arXiv:2602.01978v2 [cs.NE] 19 Nov 2026

SpikingGamma: Temporally Precise Online SNN Training Through Smoothed Temporal Delays

Roel Koopman
CWI
roel.koopman@cwi.nl

Sebastian Otte
University of Luebeck
sebastian.otte@uni-luebeck.de

Sander Bohté
CWI
University of Amsterdam
s.m.bohte@cwi.nl

Abstract

Neuromorphic hardware implementations of Spiking Neural Networks (SNNs) promise energy-efficient, low-latency AI through sparse, event-driven computation. Yet, training SNNs under fine temporal discretization remains challenging, hindering low-latency responsiveness and the mapping of software-trained SNNs to efficient hardware. In current approaches, spiking neurons are modeled as self-recurrent units, embedded into recurrent networks to maintain state over time, and trained with BPTT or RTRL variants based on surrogate gradients. These methods scale poorly with temporal resolution, while online approximations exhibit instability for long sequences and tend to fail at capturing temporal patterns. To address these limitations, we develop SpikingGamma, an SNN architecture in which neurons maintain a compact bank of smoothed-delay states and communicate through sigma-delta spike-coding. We show that in feedforward networks, SpikingGamma supports direct online error backpropagation through the continuous reconstruction signal, avoiding surrogate gradients through the spike discontinuity. This enables stable learning of temporal patterns with minimal spiking and scales feedforward SNNs to complex tasks and benchmarks with competitive accuracy, all while remaining robust to the temporal resolution of the model. Our approach offers both an alternative to recurrent SNNs trained with surrogate gradients, and a novel route for mapping SNNs to neuromorphic hardware.

1 Introduction

Inspired by the brain, Spiking Neural Networks (SNNs) [1] hold promise for energy-efficient AI models [2, 3] as they use sparse and discrete spikes for communication and enable event-driven, asynchronous information processing. Spiking neurons can thus respond only when needed, achieving low-latency responsiveness while lowering energy consumption, especially when tasks exhibit extended temporal dynamics. Consequently, SNNs are well-suited for real-world tasks such as autonomous driving [4], auditory signal processing [5], and drone control [6]. Treating spiking neurons as self-recurrent neural units and using a so-called Surrogate Gradient (SG) to backpropagate errors through the discontinuity of the spiking process [7], current SNNs now demonstrate competitive performance with classical neural networks for small scale tasks [8, 9].

Yet, since current SNNs are typically treated as time-stepped Recurrent Neural Networks (RNNs), they inherit the training drawbacks of recurrent models: vanishing and exploding gradients, and the need to preserve or approximate past influences on current state. Exact methods such as Backpropagation Through Time (BPTT) and Real-Time Recurrent Learning (RTRL) [10] incur prohibitive memory or timestep complexity, while online approximations [11, 12] often degrade when scaling in sequence

Code is available at: https://github.com/RoelMK/SpikingGamma

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

length or network size. Surrogate gradients further compound this issue and can limit attainable sparsity, as excessive sparsity amplifies gradient vanishing and can collapse training [10].

This scaling problem is especially restrictive for hardware-compatible SNNs. Low-latency operation and faithful deployment to neuromorphic substrates require training over many small timesteps, so model dynamics must closely match physical device dynamics [13, 14, 15]. Fine-grained temporal simulation therefore directly conflicts with the dominant recurrent SNN training paradigm, including, as we show, online BPTT/RTRL approximations such as FPTT [16] and ES-D-RTRL [12].

Here, we introduce the SpikingGamma model, which enables online training at arbitrarily fine temporal resolution. Building on causal-delay models including the Gamma-model [17], TKRNN [18], and Fractionally Predictive SNNs [19, 20], SpikingGamma stores temporal history in recursively computed smoothed-delay kernels rather than recurrent state. Unlike discrete-delay SNNs [8], whose memory requirements grow linearly with temporal resolution, this representation can be evaluated at finer temporal discretizations without introducing a corresponding memory cost.

In the SpikingGamma model, spiking neurons use adaptive recursive memory to construct delayed representations of past inputs, while sigma-delta spike-coding [21, 20] encodes the rectified internal state into a spike train that downstream neurons can reconstruct, analogous to pulsed sigma-delta coding in electrical circuits [21]. This combination removes self-recurrency and allows feedforward SNNs to be trained directly by error backpropagation rather than BPTT, without surrogate gradients through the spiking discontinuity. The resulting internal delays allow SpikingGamma neurons to detect co-occurrences of temporally disjoint features without externally maintained memory, such as persistent activity. SpikingGamma thus enables a causal feedforward model where the state summarizing past presynaptic events replaces the unrolled recurrent state for temporal credit assignment while retaining spike-based communication.

Here, we show how feedforward SpikingGamma SNNs can compete with state-of-the-art recurrent SNN approaches. We show that SpikingGamma SNNs can learn sparse and precise temporal coding over long timespans, while being insensitive to the temporal resolution of the simulated dynamics. We demonstrate this for both simple delay tasks, an echo-location task, and standard neuromorphic benchmarks such as DVS Gesture, SHD and SSC, where we show superior accuracy compared to current online methods with scalable temporal resolution. Our approach thus presents an alternative to current recurrent SNNs based on Surrogate Gradient training, where SpikingGamma SNNs can be trained and computed online and at arbitrarily high temporal resolution.

2 Related work

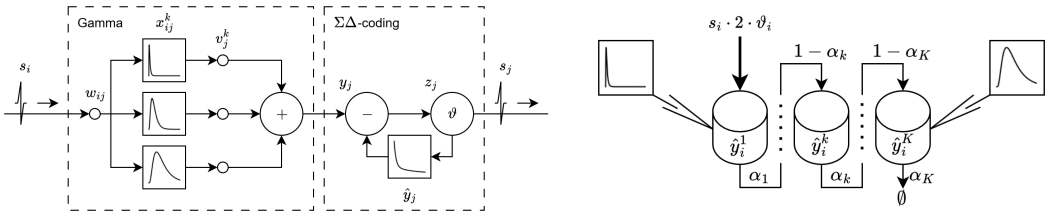
Deep SNNs have been trained via ANN-to-SNN conversion, spiking variants of backpropagation, and biologically motivated plasticity rules such as STDP, each with distinct trade-offs in accuracy and hardware efficiency [22]. Where surrogate gradients have enabled training of SNNs with standard recurrent deep learning approaches, several alternatives to BPTT and RTRL have been explored to improve the training efficiency of SNNs. ANN-to-SNN conversion, while straightforward and scalable, fails to exploit the temporal dynamics and event-driven sparsity of SNNs, and most existing methods remain constrained to inefficient rate-based coding schemes [23]. Various approximations of RTRL and BPTT have been proposed that maintain temporal traces for credit assignment or regularize gradients through time rather than computing exact gradients, such as DECOLLE [11], OSTL [24], OTPE [25], FPTT [26], OTTT [27], and ES-D-RTRL [12]. These methods have shown limited success on tasks demanding complex temporal reasoning. Biologically motivated work approximates backpropagation through space and time online, using only local signals [28]. Another line of work seeks to exploit the inherent sparsity of SNN activity to reduce the memory footprint of BPTT, such as EventProp [29], though its scalability to large models and datasets remains unexplored.

Our work is inspired by a different idea: maintaining a compressed representation of history within the forward pass to enable temporal learning without requiring BPTT. Related ideas have been explored in ANNs. For example, in the TKRNN [18] all neurons are linked through all timesteps using a learned weight matrix, akin to delays, WaveNet [30] uses dilated causal convolutions to model long-range temporal dependencies in a feedforward architecture, performing comparably to LSTM-based RNNs, and Legendre Memory Units [31] compress a sliding input window into a fixed-size continuous-time state, but inside a recurrent cell trained with BPTT. The Gamma Model [17] similarly maintains adaptive internal memory representing past inputs, functioning like a form of smoothed temporal

convolution. Notably, so-called time-cells have been identified in the brain with similar delayed response properties [32], and Gamma-models have been proposed to account for internal dynamics in biological neurons such as spike-rate adaptation [33]. Smoothed-delay representations have also been used to model temporally disjunct cue–reward associations in neuroscience, but not scaled to multilayer SNNs or complex temporal benchmarks [34]. A related approach was applied to SNNs to avoid BPTT [19], that delay-structure however is incompatible with scalable learning as it cannot be computed in a recurrent manner – the concepts introduced in [17] and [19] however form the basis for our work.

3 Methods

3.1 The neuron model



(a) (b)

Figure 1: (a) Neural processing model. Incoming spikes generate weighted synaptic currents across multiple timescales. The neuron weights and sums these responses into a continuous signal, which is converted back to spikes via $\Sigma\Delta$-coding so downstream synapses can reconstruct it. (b) Temporal kernel computation as a cascade of leaky "buckets" draining into one another at rates α_k, with each bucket representing one kernel.

