b,c,t-d} & t \geq d, \\ 0 & t < d, \end{cases} \qquad d \in \{0, 1, \ldots, D_{\max}\}, \tag{48}$$

where zero-padding is applied at the beginning of the sequence for $t < d$, consistent with a causal delay. Each delayed copy is then passed through a per-channel depthwise 1×1 convolution $f_d : \mathbb{R}^D \to \mathbb{R}^D$ (i.e. a learned per-channel scalar gain with no cross-channel mixing), yielding a set of $D_{\max} + 1$ transformed delayed representations. The final output is a softmax-weighted mixture over all candidate delays:

$$\boldsymbol{y}_{b,c,t} = \sum_{d=0}^{D_{\max}} w_{d,c} \cdot f_d\Big(\boldsymbol{z}^{(d)}_{b,c,t}\Big), \tag{49}$$

where the per-channel mixing weights are:

$$w_{d,c} = \frac{\exp(m_{d,c})}{\sum_{d'=0}^{D_{\max}} \exp(m_{d',c})}, \qquad \boldsymbol{m} \in \mathbb{R}^{(D_{\max}+1)\times D}, \tag{50}$$

and $\boldsymbol{m}$ is a learnable parameter matrix initialised to zero, so that all delays are equally weighted at the start of training. The softmax in Equation (50) ensures $\sum_d w_{d,c} = 1$ for each channel c, making the operation a convex combination of the delayed copies and guaranteeing that the output magnitude is bounded by the input magnitude. Crucially, because both $\boldsymbol{m}$ and the depthwise conv weights are continuous and the shift operation in Equation (48) is fixed, the entire layer is **fully differentiable**, no straight-through estimator is required.

Biological interpretation. The weight $w_{d,c}$ can be interpreted as the probability that feature channel c is routed through a synaptic pathway with conduction delay d. A channel for which the softmax concentrates on $d = 0$ behaves as an undelayed feedforward projection; one for which the weight is spread across multiple d values integrates information from a range of past time steps, analogous to a postsynaptic neuron receiving input from an axonal arbour with heterogeneous path lengths. The depthwise gain f_d additionally allows the network to scale the contribution of each delayed copy independently, loosely corresponding to distance-dependent synaptic efficacy modulation observed in long-range cortical projections Swadlow [1985].

Two-stage delay hierarchy. Together with the polarity-level `LearnableDelayLayer` described in Section 3.3, this layer constitutes a two-stage biologically-motivated temporal processing hierarchy:

1. **Sensory pathway delays** (polarity level, Section 3.3): two scalar delays applied to the raw ON and OFF polarity streams before spatial feature extraction, modelling differential retinal ganglion cell response latency.
2. **Conduction pathway delays** (feature level, this appendix): a per-channel soft delay distribution applied to the D-dimensional feature sequence after the backbone, modelling population-level axonal conduction delay distributions.

This separation is intentional. The polarity-level delay operates in the original spatial domain $\mathbb{R}^{B\times T\times 2\times H\times W}$ and must handle the full $H \times W$ spatial resolution, a fine-grained, channel-sparse operation well-suited to a hard shift. The feature-level delay operates on the compact latent sequence $\mathbb{R}^{B\times D\times T}$ where spatial information has already been pooled, and the richer per-channel expressivity of the soft mixture is both affordable and appropriate.

Computational cost. The layer introduces $(D_{\max}+1)\times D$ mixing parameters in $\boldsymbol{m}$ and $(D_{\max}+1)\times D$ depthwise conv weights, totalling $2(D_{\max}+1)\times D$ parameters. For $D_{\max} = 5$ and $D = 256$, this amounts to 3,072 additional parameters, less than 0.75% of the MBSE backbone, with a forward pass cost of $O(D_{\max} \times D \times T)$ per sample, which is negligible relative to the spatial feature extraction stage.

K Surrogate-gradient sensitivity and measured firing activity

Purpose and experimental protocol. We investigate whether changing the surrogate derivative affects the learning signals received by SpikeCore's learnable decay and adaptation parameters, and quantify the resulting network activity at a trained checkpoint. We use a separate SSC diagnostic run with $T = 50$ temporal bins, a 512-neuron hidden SpikeCore layer, and a 35-neuron output layer. The model was trained for 50 epochs with seed 42 and batch size 32, using AdamW with an initial learning rate of 10^{-3} and weight decay 10^{-3}. Training used temporal and frequency augmentation and a validation-driven ReduceLROnPlateau scheduler.

