# FiTS: Interpretable Spiking Neurons via Frequency Selectivity and Temporal Shaping

**Jongmin Choi** **Joon Son Chung**
Korea Advanced Institute of Science and Technology (KAIST)

## Abstract

Spiking Neural Networks (SNNs) are a promising framework for event-driven temporal processing. Prior work has improved temporal modeling through richer neuron dynamics and network-level mechanisms such as recurrence and delays, but it remains unclear how individual spiking neurons should specialize within a network. In this work, we introduce **FiTS**, a spiking neuron that factorizes temporal computation within each neuron into Frequency Selectivity (FS) and Temporal Shaping (TS). The FS module parameterizes each neuron's target frequency as the maximizer of its subthreshold magnitude response, while the TS module reshapes when frequency components contribute to membrane voltage accumulation through group-delay modulation. On auditory benchmarks where frequency selectivity and timing are central to the input structure, FiTS consistently improves over a plain Leaky Integrate-and-Fire (LIF) baseline in simple feedforward SNNs without recurrence or network-level delays, while remaining competitive with strong temporal SNN baselines. Beyond accuracy, the learned target frequencies and group-delay shifts provide interpretable neuron-level summaries of the frequency and timing organization learned within the network. Our code is available at `https://github.com/kaistmm/FiTS`.

## 1 Introduction

Spiking Neural Networks (SNNs) are widely regarded as the third generation of neural networks [28]. They represent information through spike trains over time, and their event-driven sparse computation makes them a promising framework for efficient processing of temporally structured data such as audio and event streams, particularly on neuromorphic hardware [7, 14, 32]. Although recent work has substantially improved SNN performance on temporal tasks [2, 5, 11, 16, 21, 30], it remains unclear how to parameterize the temporal role of an individual spiking neuron within a network.

Prior work has improved temporal modeling along two main directions. At the neuron level, adaptive and resonant dynamics enrich temporal responses [2, 11, 20, 21, 23, 45]. At the network level, recurrence, learnable synaptic delays, and broader architectural inductive biases strengthen temporal representations [4, 8, 16, 30]. These advances are effective, but they often leave each neuron's frequency preference and response timing implicit in learned coefficients or network mechanisms.

In this work, we introduce FiTS, a spiking neuron that makes these roles explicit by factorizing neuron-level temporal computation into Frequency Selectivity (FS) and Temporal Shaping (TS). Motivated by intrinsic neuronal resonance, where neurons respond selectively to inputs at preferred frequencies [22], the FS module parameterizes each neuron's target frequency as the maximizer of its subthreshold magnitude response. The TS module uses group-delay modulation within the neuron to reshape when frequency components contribute to pre-spike membrane voltage accumulation. After training, these parameters can be read as parameter-level summaries of the learned frequency and timing organization. Our main contributions are summarized as follows.

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

- We introduce the FS module, which maps a response-level target frequency to the adaptation strength through a closed-form inverse, enabling frequency-domain initialization, learning, and post-training interpretation within the same coordinate.
- We introduce the TS module, which reshapes pre-spike membrane voltage accumulation through group-delay modulation. Unlike network-level delays that change when emitted spikes reach downstream neurons, the TS module controls when frequency-specific input responses contribute to spike generation within a neuron.
- We show that our FiTS neuron consistently improves simple feedforward SNNs across auditory benchmarks and yields learned target frequencies and group-delay shifts that summarize the network's learned organization of frequency and timing information.

## 2 Related Work

Early work on training deep SNNs addressed the differentiability problem caused by spike generation and reset using surrogate gradients [3, 34, 42]. Subsequent methods improved temporal learning by reshaping training objectives, simplifying backpropagation through time, or facilitating temporal gradient propagation [9, 15, 29, 38]. Neuron-pathway designs also improve long-range credit assignment or temporal-gradient flow [13, 19, 44]. These methods improve how SNNs are optimized through time, but they do not directly parameterize a neuron's frequency preference or response timing.