In the SpikingGamma model, we base the spiking neuron model on a Linear-Non-Linear Sigma Delta (LNL-SD) temporal filtering model [20], where a neuron j computes an internal signal y_j as the sum of the weighted and filtered inputs into the neuron. To implement sigma-delta spike-coding, the neuron additionally tracks a signal $\hat{y}_j$ as the sum of spike-triggered refractory responses [35]: as in sigma-delta coding [21], this sum of refractory responses approximates the rectified y_j by emitting a spike and adding a refractory response whenever the positive approximation error between $\hat{y}_j$ and y_j exceeds a threshold ϑ [36]. The approximation of y_j is thus encoded by the spike-train determined by those threshold exceedances as downstream neurons can reconstruct the signal $\hat{y}_j$ at their input – the neural processing model is illustrated in Figure 1a. Mathematically, the signal y_j is computed as:

$$y_j(t) = \text{ReLU}(x_j(t)) \tag{1}$$

with x_j the unrectified neuron signal. This is computed by filtering the synaptic input signals x_{ij}^k from neuron i to neuron j for $k < K$ with each synapse having K temporal kernels. The filters can be instantiated as per-neuron or per-synapse. For per-neuron this is described as:

$$x_j(t) = \sum_k \sum_i x_{ij}^k(t) \cdot v_j^k \tag{2}$$

where v_j^k is a parameter that weights the value of kernel k for neuron j. We refer to this as the "bucket weight". The synaptic signal x_{ij}^k is then computed as:

$$x_{ij}^k(t) = \hat{y}_i^k(t) \cdot w_{ij} \tag{3}$$

where w_{ij} is the synaptic weight. The term $\hat{y}_i^k(t)$ denotes the contribution of temporal kernel k at time t to the estimate of the signal of neuron i, as encoded by the output spikes t_i from neuron i to neuron j:

$$\hat{y}_i^k(t) = \sum_{t_i \leq t} \kappa^k(t - t_i) \tag{4}$$

where t_i denotes spike-times of neuron i up to time t, and κ^k denoting a set of temporal kernel functions implementing different delays. Amplitude modulation is ignored here.

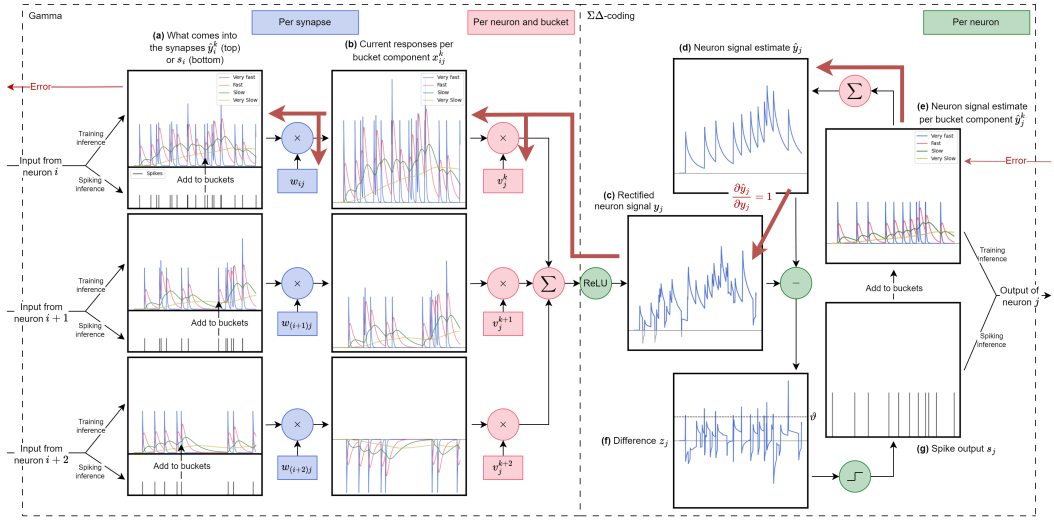


Figure 2: Signal evolution and error propagation in the neuron model. **(a)** Input synapses receive presynaptic signal estimates $\hat{y}_i^k$ during training, or reconstruct them from incoming spikes s_i during spiking inference via Eq. (5). **(b)** Synaptic weights w_{ij} scale each input into per-bucket current responses. The negative weight $w_{(i+2)j}$ inverts the responses of input $i+2$. **(c)** Bucket weights v_j^k (or v_{ij}^k for per-synapse weighting) combine these responses, which are summed and rectified to form y_j. The horizontal line marks zero, and gray shows the negative values removed by rectification. **(d–g)** Sigma-delta coding maintains an estimate $\hat{y}_j$ in the same kernel basis, emits a spike s_j when the mismatch $z_j = y_j - \hat{y}_j$ exceeds threshold, and adds this correction back into the estimate. Because $\hat{y}_j^k$ and $\hat{y}_i^k$ share the same representation, $\hat{y}_j^k$ can be passed directly to downstream synapses during training. Red arrows denote the training error path: gradients flow from the signal estimate to the neuron signal, inputs, and parameters, bypassing spikes and avoiding surrogate gradients.

Following the Gamma-model [17, 33] the kernels κ^k are computed as a series of "buckets" that spill over into each other. This is illustrated in Figure 1b, and described as:

$$\hat{y}_i^k(t) = \begin{cases} \hat{y}_i^k(t-1)\alpha_k + s_i(t)\,2\,\vartheta_i(t) & k = 0, \\ \hat{y}_i^k(t-1)\alpha_k + \hat{y}_i^{k-1}(t-1)(1-\alpha_k) & k > 0. \end{cases} \tag{5}$$

with α_k specifying the transfer rate between buckets k and $k-1$, $\vartheta_i(t)$ denoting the threshold function of the spiking neuron, as defined in Section 3.3, and $s_i(t) = 1$ if $t \in \{t_i\}$ for an output spike-train $\{t_i\}$, else 0. The factor 2 ensures that $\hat{y}$ matches the magnitude of y instead of being biased low [36]. As this is a set of linearly coupled differential equations, each $\hat{y}_i^k$ can be reformulated as Eq. (4) with each κ^k not relying on any other kernel: each κ^k, and thus each bucket $\hat{y}_i^k$ can be modeled in a feedforward fashion without recurrency – that is, knowing the spike times of the inputs $t_i < t$, one can exactly compute the value of $\hat{y}_i^k(t)$ for any t [19].

Spike thresholding implements sigma-delta spike-coding [21, 20] and is done by first computing a variable $z_j(t)$ that is similar to the membrane potential by subtracting an approximation of the signal $\hat{y}_j$, which is what will be encoded by the to be emitted spikes, from the actual internal signal y_j:

$$z_j(t) = y_j(t) - \hat{y}_j(t-1), \quad \text{with} \quad \hat{y}_j(t) = \sum_k \hat{y}_j^k(t) \tag{6}$$

where $\hat{y}_j^k$ is defined per Eq. (5) for the output of neuron j itself. When the difference exceeds a threshold, the signal approximation $\hat{y}_j(t)$ is then updated by emitting a new spike:

$$s_j(t) = z_j(t) > \vartheta_j(t), \quad t_j = t \text{ if } s_j(t) = 1 \tag{7}$$

with $\{t_j\}$ denoting the output spike-train. Figure 2 visualizes the complete forward path together with example signal values at each stage. Pseudocode is provided in Appendix A.

Thus formulated, both the signal approximation at the output of the neuron and the signal received at the downstream synapses are the same, making it possible to directly use $\hat{y}_j^k(t)$ for computations at

those synapses (i.e., for Eq. (3)). To filter per synapse, the only change needed to the model above is to use individual per-synapse bucket weights v_{ij}^k instead of v_j^k. Notably, SpikingGamma can be interpreted as a nonlinear SSM with linear latent dynamics and dual readouts (Appendix B).

3.2 Error-backpropagation

For classification, given some desired output label $y_{\text{true}}(t)$ and the actual neuron signal of the output neuron $y_{\text{out}}(t)$, we compute a loss such as the Cross-Entropy (CE) loss $L(t) = L^{\text{CE}}(y_{\text{out}}(t), y_{\text{true}}(t))$, evaluated independently at each discrete timestep, since temporal credit assignment is handled explicitly via bucketed histories rather than recurrent state (see Figure 3). If the task requires a precise spike timing, we instead use a Mean Squared Error (MSE) loss between the expected $\hat{y}_{\text{true}}(t)$ that would follow from a spike in the correct class at the right time, and actual neuron output $\hat{y}_{\text{out}}(t)$. Then, $L(t) = L^{\text{MSE}}(\hat{y}_{\text{out}}(t), \hat{y}_{\text{true}}(t))$.

As illustrated in Figure 2, for each discrete timestep t, we use the variables at that timestep to compute the loss with respect to the synaptic weights w_{ij} and the bucket weights v_j^k (or v_{ij}^k in the case of per-synapse filtering) (we omit the explicit time index (t)):

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial y_j}\frac{\partial y_j}{\partial x_j} \times \sum_k \frac{\partial x_j}{\partial x_{ij}^k}\frac{\partial x_{ij}^k}{\partial w_{ij}} \tag{8}$$

$$\frac{\partial L}{\partial v_j^k} = \frac{\partial L}{\partial \hat{y}_j}\frac{\partial \hat{y}_j}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial v_j^k} \tag{9}$$

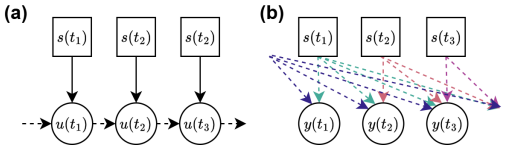


Figure 3: Transition of neuron states over time in recurrent architectures (a) versus SpikingGamma (b). Recurrent models rely on internal states (e.g., membrane potential u) and require BPTT for training, while SpikingGamma does not as it has access to the entire history via buckets for each timestep.

where $\frac{\partial L}{\partial \hat{y}_j}$ follows directly from the definition of the loss function. We avoid differentiating through the binary spike-generation function by propagating the error through the continuous reconstruction path, using the identity estimator $\partial \hat{y}_j / \partial y_j \approx 1$, justified when $\hat{y}_j \approx y_j$ through sigma-delta coding (see Appendix C.8 for an empirical analysis of this approximation).

The other terms follow directly from their forward pass, namely: $\frac{\partial y_j}{\partial x_j} = \text{ReLU}'(x_j)$, $\frac{\partial x_j}{\partial x_{ij}^k} = v_j^k$, and finally $\frac{\partial x_{ij}^k}{\partial w_{ij}} = \hat{y}_i^k$ and $\frac{\partial x_j}{\partial v_j^k} = \sum_i x_{ij}^k$ (again, replace v_j^k with v_{ij}^k for per synapse filtering). To traverse the computational tree over a layer, we use $\frac{\partial x_{ij}^k}{\partial \hat{y}_i^k} = w_{ij}$. Since neuron i produces K components $\hat{y}_i^k$, their gradients are aggregated as $\frac{\partial L}{\partial y_i} = \sum_k \frac{\partial L}{\partial \hat{y}_i^k}$ to continue backpropagation.