Controlled surrogate perturbation. Let z denote the input to the spike function and let $\mathrm{sg}(\cdot)$ denote stop-gradient. We implement the surrogate family

$$\widetilde{H}_s(z) = \sigma(sz) + \mathrm{sg}[H(z) - \sigma(sz)], \qquad s \in \{0.5, 1, 2\}, \tag{51}$$

where $H(z) = \mathbb{1}[z > 0]$. Thus, the forward output remains binary, while the derivative used in backpropagation is

$$\frac{\partial \widetilde{H}_s}{\partial z} = s\,\sigma(sz)\big(1 - \sigma(sz)\big). \tag{52}$$

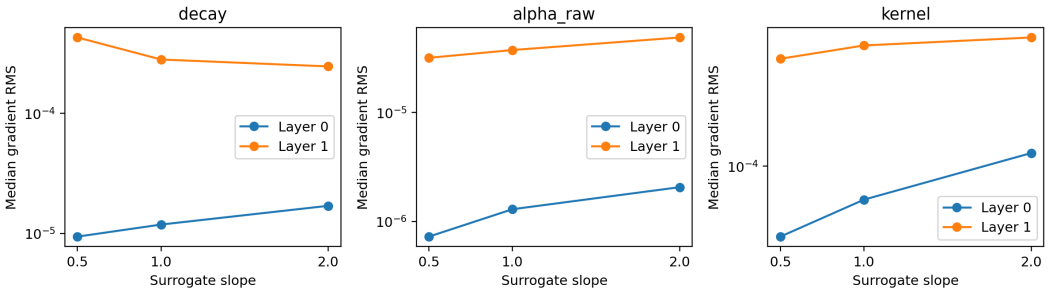


Figure 6: Local surrogate-gradient sensitivity at the validation-selected SSC diagnostic checkpoint. Panels show median gradient RMS for the raw decay parameter, raw adaptation parameter, and synaptic kernel. Layer 0 is the 512-neuron hidden layer and Layer 1 is the 35-neuron output layer. Each point summarizes 50 minibatches comprising 1,600 test examples. Weights and forward spike outputs are held fixed across surrogate slopes; no retraining is performed.

Table 11: Local gradient changes relative to the default surrogate slope $s = 1$. Entries are median cosine similarities C_s and median norm ratios R_s over 50 minibatches. The reference slope has $C_1 = R_1 = 1$. All sampled parameter-group gradients were present and finite.

		$s = 0.5$		$s = 2$	
Layer	Parameter	C_s	R_s	C_s	R_s
Hidden	Decay	0.904	0.663	0.910	1.275
Hidden	Adaptation	0.853	1.195	0.743	0.740
Hidden	Kernel	0.907	0.696	0.853	1.202
Output	Decay	0.974	0.939	0.965	1.026
Output	Adaptation	0.975	1.100	0.986	1.010
Output	Kernel	0.984	0.931	0.982	1.052

Changing s changes both the width and peak magnitude of the surrogate derivative.

For each of 50 randomly selected test minibatches of 32 examples, we compute gradients under all three slopes using identical weights and inputs. The model remains in evaluation mode, no optimizer updates are performed, and equality of the forward spike outputs is checked. The diagnostic therefore measures local backward sensitivity at the selected checkpoint. It does not compare models trained independently with different surrogate functions.

Gradient measurements. We inspect the unconstrained parameters underlying membrane decay and adaptation, together with the synaptic kernel, in each SpikeCore layer. In the implementation, these parameter groups are named `decay`, `alpha_raw`, and `kernel`. For a flattened parameter-group gradient g_s, we compute its root-mean-square magnitude,

$$\mathrm{RMS}(g_s) = \sqrt{\frac{1}{|g_s|}\sum_j g_{s,j}^2}, \tag{53}$$

its cosine similarity to the default-slope gradient g_1, and its relative norm:

$$C_s = \frac{g_s^\top g_1}{\|g_s\|_2\|g_1\|_2}, \qquad R_s = \frac{\|g_s\|_2}{\|g_1\|_2}. \tag{54}$$

We report medians over minibatches. These medians summarize variation across sampled data at one checkpoint; they are not uncertainty estimates across independent training seeds.

Interaction with decay and adaptation. All inspected parameter-group gradients were present and finite on every probe minibatch. As shown in Table 11, decay gradients retain median cosine similarities of at least 0.904 under the tested perturbations. Output-layer adaptation gradients are also closely aligned with the default, with median cosine similarities above 0.975. Hidden-layer adaptation is more sensitive: increasing the slope to $s = 2$ reduces its median cosine similarity to 0.743 and its median norm ratio to 0.740.