Another line of work improves temporal processing through network-level mechanisms. Recurrent spiking architectures with adaptive neurons extend effective memory across long temporal sequences [3, 4, 40], while learnable synaptic or axonal delays adjust spike arrival times to improve temporal alignment in feedforward or recurrent architectures [8, 16, 30, 35]. More recently, delay variables have also been introduced inside neuron state dynamics to provide each neuron with access to a finite temporal input history [24]. Frequency-domain attention modules such as FSTA-SNN [41] reweight spike feature maps using DCT-based spatial-temporal attention, but operate as feature-level modules rather than intrinsic neuron-level frequency-response parameterizations. Thus, these methods improve temporal routing, memory, or feature reweighting, but they do not directly parameterize the group-delay structure by which frequency-specific subthreshold responses accumulate before spike generation.

A parallel direction enriches single-neuron temporal dynamics. Adaptive LIF-type neurons, learnable time-constant variants, and gated LIF models introduce richer timescales or adaptation dynamics [2, 12, 39, 45]. Resonate-and-fire and structured state-space spiking models make oscillatory dynamics or frequency-sensitive regimes more accessible to learning [11, 17, 20, 21, 23, 33]. Structural neuron models further expand temporal computation through multiple branches, compartments, or parallel internal pathways [5, 36, 43, 46]. When frequency preference appears in these models, it is typically an implicit consequence of learned dynamical or state-transition parameters, rather than a response-level target frequency directly exposed to optimization. Classical impedance-based analyses characterize subthreshold resonance by the input frequency that maximizes the membrane-voltage response for a given neuron model [31]. FiTS builds on this view by making the frequency that maximizes the magnitude response a learnable target in the FS module. It further introduces the TS module to modulate group delay, shaping when frequency-specific responses contribute to membrane voltage before threshold crossing.

## 3 Methods

FiTS is a spiking neuron with two explicit mechanisms for temporal computation within each neuron. The Frequency Selectivity (FS) module parameterizes the nonzero target frequency that maximizes a neuron's subthreshold magnitude response. The Temporal Shaping (TS) module reshapes when frequency components contribute to pre-spike membrane voltage accumulation through group-delay modulation. We first present the FS module in Section 3.1, and then build on it with the TS module in Section 3.2.

### 3.1 FS Module: Learnable Frequency Selectivity

The standard LIF neuron follows first-order leaky membrane voltage dynamics:

$$\dot{V}(t) = -\mu V(t) + I(t), \quad \mu := \frac{1}{\tau_m}, \tag{1}$$

where $\tau_m$ is the membrane time constant and $I(t)$ is the input current. In the spiking model, spikes are emitted when the voltage crosses the threshold $V_{\mathrm{th}}$, followed by a reset.

To introduce frequency-selective subthreshold dynamics, we use a LIF neuron with a voltage-dependent adaptation current, a form used in adaptive and resonant spiking neuron models [2, 21, 23].

$$\dot{V}(t) = -\mu V(t) + I(t) - \eta a(t), \quad \dot{a}(t) = -\rho a(t) + \gamma V(t), \quad \rho := \frac{1}{\tau_a}, \tag{2}$$

where $a(t)$ denotes the adaptation current, $\tau_a$ is the adaptation time constant, $\gamma \geq 0$ is the coupling of voltage to adaptation, and $\eta \geq 0$ is the feedback from adaptation to voltage. We use the subthreshold response of the system in (2) to define an explicit target frequency for each neuron.

#### 3.1.1 Learnable Frequency Selectivity in Continuous Time

Under zero initial conditions, applying the Laplace transform to (2) yields the subthreshold frequency response

$$H(j\Omega) = \frac{V(j\Omega)}{I(j\Omega)} = \frac{\rho + j\Omega}{(\mu\rho + \kappa - \Omega^2) + j(\mu + \rho)\Omega} = |H(j\Omega)|e^{j\phi(\Omega)}. \tag{3}$$

Here, $\kappa := \eta\gamma$, $|H(j\Omega)|$ denotes the magnitude response, and $\phi(\Omega)$ denotes the phase response. When $|H(j\Omega)|$ has a unique nonzero global maximizer, we define the corresponding frequency as the *target frequency* $\Omega^\star$. This response-level definition is distinct from a pole-based intrinsic frequency, as discussed in Appendix A.6.