3.3 Initialization, regularization and constraints

Before training, parameters are initialized. During training, we apply regularization and add constraints to prevent overfitting, improve model generalization, and increase sparsity. We use standard methods as summarized below, and additionally describe our *Adaptive Thresholding* and *Bucket-transfer Rate Initialization* in detail.

Dropout on neuron signal. Randomly zeros out the neuron signal y with a given probability during training. The decision to drop is independent according to a Bernoulli distribution. Applied to every layer except the output layer.

Layer normalization. Normalizes the neuron signal before rectification (x) based on layer statistics [37]: $x_{\text{norm}} = \frac{x - \text{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \cdot \gamma + \beta$, with γ the gain, β the bias (both trainable), and $\text{E}[x]$ and $\text{Var}[x]$ respectively the mean and variance of the neuron signal over all neurons in the layer. Group normalization can be used as an alternative when specified (see Appendix D.4 for details).

Gain loss. Penalizes the magnitude of the gain term γ_l of the normalization layers: $L_{\text{gain}} = G \cdot \sum_l \gamma_l^2$, where G is a constant that affects the relative importance of the gain loss term, and l is the normalization layer index. This shrinks the signal, in turn reducing the number of spikes.

Weight initialization. Synaptic weights and biases on fully-connected and convolutional layers are sampled from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ with $k = \frac{1}{\text{input features}}$. The bucket weights are sampled from $\mathcal{N}(0, 0.1)$. Layer normalization gains and biases are initialized to 1 and 0, respectively.

Adaptive thresholding. In the sigma-delta spike-coding model, whenever the difference between y and $\hat{y}$ exceeds a fixed threshold, a constant input s (pulse) is added to $\hat{y}$. As y increases, the leak from the buckets (which grows relative to $\hat{y}$) eventually balances this constant input. Beyond that point, if y reaches an even higher value, the leak will prevent $\hat{y}$ from approaching y. Consequently, the relation $\frac{\partial \hat{y}}{\partial y} = 1$ no longer holds, which can disrupt training.

To overcome this, we use an adaptive thresholding mechanism [36] that increases the threshold, and therefore the amount added to the buckets, in proportion to $\hat{y}(t-1)$:

$$\vartheta(t) = \vartheta_0 + \hat{y}(t-1) \cdot m_f \tag{10}$$

where ϑ_0 is the minimum threshold, and m_f is a constant scaling factor. In our experiments, we set $m_f = \vartheta_0$. With this adaptive thresholding, $\hat{y}$ can approximately track any positive value of y (Figure C2, Appendix). With adaptive thresholds, binary spikes suffice only if each receiving synapse replicates the sender's threshold by tracking $\hat{y}$ per synapse rather than per neuron, at the cost of extra per-synapse state and computation. Otherwise, spikes must carry the threshold value as a graded payload [36], which requires hardware support for graded spikes.

Bucket transfer rate initialization. Following [19], we ensure that the combined sum of buckets follows a power-law-like curve to facilitate learning multi-timescale temporal filtering. To achieve this, we generate linearly separated values $l_k \in \text{linspace}(L_{\text{start}}, L_{\text{end}}, K)$, with $L_{\text{start}} \in (0, 1)$ denoting the starting value, $L_{\text{end}} \in (0, 1)$ the final value (both included in the generated value range), and K the number of values, one for each bucket. This is then used to compute the transfer rates $\alpha_k = (l_k)^F$, with $F \in (0, 1)$ the transfer rate factor. Combined with L_{start}, L_{end}, and K this determines the shape of the curve. For all experiments, $L_{\text{start}} = 0.001$ and $L_{\text{end}} = 0.86$. In Table C5 (Appendix), the neuron response for different values of F are shown.

4 Experiments and results

We demonstrate how deep feedforward SpikingGamma SNNs can (1) learn to detect and respond to precisely timed spikes, with minimal spiking, (2) achieve accuracy in temporally sensitive benchmarks competitive with and exceeding current online learning approaches for SNNs, and, (3) scale without loss of accuracy to fine-temporal resolutions which are infeasible with exact recurrent learning and where current approximate online learning approaches fail.

4.1 Learning exact timings through delayed responses

We demonstrate the ability of SpikingGamma to efficiently learn to detect fine temporal structure in two examples involving temporally structured computations.

Learning delays. In the first task, a single input spike needs to be propagated with a precise delay through a hidden neuron to a single output neuron. This task allows us to compare BPTT learning with BP learning in the feedforward SpikingGamma model.

Specifically, an input is provided at $t = 0$, and the network is trained to produce an output at $t = 150$. Each of the two trainable layers in the network contains a single neuron, with only the bucket weights being trainable while the synaptic weights between input and hidden, and hidden and output neuron remain fixed. We compute the MSE loss between the actual output and expected output per timestep. We define output per the trace $\hat{y}$ that follows from the spike output.

For BPTT with SGs, the gradient is computed as:

$$\frac{\partial L}{\partial v_j^k} = \frac{\partial L}{\partial \hat{y}_j(t)} \frac{\partial \hat{y}_j(t)}{\partial s_j(t)} \frac{\partial s_j(t)}{\partial z_j(t)} \frac{\partial z_j(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial x_j(t)} \left(\sum_{t'<t} \frac{\partial x_j(t)}{\partial x_j(t')} \frac{\partial x_j(t')}{\partial v_j^k} + \frac{\partial x_j(t)}{\partial v_j^k} \right) \tag{11}$$

BPTT propagates back through the spikes via an SG z_j and then back in time through the buckets, rather than treating the system as feedforward as in the SpikingGamma model.

In Figure 4, we show neuron signal dynamics after training (left) and bucket-weight evolution during training (right). Both approaches succeed in learning the long delay. However, the SpikingGamma SNN can precisely target delay kernels at the timesteps where signal increases should occur, allowing it to realize the delay with minimum spikes (two), whereas BPTT with SGs reinforces a broader band of kernels, an effect that may reflect diffuse temporal credit assignment and the smooth surrogate derivative spreading gradients across nearby timesteps, consistent with [38].

Learning delayed coincidence detection. In the second task, illustrated in Figure 5, the goal is to learn delayed coincidence detection. Here, a sound arriving from a particular direction is represented by a distinct pair of spike times across two input channels, where the class is determined by the relative time-difference between the two spikes. This mimics the mechanism used by barn owls to localize prey based on subtle interaural time differences in the arrival time of spikes as a function of azimuth of the prey relative to the owl's head direction [39].

The input consists of a spike pair (left (L) spike time, right (R) spike time) as defined by the class, plus jitter drawn from $\mathrm{Uniform}(0, 2)$. For the four classes, the pairs are: (4, 60), (4, 20), (20, 4), and

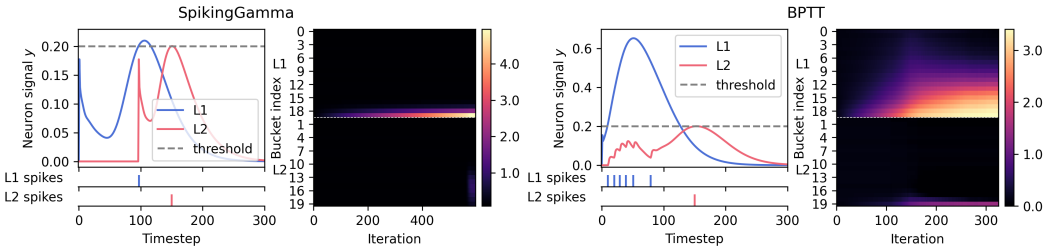


Figure 4: Comparison between SpikingGamma (left) and BPTT with SGs (right), following the backward path from (g) to (f) to (c) in Figure 2. Left shows the neuron dynamics after converging, and right the bucket-weight evolution during training. Full details of the experiment setup in Appendix D2.

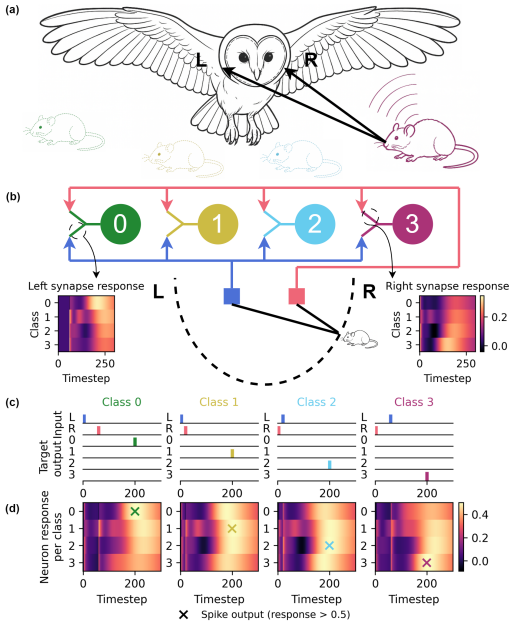


Figure 5: **(a)** An owl detects sound waves emitted by a target, such as moving prey, and these waves reach each ear at slightly different times. This difference in travel distance creates an interaural time difference, which the owl uses to accurately determine the prey's position in the horizontal plane. **(b)** The architecture of the coincidence detection network, shown activated for a prey in the right-most (brown) position. The weighted bucket activations are shown for the left and right synapses of each class, with the left being more delayed. **(c)** Input and target spikes for all four classes. (**d**) Output neuron response to the correct class input as given in (c). Output spike (colored cross) is emitted at about $t = 200$ in the correct class. Summing up the synaptic responses from (b) would result in Class 3 activation.