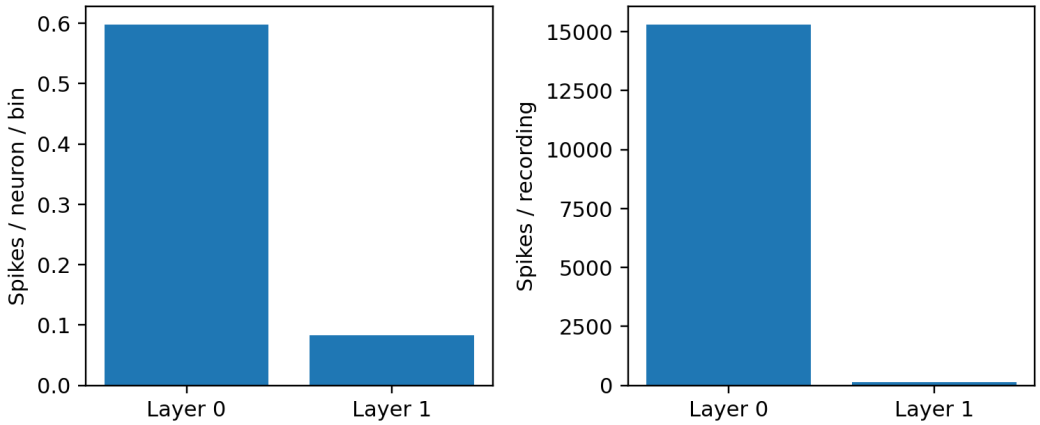


Figure 7: Measured SpikeCore activity over the complete SSC test split for the validation-selected diagnostic checkpoint. Left: mean spikes per neuron per temporal bin. Right: mean spikes per recording. Layer 0 contains 512 hidden neurons; Layer 1 contains 35 output neurons. Measurements use $T = 50$ and include all 20,382 test recordings. Counts include SpikeCore outputs and exclude the dense front-end activations.

These observations show that the surrogate affects both the magnitude and direction of the local parameter update. The sigmoid-constrained decay and detached spike-history update do not make parameter learning invariant to surrogate choice. Furthermore, gradients of the raw parameters include the derivatives of their respective parameterizations; differences in RMS magnitude between parameter groups should not be interpreted directly as differences in importance or learning quality.

Layer-wise firing statistics. We measure binary spike outputs on all $N = 20{,}382$ test recordings, using the same selected checkpoint and evaluation mode. For layer ℓ with n_ℓ neurons, the mean firing rate is

$$r_\ell = \frac{\sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{j=1}^{n_\ell} h_{i,t,j}^{(\ell)}}{N T n_\ell}, \tag{55}$$

and the mean spike count per recording is

$$S_\ell = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{j=1}^{n_\ell} h_{i,t,j}^{(\ell)} = T n_\ell r_\ell. \tag{56}$$

These rates are dimensionless spikes per neuron per temporal bin, rather than firing frequencies in hertz.

Figure 7 shows approximately 60% firing activity in the hidden layer and 8% in the output layer. Together, the two layers emit an average of 15,461.82 spikes per recording. The hidden layer dominates the total count because it combines higher activity with a substantially larger neuron population.

Implications and limitations. The measured activity is heterogeneous across layers. In particular, the output firing-rate regularization target of 0.04 should not be interpreted as a measured network-wide firing rate or as a constraint on hidden-layer activity. The present measurements provide an empirical activity profile for this checkpoint, but do not directly measure energy consumption. Converting activity into energy requires operation-specific accounting and assumptions about the hardware mapping, including the dense front end, neuron-state updates, and memory access.

The gradient study likewise has a defined scope: it establishes finite sampled gradients and quantifies their local sensitivity to three surrogate slopes at one trained checkpoint. It does not establish unbiased gradients, rule out vanishing gradients at other training stages or temporal positions, or demonstrate equal convergence across independently trained surrogate variants. The experiment evaluates pre-segmented classification at $T = 50$; continuous streaming and longer-unroll optimization are outside this diagnostic.

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 abstract and introduction accurately reflect the scope and contributions of the work.

   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: We have included a separate section, discussing the limitations of the work and suggestions for future work.

   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: [N/A]

   Justification: We include comprehensive ablation studies and analyses of the different components of our method. This includes evaluations of individual SpikeCore units, comparisons of the MBSE backbone with ResNet-style backbones, and assessments of the trainability and discriminative performance of SpikeCore against adLIF and SNU.

   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 experimental details to support reproducibility. Our method is built using standard PyTorch modules, making it straightforward to integrate with existing frameworks and workflows.

   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: Our experiments are based on open-source standard benchmarks and existing methods, with the source code available for public upon publication.

   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: We provide all details of the experiments in the Appendix.

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 the mean and standard deviation over 5 independent runs for each experiment to ensure statistically reliable 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: We have provided information about the computing platform and the source code for easy reproduction as well.

   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: Our research adheres to the NeurIPS Code of Ethics in every respect.

   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: [No]

    Justification: This work is primarily methodological and does not target a specific real-world deployment. However, it may have indirect positive societal impacts by improving the efficiency of neuromorphic data processing, which could support the development of low-power and real-time machine learning systems, particularly in edge computing or robotics.

    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 poses no such risks.

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 original paper for datasets we used are all cited.

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: [No]

Justification: We have used only public datasets.

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 nor 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 nor 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: The main methods in this research don't rely on large language models (LLMs) in any meaningful, original, or non-standard way. The LLMs were only used to check grammar and help tidy up the writing.

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.