Table 1: Comparison of online SNN training methods. Results are reported as mean ± std; primary values are final test accuracies, with peak test accuracies during training in parentheses where available. Architectures are given as [hidden neurons × hidden layers] for fully connected networks; more complex architectures are detailed in Appendix D.2.1. All results are from original papers except where noted and SSC for ES-D-RTRL, which is our reproduction. Dashes denote unavailable values. Offline BPTT SG SoTa given for reference.

Task	Frames	Method	Architecture	Accuracy ± std (peak)
DVS	1000	FPTT	2-layer FC, LTC	- (91.28 ± 1.05)
Gesture	500	DECOLLE	4-layer CNN, CUBA	- (95.54 ± 0.16)
	2000	SpikingGamma	2-layer FC	**92.30 ± 0.18 (93.81 ± 0.18)**
	2000	SpikingGamma	4-layer CNN	**95.08 ± 0.93 (96.08 ± 0.18)**
	20	*BPTT PLIF [40]*	*7-layer CNN*	*- (97.6)*
SHD	50	OTTT [25]	$[512 \times 3]$, LIF	71.2 ± 0.8 (-)
	50	OSTL [25]	$[512 \times 3]$, LIF	70.6 ± 0.7 (-)
	50	OTPE [25]	$[512 \times 3]$, LIF	75.4 ± 0.5 (-)
	100	DECOLLE [41]	$[450]$, ALIF	62.01 ± 0.61 (-)
	100	ES-D-RTRL	$[1024 \times 3]$, RadLIF	- (93.35 ± 0.36)
	100	e-prop [41]	$[450]$, ALIF	80.79 ± 0.39 (-)
	250	SpikingGamma	$[256 \times 3]$	**92.81 ± 0.68 (93.55 ± 0.48)**
	250	*BPTT SE-adLIF [42]*	$[360 \times 2]$	*93.79 ± 0.76*
SSC	100	ES-D-RTRL	$[1024 \times 3]$, RadLIF	- (66.63 ± 0.53)
	250	SpikingGamma	$[512 \times 3]$	**75.63 ± 0.44 (75.91 ± 0.54)**
	300	*BPTT SE-adLIF [42]*	$[720 \times 2]$	*80.44 ± 0.26*

(60, 4). The target for each class is a single output spike at time 200 from the corresponding class neuron, with other output neurons remaining silent (see Figure 5b). The loss is computed the same way as for learning delays. Both the left and right input have their own unique synaptic connections to all four output neurons (so 8 connections in total), thus each output neuron has two synaptic inputs (see Figure 5a). Each of these synapses has 25 buckets, where the bucket weights are trainable and applied on synapse level. Within each neuron, the synaptic responses of both inputs are summed up.

Figure 5c shows the synaptic responses after training. Effectively, each synapse learns a specific approximate delay such that for a specific pair of input spike times, their sum (L+R) results in a spike for the right class at the right time (see Figure 5d). Accuracy (per time-to-first-spike coding) is 100% after training, with minimal spiking (maximal efficiency). This demonstration suggests that the SpikingGamma SNN is well suited for temporally structured classification tasks and aligned with biological neural coincidence detection.

4.2 Scaling through space and time

To demonstrate scaling to larger networks, problems and fine temporal resolution, we evaluate the SpikingGamma model on three widely used benchmarking datasets. We use DVS Gesture [43], SHD, and SSC [44]; while SHD and SSC can be handled with relatively shallow fully connected network topologies, they remain challenging because of the need to capture rich temporal dynamics. In contrast, DVS Gesture is less demanding temporally but has greater spatial complexity, for which we adopt both a shallow fully-connected and a deeper convolutional architecture. For SHD, we also include an ablation study to examine the effect of normalization, ReLU, and having a power-law shaped neuron response, and a study into the computational complexity and resource usage, both in Appendix C. Details of the experimental setup are provided in Appendix D.

As noted, online training methods seek temporal credit assignment without BPTT or RTRL. However, because existing approaches approximate temporal gradients through traces or regularizers, they may fail to capture temporal dynamics accurately. To assess their effect on accuracy and compare directly with SpikingGamma SNNs, Table 1 reports our best results alongside prior online training methods. We include approaches that either provide results for training with a large number of frames (DVS Gesture) or report any results at all (SHD). We also distinguish methods reporting peak test accuracy

during training, since this can introduce mild test set leakage [29]. For reproducibility, we report both final-epoch and peak test accuracy.

As shown in Table 1, SpikingGamma SNNs achieve competitive test accuracy across datasets at superior temporal resolution. This is evident for FPTT and DECOLLE on DVS Gesture, and even more so for SHD, where most other online methods struggle. While DECOLLE performs well on DVS Gesture, it shows limited accuracy on SHD. For SHD, ES-D-RTRL is competitive provided just 100 input frames are used. Compared to offline BPTT, SpikingGamma closes much of the accuracy gap while preserving online per-timestep training and stable behavior under much finer discretization. Finally, when examining the temporal dynamics learned by SpikingGamma neurons we find they qualitatively resemble biological time-cell responses (details in Appendix C.5).

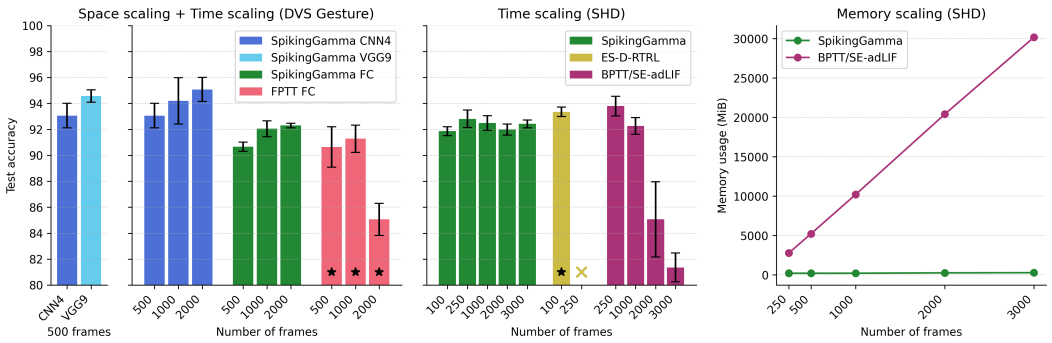


Figure 6: Effect of increasing the number of frames (time scaling) and model size (space scaling) on test accuracy and memory usage. The leftmost panel shows space scaling by comparing a 4-layer CNN (CNN4) with a VGG9 network at 500 frames on DVS Gesture. The remaining panels illustrate how increasing frames affects test accuracy and memory. The "×" denotes chance-level performance. Bars with an asterisk (*) indicate peak test accuracy during training; others show final test accuracy after training. Detailed results, including ES-D-RTRL performance between 100 and 250 frames, are in Appendix C.1. Peak SpikingGamma results appear in Appendix C.2. Additionally, [26] reports that LSTM, ASRNN, and LTC neurons trained with BPTT on DVS Gesture show degrading test accuracy as frame count increases, notably between 200 and 500 frames.

Temporal precision Increasing the temporal resolution results in longer sequences, which by itself has a strongly detrimental effect on BPTT performance [26]. Moreover, BPTT/RTRL approximations often perform well only after extensive hyperparameter tuning [45] and studies of achievable temporal precision are lacking except for comparatively easy datasets such as DVS Gesture [12, 26], which is known to contain limited temporal structure [46]. Here, we perform a more extensive study by evaluating performance at finer temporal resolutions (i.e., more frames of smaller time bins, with unchanged sample duration; t indexes timesteps, not physical time) for both DVS Gesture and SHD. To keep the temporal dynamics consistent, we scale the bucket transfer rate by the same factor as the timestep change. To further broaden the comparison, we additionally train with a larger number of frames using publicly available implementations of FPTT, ES-D-RTRL, and SE-adLIF.

Figure 6 shows that as the number of frames increases, SpikingGamma maintains stable accuracy on DVS Gesture, even improving slightly, while FPTT degrades sharply. On SHD, ES-D-RTRL collapses, and SE-adLIF also deteriorates substantially, whereas SpikingGamma maintains performance. Memory usage remains low and theoretically constant over time, increasing slightly in practice due to software overhead, while BPTT-based methods scale linearly with the number of frames. The leftmost panel further shows that SpikingGamma scales effectively to larger and deeper models (from CNN4 to VGG9), resulting in improved accuracy.

Sparsity SpikingGamma SNNs can theoretically achieve efficient spike-coding, as shown by the coincidence detection task, by keeping dynamics mostly subthreshold and using spike-time codes rather than relying solely on rate codes. This is similar to how delays or adaptive neurons enable efficient neural codes. When applying gain loss on SHD, we reach a spike density comparable to competitive methods (5 to 6 spikes/neuron/sample [47] with little accuracy loss), as shown in Figure C4 (Appendix). However, because SHD is essentially rate-coded [44], its potential for sparsity is inherently limited [48]. For datasets with richer temporal structures, we expect larger gains in sparsity.

5 Discussion and Conclusion

We introduce the SpikingGamma model, an online training approach for SNNs that enables direct error backpropagation through the sigma-delta step using a straight-through estimator, without needing surrogate gradients or temporal unrolling. The method thereby avoids the temporal credit assignment approximations inherent to existing online methods. This allows SpikingGamma to learn at fine temporal resolutions while preserving the capacity to capture complex temporal dependencies across multiple timescales. By maintaining information through subthreshold dynamics, the model also encourages sparse spike coding, improving compatibility with the communication constraints of neuromorphic hardware.

Further sparsity may be achievable by exploiting SpikingGamma's ability to learn complex temporal feature detectors at both the neuron and synapse level. However, many large-scale benchmarks are defined by rate-code models, making it difficult to evaluate and benefit from sparse temporal coding strategies [48]. Benchmarks designed around temporal coding could therefore better reveal the efficiency gains enabled by SpikingGamma.

At the same time, SpikingGamma introduces additional state and computation through its bucket representation, with costs scaling with the number of buckets K. Moreover, while SpikingGamma avoids BPTT in feedforward architectures, BPTT-like training remains necessary for networks with recurrent connections or other feedback pathways. Finally, although our software results support temporal precision and stable online training, the expected efficiency benefits depend on hardware support for local bucket dynamics and still require validation on neuromorphic substrates. With modern AI dominated by feedforward architectures such as Transformers, analogous extensions of SpikingGamma may offer a path toward scalable deep SNNs for sequence learning.

Acknowledgments

Roel Koopman is funded by the Dutch Research Council under Grant agreement KICH1.ST04.22.021. Sander Bohté is supported by NWO-NWA grant NWA.1292.19.298. Sebastian Otte acknowledges support from the Alexander von Humboldt Foundation (Feodor Lynen Research Fellowship) during the early phase of this work.

References

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

[2] Yuhang Li, Ruokai Yin, Youngeun Kim, and Priyadarshini Panda. Efficient human activity recognition with spatio-temporal spiking neural networks. *Frontiers in Neuroscience*, 17: 1233037, 2023.

[3] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Are snns really more energy-efficient than anns? an in-depth hardware-aware study. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(3):731–741, 2022.

[4] Fernando S Martínez, Jordi Casas-Roma, Laia Subirats, and Raúl Parada. Spiking neural networks for autonomous driving: A review. *Engineering Applications of Artificial Intelligence*, 138:109415, 2024.

[5] Suwhan Baek and Jaewon Lee. Snn and sound: a comprehensive review of spiking neural networks in sound. *Biomedical Engineering Letters*, 14(5):981–991, 2024.

[6] Jesse J Hagenaars, Federico Paredes-Vallés, Sander M Bohté, and Guido CHE De Croon. Evolved neuromorphic control for high speed divergence-based landings of mavs. *IEEE Robotics and Automation Letters*, 5(4):6239–6246, 2020.

[7] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[8] Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Learning delays in spiking neural networks using dilated convolutions with learnable spacings. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[10] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

[11] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

[12] Chaoming Wang, Xingsi Dong, Zilong Ji, Jiedong Jiang, Xiao Liu, and Si Wu. Brainscale, enabling scalable online learning in spiking neural networks. *bioRxiv*, pages 2024–09, 2024.

[13] Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022.

[14] Jonghyun Ko, Dongseok Kwon, Joon Hwang, Kyu-Ho Lee, Seongbin Oh, Jeonghyun Kim, Jiseong Im, Ryun-Han Koo, Jae-Joon Kim, and Jong-Ho Lee. Snnsim: Investigation and optimization of large-scale analog spiking neural networks based on flash memory devices. *Advanced Intelligent Systems*, 6(4):2300456, 2024.

[15] Roel Koopman, Amirreza Yousefzadeh, Mahyar Shahsavari, Guangzhi Tang, and Manolis Sifalakis. Exploring the limitations of layer synchronization in spiking neural networks. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

[16] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, pages 5189–5200. PMLR, 2021.

[17] Bert De Vries and Jose C Principe. The gamma model—a new neural model for temporal processing. *Neural networks*, 5(4):565–576, 1992.

[18] Ilya Sutskever and Geoffrey Hinton. Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243, 2010.

[19] Sander M Bohte. Error-backpropagation in networks of fractionally predictive spiking neurons. In *International conference on artificial neural networks*, pages 60–68. Springer, 2011.

[20] Jaldert Rombouts and Sander Bohte. Fractionally predictive spiking neurons. *Advances in Neural Information Processing Systems*, 23, 2010.

[21] Young C Yoon. Lif and simplified srm neurons encode signals into spikes via a form of asynchronous pulse sigma–delta modulation. *IEEE transactions on neural networks and learning systems*, 28(5):1192–1205, 2016.

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

[23] Chenlin Zhou, Han Zhang, Liutao Yu, Yumin Ye, Zhaokun Zhou, Liwei Huang, Zhengyu Ma, Xiaopeng Fan, Huihui Zhou, and Yonghong Tian. Direct training high-performance deep spiking neural networks: a review of theories and methods. *Frontiers in Neuroscience*, 18: 1383844, 2024.

[24] Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8894–8908, 2022.

[25] Thomas M Summe, Clemens JS Schaefer, and Siddharth Joshi. Estimating post-synaptic effects for online training of feed-forward snns. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 264–271. IEEE, 2024.

[26] Bojian Yin, Federico Corradi, and Sander Bohté. Accurate online training of dynamical spiking neural networks through forward propagation through time. *Nature Machine Intelligence*, 5: 1–10, 05 2023. doi: 10.1038/s42256-023-00650-4.

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

[28] Benjamin Ellenberger, Paul Haider, Federico Benitez, Jakob Jordan, Kevin Max, Ismael Jaras, Laura Kriener, and Mihai A Petrovici. Backpropagation through space, time and the brain. *Nature Communications*, 17(1):66, 2025.

[29] Balázs Mészáros, James C Knight, and Thomas Nowotny. Efficient event-based delay learning in spiking neural networks. *Nature Communications*, 16(1):10422, 2025.

[30] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12:1, 2016.

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

[32] Yue Liu, Zoran Tiganj, Michael E Hasselmo, and Marc W Howard. A neural microcircuit model for a scalable scale-invariant representation of time. *Hippocampus*, 29(3):260–274, 2019.

[33] Patrick J Drew and Larry F Abbott. Models and properties of power-law adaptation in neural systems. *Journal of neurophysiology*, 96(2):826–833, 2006.

[34] Elliot A Ludvig, Richard S Sutton, and E James Kehoe. Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural computation*, 20(12): 3034–3054, 2008.

[35] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[36] Davide Zambrano, Roeland Nusselder, H Steven Scholte, and Sander M Bohté. Sparse computation in adaptive spiking neural networks. *Frontiers in neuroscience*, 12:987, 2019.

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

[38] Yang Li, Feifei Zhao, Dongcheng Zhao, and Yi Zeng. Directly training temporal spiking neural network with sparse surrogate gradient. *Neural Networks*, 179:106499, 2024.

[39] CE Carr and M Konishi. A circuit for detection of interaural time differences in the brain stem of the barn owl. *Journal of Neuroscience*, 10(10):3227–3246, 1990.

[40] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2661–2671, 2021.

[41] Fernando M Quintana, Fernando Perez-Peña, Pedro L Galindo, Emre O Neftci, Elisabetta Chicca, and Lyes Khacef. Etlp: event-based three-factor local plasticity for online learning with neuromorphic hardware. *Neuromorphic Computing and Engineering*, 4(3):034006, 2024.

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

[43] 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.

[44] 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.

[45] Lang Xue, HanWen Liu, Jing Wang, and Hong Qu. Reasonable gradients for online training algorithms in spiking neural networks. In *ECAI 2024*, pages 2709–2716. IOS Press, 2024.

[46] Xinyi Chen, Chenxiang Ma, Yujie Wu, Kay Chen Tan, and Jibin Wu. Neuromorphic sequential arena: A benchmark for neuromorphic temporal processing. In *The 34th International Joint Conference on Artificial Intelligence (IJCAI'25)*, 2025.

[47] 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.

[48] Ziqiao Yu, Pengfei Sun, and Dan FM Goodman. Beyond rate coding: Surrogate gradients enable spike timing learning in spiking neural networks. *arXiv preprint arXiv:2507.16043*, 2025.

[49] Akihiro Shimbo, Ei-Ichi Izawa, and Shigeyoshi Fujisawa. Scalable representation of time in the hippocampus. *Science Advances*, 7(6):eabd7013, 2021. Published at `https://www.science.org/doi/10.1126/sciadv.abd7013`, licensed under CC BY-NC 4.0.

[50] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[52] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661. IEEE, 2016.

[53] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in neuroscience*, 15:773954, 2021.

[54] Manu V Nair and Giacomo Indiveri. An ultra-low power sigma-delta neuron circuit. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

A Pseudocode

Algorithm 1 SpikingGamma neuron forward pass for a single timestep from $t-1$ to t

Require: current bucket state of previous layer $\hat{\mathbf{y}}_{\text{in}}[t] \in \mathbb{R}^{N_{\text{in}} \times K}$, previous bucket state of current layer $\hat{\mathbf{y}}_{\text{out}}[t-1] \in \mathbb{R}^{N_{\text{out}} \times K}$, synaptic weights $\mathbf{W} \in \mathbb{R}^{N_{\text{in}} \times N_{\text{out}}}$, bucket weights $\mathbf{V} \in \mathbb{R}^{N_{\text{out}} \times K}$, spike threshold $\vartheta[t] \in \mathbb{R}_{>0}$, shared bucket transfer rates $\boldsymbol{\alpha} \in \mathbb{R}^{K}$

Ensure: updated bucket state $\hat{\mathbf{y}}_{\text{out}}[t] \in \mathbb{R}^{N_{\text{out}} \times K}$, output spikes $\mathbf{s}[t] \in \{0,1\}^{N_{\text{out}}}$

$\mathbf{x}[t] \leftarrow \sum_k \left((\hat{\mathbf{y}}_{\text{in}}[t]^T \mathbf{W})^T \circ \mathbf{V} \right)_{:,k}$
$\mathbf{y}[t] \leftarrow \text{ReLU}(\mathbf{x}[t])$
$\mathbf{z}[t] \leftarrow \mathbf{y}[t] - \sum_k \hat{\mathbf{y}}_{\text{out}}[t-1]_{:,k}$
$\mathbf{s}[t] \leftarrow \mathbf{z}[t] > \vartheta[t]$
$\hat{\mathbf{y}}_{\text{out}}[t]_{:,0} \leftarrow \alpha_0 \hat{\mathbf{y}}_{\text{out}}[t-1]_{:,0} + 2\vartheta[t]\mathbf{s}[t]$
for $k = 1$ to $K-1$ **do**
  $\hat{\mathbf{y}}_{\text{out}}[t]_{:,k} \leftarrow \alpha_k \hat{\mathbf{y}}_{\text{out}}[t-1]_{:,k} + (1-\alpha_k)\hat{\mathbf{y}}_{\text{out}}[t-1]_{:,k-1}$
end for

B SpikingGamma SSM interpretation

SpikingGamma can be interpreted within the State-Space Model (SSM) framework. For layer ℓ, we stack all bucket states into $\mathbf{h}^\ell(t) = \left[\hat{y}_i^{k,\ell}(t)\right]_{i,k}$, yielding

$$\mathbf{h}^\ell(t) = \mathbf{A}\,\mathbf{h}^\ell(t-1) + \mathbf{B}\,\mathbf{s}^\ell(t), \qquad \mathbf{D} = 0 \tag{B1}$$

where $\mathbf{A}$ implements the Gamma bucket dynamics and $\mathbf{B}$ injects spike events into the first bucket (cf. Eq. (5), ignoring the adaptive threshold). The model has two readouts,

$$\hat{\mathbf{y}}^\ell(t) = \mathbf{C}_{\text{out}}\,\mathbf{h}^\ell(t), \qquad \mathbf{y}^\ell(t) = \phi\big(\mathbf{C}_{\text{in}}\,\mathbf{h}^{\ell-1}(t)\big) \tag{B2}$$

where $\mathbf{C}_{\text{out}}$ just sums buckets per neuron, while $\mathbf{C}_{\text{in}}$ also applies w_{ij} and v_j^k, and $\phi(\cdot) = \text{ReLU}(\cdot)$.

The state is driven by $\mathbf{s}^\ell(t)$, a quantized representation of the continuous drive $\mathbf{y}^\ell(t)$. Spikes are generated by elementwise thresholding the prediction error:

$$\mathbf{s}^\ell(t) = (\mathbf{y}^\ell(t) - \hat{\mathbf{y}}^\ell(t-1) > \vartheta(t)). \tag{B3}$$

Each spike thus both corrects the local reconstruction via $\hat{\mathbf{y}}^\ell$ and drives the next layer via $\mathbf{y}^{\ell+1}$.

C Extended results

C.1 Performance comparison when increasing number of frames

For the reproduced works, we examined how time constants were applied and whether they required adjustment when the number of frames (and thus the timestep size) changed. For FPTT, ES-D-RTRL, and SE-adLIF, the time constants are trainable. We observed no performance improvements from modifying their bounds or initialization.

Table C1: Test accuracy (peak for FPTT, final for SpikingGamma) for models trained on DVS Gesture. *Reproduced from `https://github.com/byin-cwi/sFPTT`. No annotation means the number is from the original work.

Frames	SpikingGamma (CNN4)	SpikingGamma (VGG9	SpikingGamma (FC)	FPTT
500	93.06 ± 0.94	94.57 ± 0.47	90.66 ± 0.36	90.64 ± 1.56
1000	94.19 ± 1.79	-	92.05 ± 0.62	91.28 ± 1.05
2000	95.08 ± 0.93	-	92.30 ± 0.18	85.07 ± 1.24*

Table C2: Test accuracy (peak for ES-D-RTRL, final for SpikingGamma and BPTT/SE-adLIF) for models trained on SHD. *Reproduced from `https://github.com/chaobrain/brainscale-exp-for-snns` (commit 11720f5). The number of epochs was reduced from 100 to 30. For 250+ frames, ES-D-RTRL no longer converges, see Figure C1 for transition and collapse from 100 to 300 frames. Therefore, we did not reproduce for 1000 or more frames. **Reproduced from `https://github.com/IGITUGraz/SE-adlif`. No annotation means the number is from the original work.

Frames	**SpikingGamma**	**ES-D-RTRL**	**BPTT/SE-adLIF**
100	91.86 ± 0.34	93.35 ± 0.36	-
250	92.81 ± 0.68	4.5*	93.79 ± 0.76
1000	92.49 ± 0.57	-	92.26 ± 0.65**
2000	91.99 ± 0.43	-	85.06 ± 2.89**
3000	92.42 ± 0.31	-	81.36 ± 1.11**

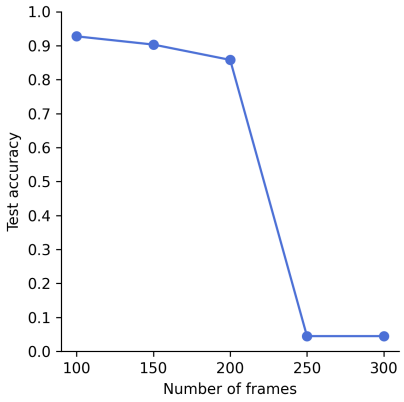


Figure C1: Test accuracy (best over all epochs) for ES-D-RTRL (reproduced) trained on SHD from 100 to 300 frames. The accuracy for 250 frames was verified over 3 independent training sessions.

C.2 Peak test accuracy during training

A common practice in prior work is to select the peak test performance observed during training, effectively using the test set to guide model selection. This introduces test set information into the training process and can lead to fitting on the test set, thereby overestimating generalization performance. To avoid this issue, we did not adopt this practice. However, to ensure a fair comparison with related works that did use this strategy, in Table C3 we additionally report the peak test performance achieved over the course of training (with the number of epochs increased to 30 for SHD/SSC and to 150 for DVS Gesture). As expected, these numbers are slightly higher, but the overall trends remain consistent.

Table C3: Peak test accuracies achieved during training.

Task	Frames	Accuracy ± std
DVS Gesture (CNN4)	500	95.58 ± 0.18
	1000	96.21 ± 0.31
	2000	96.08 ± 0.18
DVS Gesture (VGG9)	500	95.71 ± 0.36
DVS Gesture (FC)	500	92.80 ± 0.62
	1000	93.31 ± 0.18
	2000	93.81 ± 0.18
SHD	100	92.64 ± 0.27
	250	93.55 ± 0.48
	1000	93.18 ± 0.34
	2000	92.61 ± 0.39
	3000	93.31 ± 0.24
SSC	250	75.91 ± 0.54

C.2.1 Clarifying reported test accuracy for DECOLLE

Since the original work on DECOLLE in [11] was ambiguous about whether it reported peak or final epoch test accuracy, we reproduced their results using the original code (`https://github.com/nmi-lab/decolle-public`) to clarify this point. Across four independent runs, we obtained a peak test accuracy of 95.70 ± 0.47 and a final epoch test accuracy of 94.19 ± 0.94. The peak accuracy aligns much more closely with the reported value of 95.54 ± 0.16, indicating that the original paper most likely reported peak test accuracy rather than final epoch accuracy.

C.3 Effect of introducing a variable threshold

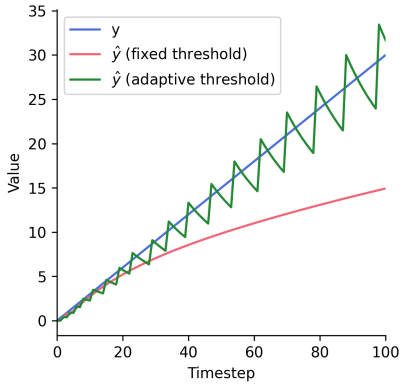


Figure C2: A demonstration showing that, without adaptive thresholding, $\hat{y}$ is unable to accurately estimate higher values of y, whereas the use of adaptive thresholding enables $\hat{y}$ to effectively estimate these higher values.

C.4 ES-D-RTRL result for SSC

The SSC results are based on reproduced work from `https://github.com/chaobrain/brainscale-exp-for-snns` (commit 11720f5). We ran 4 training sessions with the same hyperparameters as they used for their best SHD results, and then reported the mean and standard deviation of the peak epoch of each run.

C.5 Alignment with biological time-cell dynamics

To examine whether neurons in our model show time-cell-like dynamics, we analyzed the responses of neurons in a single layer to a single spike input after training on SHD. Comparing these neuronal responses to recorded biological time-cell activity revealed similar population dynamics. In both cases, the units activate sequentially over time with gradually decaying and temporally broadened responses, forming a characteristic power-law-like pattern (Figure C3). This suggests that our model captures temporal information in a manner consistent with biological neural representations of time.

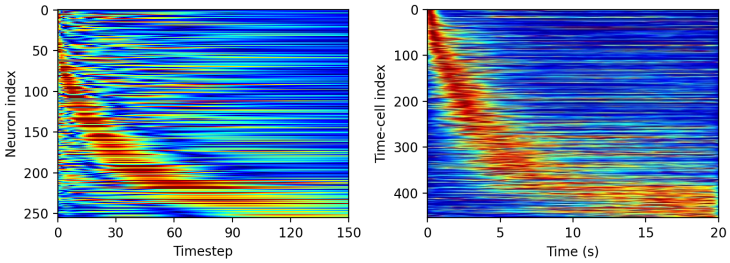


Figure C3: Comparison of neuron responses from our method with recorded rat time-cell activity. On the left: Neuron response for input at $t = 0$ for neurons from the second hidden layer of the best performing model on SHD. Neuron indices sorted by peak activity timestep. On the right: Firing patterns of rats' time-cells during a temporal bisection task. Adapted from [49].

C.6 Ablation study on SHD

To gather more insight into the effects of normalization, activation, and hyperparameters, we performed an ablation study on SHD. We used the same experimental setup as for our best-performing model, but introduced the modifications listed in Tables C4 and C5.

During our experiments, the neuron signal was first normalized using layer normalization and then rectified. As shown in Table C4, removing the ReLU activation used to compute the neuron signal y leads to a clear drop in performance. This is expected, as negative neuron signal values cannot be represented by spikes in our method, resulting in a mismatch between the signal y and its estimate $\hat{y}$.

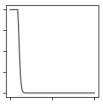
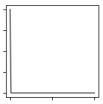
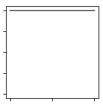
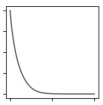
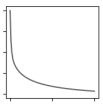
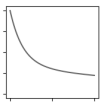
Additionally, altering or removing this normalization can have substantial impact on accuracy. RMS normalization [50] performs similarly to layer normalization, which is expected since both normalize using the same underlying statistics. In contrast, batch normalization [51] completely fails. This aligns with previous findings that batch normalization is difficult to apply to the hidden-to-hidden transitions of RNNs [52]. Batch normalization through time [53] was designed to better capture temporal characteristics than standard batch normalization, and our results confirm this, though its performance still lags behind layer and RMS normalization. Finally, omitting normalization altogether yields mixed results: when bucket weights are assigned to neurons, performance is poor, but assigning them to synapses improves results, though still with much higher variance. In summary, properly normalizing neuron signal dynamics is essential for stable training and good performance when scaling to a more complex task like SHD.

Table C4: Ablation study results for SHD.

Ablated	**Accuracy ± std**
No normalization, bucket weights on neuron	50.83 ± 9.10
No normalization, bucket weights on synapse	88.31 ± 3.15
RMS normalization	92.54 ± 0.63
Batch normalization	16.59 ± 3.28
Batch normalization through time	90.57 ± 0.60
No ReLU	85.38 ± 1.37

In Section 3.3, we introduced the standard method for initializing the bucket transfer rates. This approach is based on the idea that a well-chosen power-law-like shape (given by the sum of all buckets) is most effective at retaining information. We test this hypothesis by using different shapes and rates, as shown in Table C5. The results confirm that a power-law sum of responses, decaying over an appropriate time interval, yields the best performance, while all other response shapes underperform.

Table C5: Results from experiments with alternative kernel parameters and shapes. The "Power-law" shape type refers to the default initialization method as introduced in section 3.3, while "Fixed rate" refers to an alternative scheme where α_k is fixed to one value (given as "Rate") for all K kernels. The visualized neuron responses show the sum of all kernel responses, with on the x-axis 100 timesteps and on y-axis values from 0 to 1.

Shape type	**Rate / F**	**K**	**Neuron response**	**Accuracy**
Fixed rate	0.82	10		75.45 ± 1.16
Fixed rate	0.15	10		67.04 ± 1.48
Power-law	1.00	1		32.82 ± 1.09
Power-law	0.00	1		22.14 ± 4.71
Power-law	0.90	10		84.49 ± 0.88
Power-law	**0.15**	**10**		**92.81 ± 0.68**
Power-law	0.01	10		87.52 ± 0.38

We also experimented with the gain–loss constant to achieve sparsity, as shown in Figure C4. Setting the constant to 0.1 resulted in a significant increase in sparsity with little to no degradation in accuracy. Increasing the constant further continued to improve sparsity, but at the cost of progressively larger accuracy losses.

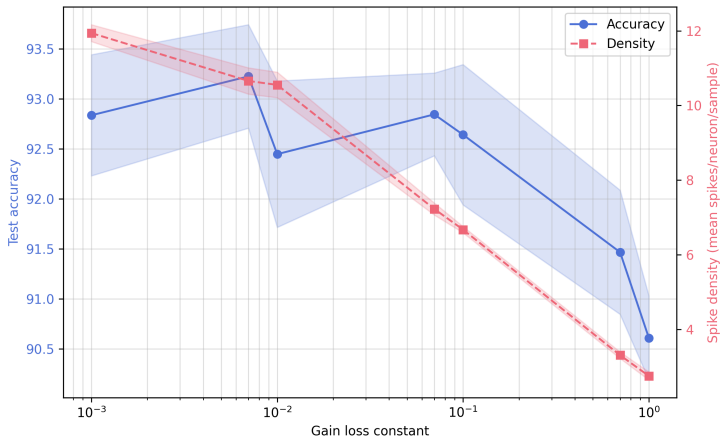


Figure C4: Effect of the gain-loss constant on accuracy and spike density (shown as mean ± std). The spike density is computed using Eq. (D3).

C.7 Computational complexity

The computational cost of SNNs can be split into spike- and timestep-dependent operations. Neuronal updates occur at every timestep and require MACs, while synaptic operations are spike-driven and typically consist of cheaper ACs.

In the following table, we compare with the $\Sigma\Delta$ neuron [54] and the LIF neuron. The $\Sigma\Delta$ and LIF neuron require MACs for leakage and ACs for current accumulation. SpikingGamma follows a similar structure, but also scales with the number of buckets K. The threshold is fixed.

Table C6: Computational scaling **per layer**. A layer has N neurons, each with K buckets, and modeled for T timesteps during which there were S spikes (in/out).

Where	**Operation**	**Scaling (ACs)**	**Scaling (MACs)**
Synapse	Current addition to first input bucket	NS_{in}	–
Neuron (Gamma)	Leak over input buckets	–	TNK
	Apply bucket weights	–	TNK
	Sum weighted input buckets	$TN(K-1)$	–
Neuron ($\Sigma\Delta$)	Sum $\Sigma\Delta$ buckets	$TN(K-1)$	–
	Compute difference signal and estimate	TN	–
	Add spikes to first bucket	S_{out}	–
	Leak over $\Sigma\Delta$ buckets	–	TNK
Total (SpikingGamma)		$O(N(TK+S_{\text{in}})+S_{\text{out}})$	$O(TNK)$
Total ($\Sigma\Delta$ neuron [54])		$O(N(T+S_{\text{in}})+S_{\text{out}})$	$O(TN)$
Total (LIF neuron)		$O(NS_{\text{in}})$	$O(TN)$

With many timesteps, neuron updates dominate, making MACs more significant. Below we report ACs and MACs per SHD sample.

Table C7: Estimated computational cost per sample for SHD. The network consists of 3 layers of each 256 neurons plus an output layer of 20 neurons, with each neuron having 10 buckets. Each sample has 250 timesteps.

Where	**Operation**	**ACs**	**MACs**
Synapse	Current addition to first input bucket	1490K $\pm$ 165.7K	0
Neuron (Gamma)	Leak over input buckets	0	1970K
	Apply bucket weights	0	1970K
	Sum weighted input buckets	1773K	0
Neuron ($\Sigma\Delta$)	Sum $\Sigma\Delta$ buckets	1773K	0
	Compute difference signal and estimate	197K	0
	Add spikes to first bucket	5K $\pm$ 0.5K	0
	Leak over $\Sigma\Delta$ buckets	0	1970K
Total (SpikingGamma)		5239K $\pm$ 166.2K	5910K

We can approximately extrapolate these numbers to the $\Sigma\Delta$ neuron by dividing terms scaling with K by 10, resulting in relatively more ACs than MACs. Under standard digital assumptions, this suggests that the additional cost of SpikingGamma is dominated by timestep-dependent bucket updates. However, this should be interpreted as a rough indication, as actual energy consumption depends on hardware, precision, and memory access patterns. In mixed-signal systems, bucket operations may be significantly cheaper than MAC counts suggest.

C.8 Sigma-delta approximation quality

In Figure C5, we visualize the empirical relation between y and $\hat{y}$. Most points lie close to the identity line, showing a good overall match. A robust fit on the points with $y \geq \vartheta_0$ has a slope slightly below one. Because the threshold grows with $\hat{y}$, both the error tolerated before a spike and the size of each spike grow with $\hat{y}$, which widens the spread for larger values. Points far above the identity line occur when y drops faster than $\hat{y}$ can follow: only positive spikes are emitted, so $\hat{y}$ can only decrease through the decay of the kernel. Points far below the identity line occur when y rises quickly: at most one spike is emitted per time step, so $\hat{y}$ needs several time steps to catch up. When y is below the minimum threshold ϑ_0, no spike can be emitted, so $\hat{y}$ only contains the decaying residual of earlier spikes, or is zero.

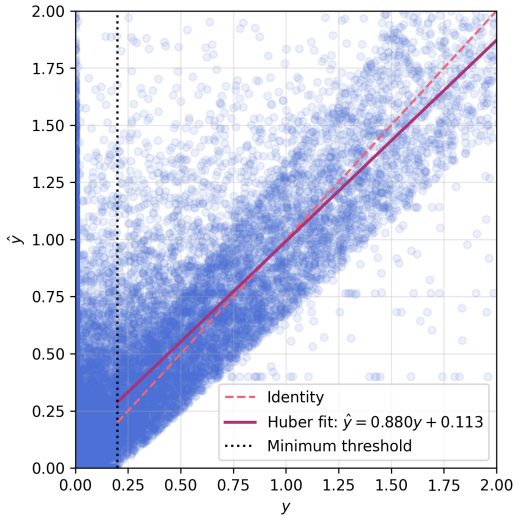


Figure C5: Empirical relationship between the activation y and its approximation $\hat{y}$ for the neurons of all three hidden layers on the SHD test set after training. Each point is one neuron at one time step of one test sample (10000 random points per layer). The Huber fit uses only points with y at or above the minimum spike threshold (dotted line). Hyperparameters as in Table D1.

D Details on the experimental setup

D.1 Datasets

The Spiking Heidelberg Digits (SHD) and Spiking Speech Commands (SSC) datasets both consist of auditory recordings. Each recording was converted into spike trains using a biologically inspired cochlear model, capturing fine-grained temporal structure in the audio [44]. The SHD dataset contains 8156 training and 2264 test samples for 20 spoken digits. The SSC dataset contains 75466 training, 9981 validation, and 20382 test samples for 35 spoken commands. We included the validation samples in the training set (and did no validation during training). For both datasets, the 700-channel cochlear outputs are downsampled to 140 channels to reduce input dimensionality.

The DVS Gesture dataset [43] contains event-based visual recordings of 11 hand and arm gestures captured using a DVS camera. It includes 1176 training and 288 test samples. The original 128×128 event frames are downsampled to 32×32 by summing up events in a 4×4 window.

To prepare the event streams from the datasets for model input, the timestamps are discretized into time-bins along the temporal axis. All events falling within the same time-bin are accumulated at their corresponding spatial indices, producing a sequence of frames. These frames are then fed to the model as inputs at successive discrete timesteps.

For computing the mean and standard deviation of the reported accuracies, we repeated the experiments with SHD and SSC 5 times, and with DVS Gesture 3 times.

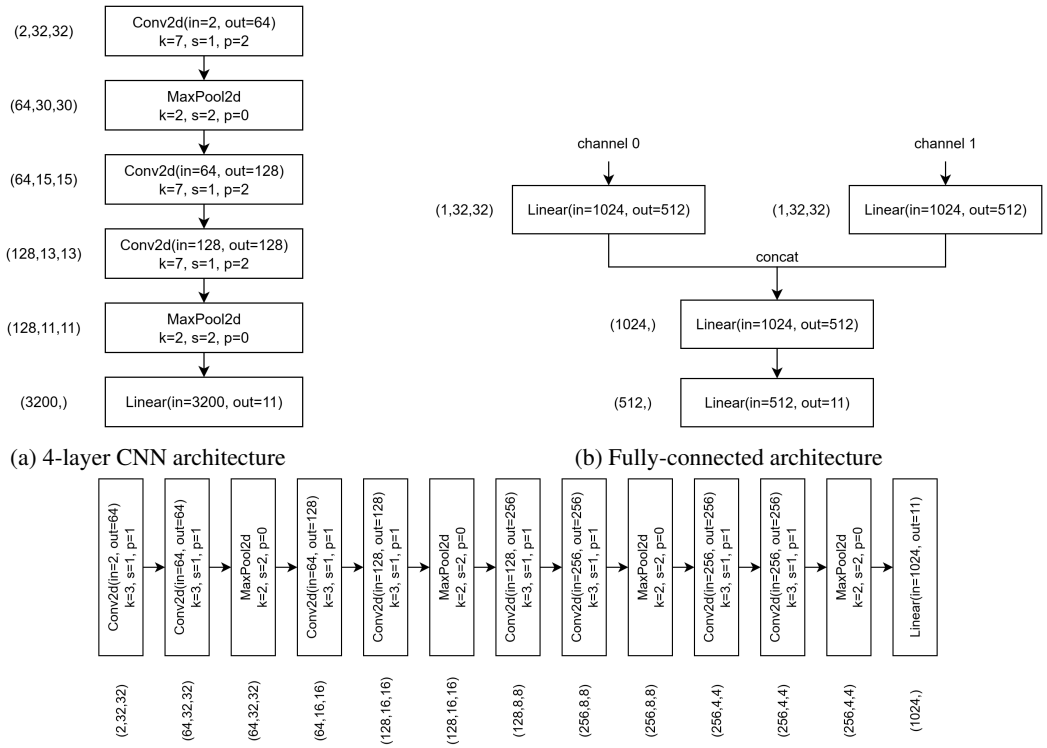
D.2 Network architectures and hyperparameters

Table D1: Network architecture and hyperparameters. The architecture is given as [neurons in hidden layers $\times$ number of hidden layers] if fully-connected. Whenever we change the number of frames / the timestep size from the values mentioned above, we scale the bucket transfer rate by the same factor as the timestep size change to keep the temporal dynamics consistent. *Times K per layer upward starting from the output towards the input.

	SHD	SSC	DVS Gesture FC/CNN4	DVS Gesture VGG9
Architecture	[256×3]	[512×3]	D.2.1	D.2.1
Timestep size	3.6 ms	3.6 ms	3 ms	12 ms
Number of frames	250	250	2000	500
Batch size	32	32	64	64
Epochs	25	25	100	70
Initial learning rate	1e-3*	1e-3*	1e-3	1e-3
Learning rate schedule	Step	Step	None	Cosine
Schedule step size	10	10		
Schedule step gamma	0.1	0.1		
Schedule η_{min}				1e-6
Schedule T_{max}				70
Minimum threshold ϑ_0	0.2	0.2	0.2	0.3
Number of buckets K	10	7	10	10
Bucket transfer rate F	0.15	0.15	0.05	0.2
Dropout	0.1	0.1	0	0
Loss function	CE	CE warm-up (D.3.1)	CE	CE

As optimizer, for experiments with SHD, SSC, and DVS Gesture, Adam is used with $\beta = (0.9, 0.999)$. For the delay and coincidence detection tasks, SGD is used. Hyperparameters were tuned by manual trial and error without a strict tuning budget or predefined search ranges.

D.2.1 DVS Gesture model architecture



(a) 4-layer CNN architecture

(b) Fully-connected architecture

(c) VGG9 architecture

Figure D1: (a) shows the 4-layer CNN architecture used for DVS Gesture (similar to what was used for DECOLLE in [11]). (b) shows the fully-connected architecture (similar to what was used for FPTT in [26]). (c) shows the VGG9 architecture used for spatial scaling. Only for VGG9, the feature extractor (i.e., all layers before the classifier) uses weight standardization (see Section D.5) and group normalization (see Section D.4) instead of layer normalization. In all figures, neuron layers (including normalization and dropout, if applicable) are omitted; they are placed after each Conv2d or Linear layer.

D.3 Performance metrics

D.3.1 CE warm-up loss

For SSC, there is a delay in sample onset. This can be problematic when training online. To improve performance, we introduce a custom loss function that trains to mute the output at the start of the sample. It interpolates between MSE to zero and CE loss using a time-dependent weight:

$$L(t) = \beta(t) \cdot L^{\text{MSE}}(\hat{y}_{\text{out}}(t), 0) + (1 - \beta(t)) \cdot L^{\text{CE}}(y_{\text{out}}(t), y_{\text{true}}(t)) \tag{D1}$$

where

$$\beta(t) = \beta_0 \cdot (\beta_{\text{decay}})^t. \tag{D2}$$

with $\beta_{\text{decay}} = 0.99$ and $\beta_0 = 1.0$.

D.3.2 Spike density

$$\text{spike density} = \frac{1}{N_{\text{samples}} \cdot N_{\text{neurons}}} \sum_{i=1}^{N_{\text{samples}}} N_{\text{spikes}}[i] \tag{D3}$$

where N_{samples} is the dataset size, N_{neurons} is the total number of neurons in the hidden layers, and $N_{\text{spikes}}[i]$ is the spike count recorded during the inference of the i-th sample.

D.4 Group normalization

Group normalization divides the C channels into G groups (chosen as the largest value in $\{32, 16, 8, 4, 2, 1\}$ such that $C \bmod G = 0$) and normalizes the neuron signal before rectification (x) based on group-wise statistics: $x_{\text{norm}} = \frac{x - \mathrm{E}_g[x]}{\sqrt{\mathrm{Var}_g[x] + \epsilon}} \cdot \gamma + \beta$, with γ the gain, β the bias (both trainable), and $\mathrm{E}_g[x]$ and $\mathrm{Var}_g[x]$ respectively the mean and variance of the neuron signal computed over all neurons within the same group.

D.5 Weight standardization

Weight standardization normalizes each output neuron's weights to zero mean and controlled variance. For $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, each row W_i is transformed as

$$\hat{W}_i = \frac{W_i - \mathrm{E}[W_i]}{\sqrt{\mathrm{Var}[W_i] \cdot d_{\text{in}} + \epsilon}}, \tag{D4}$$

where $\mathrm{E}[W_i]$ and $\mathrm{Var}[W_i]$ denote the mean and variance of the weights of neuron i.

D.6 Computer resources

Our experiments were done on a workstation equipped with 256 GB of memory, an NVIDIA RTX A6000 GPU with 49 GB of VRAM, and an Intel Xeon Bronze 3206R CPU.

D.7 Experimental setup delay experiment

Table D2: Hyperparameters for the delay experiment.

Hyperparameter	**Value**
Threshold	0.2
Batch size	1
Epochs	Until converged
Optimizer	SGD
Learning rate	10^{-1}
Synaptic weight (frozen)	5.0
Synaptic weight bias (frozen)	0.0
Bucket weight initialization	$\mathcal{N}(\mu = 0.1,\, \sigma = 0.01)$; fixed seed
Number of buckets K	20
Bucket transfer rate F	0.3
Loss	MSE
Normalization	None
Adaptive threshold	Disabled
Surrogate (BPTT only)	arctan ($\alpha = 2$)

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: Our results support our claims.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The limitations are discussed in the results and discussion sections.

   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: No theoretical results.

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: Besides the methods that describe the general idea of our approach, we included a detailed overview of the hyperparameters in the appendix. Furthermore, we included the same code as we used for our experiments. This should enable full reproducibility.

   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: At submission time we provide anonymized code for reproducibility. After acceptance we will release a package that enables anyone to easily get started with SpikingGamma.

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: Full details are provided in the main text and 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: All experiments were repeated multiple times and errors bars / standard deviations were included.

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: Computer resources used are described in the appendix. These can cover all of the experiments.

   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: No remarks.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Potential impact is discussed in the introduction.

    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: Not high risk.

    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: Done.

    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: [Yes]

    Justification: Code includes documentation.

    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: No crowdsourcing.

    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: 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: Core method development in this research does not involve LLMs as any important, original, or non-standard components.

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.