**Theorem 1** (Exact target frequency parameterization in continuous time)**.** *Consider the subthreshold frequency response* (3) *with* $\mu, \rho > 0$*. Whenever* $|H(j\Omega)|$ *admits a unique nonzero global maximizer over* $\Omega > 0$*, it is given by*

$$\Omega^\star = \sqrt{\sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)} - \rho^2}. \tag{4}$$

*Conversely, for any desired target frequency* $\Omega^\star > 0$*, there exists a unique* $\kappa^\star > 0$ *such that* $|H(j\Omega)|$ *is maximized at* $\Omega^\star$*. The corresponding value is*

$$\kappa^\star = \rho(\rho + \mu)\left[\sqrt{1 + \frac{\left(1 + (\Omega^\star/\rho)^2\right)^2}{(1 + \mu/\rho)^2}} - 1\right]. \tag{5}$$

A proof is provided in Appendix A.1.

The inverse mapping is the key parameterization of the FS module. If $\kappa$ were learned directly, the realized target frequency would remain an implicit property of the learned dynamics and would have to be recovered after training. FiTS instead uses $\Omega^\star$ as the trainable coordinate and computes the corresponding $\kappa^\star$ through (5). This makes the target frequency the common quantity used for initialization, optimization, and post-training interpretation.

The standard LIF dynamics in (1) are recovered from (2) by setting $\eta = 0$, or equivalently $\kappa = 0$. In that case, the magnitude response is maximized at $\Omega = 0$, restricting the neuron to low-pass behavior. Thus, without the voltage-dependent adaptation current, the standard LIF neuron cannot realize a nonzero target frequency.

#### 3.1.2 Discrete-Time Implementation and Realized Discrete-Time Target Frequency

Although the FS module is parameterized in continuous time for analytical tractability, practical training and inference are performed in discrete time. We therefore distinguish between the continuous-time target frequency used for parameterization and the realized discrete-time target frequency induced by the implemented update.

Let $\Delta t > 0$ denote the simulation timestep and $k$ the discrete-time index. We write $V[k]$ for the carried post-reset voltage, $a[k]$ for the adaptation current, and $I[k]$ for the input at timestep $k$. Let $V_0[k+1]$ denote the pre-reset voltage of the FS module at timestep $k+1$.

We discretize (2) using a semi-implicit Euler discretization. A comparison with a fully explicit Euler discretization is given in Appendix A.2. The resulting update is

$$\begin{aligned} V_0[k+1] &= (1-\mu\Delta t)V[k] - \eta\Delta t\, a[k] + I[k], \\ a[k+1] &= (1-\rho\Delta t)a[k] + \gamma\Delta t\, V_0[k+1]. \end{aligned} \tag{6}$$

Under the subthreshold regime and zero initial conditions, applying the $z$-transform to (6) and evaluating on the unit circle yields the discrete-time frequency response $H_d(e^{j\omega})$. When its magnitude response has a unique nonzero global maximizer, we denote the corresponding angular frequency by $\omega^\star_{\mathrm{DT}}$ and refer to it as the realized discrete-time target frequency.

The semi-implicit Euler update approximates the continuous-time dynamics. Thus, the realized discrete-time target $\omega^\star_{\mathrm{DT}}$ need not exactly match the continuous-time target $\Omega^\star$ in (4). We compare them in common frequency units using $f^\star_{\mathrm{CT}} = \Omega^\star/(2\pi)$ and $f^\star_{\mathrm{DT}} = \omega^\star_{\mathrm{DT}}/(2\pi\Delta t)$. Consequently, we use $\Omega^\star$ for learning because it gives an explicit closed-form inverse to the adaptation strength and remains close to the realized discrete-time target when $\Delta t$ is sufficiently small. Section 5.3 and Appendix A.3 analyze this gap.

### 3.2 TS Module: Learnable Temporal Shaping

While the FS module determines which temporal frequencies are emphasized, spike generation also depends critically on when frequency components contribute to membrane voltage accumulation. The phase response and corresponding group delay of $H_d(e^{j\omega})$ are

$$\phi_{\mathrm{FS}}(\omega) := \arg H_d(e^{j\omega}), \qquad \tau_{\mathrm{FS}}(\omega) := -\frac{d\phi_{\mathrm{FS}}(\omega)}{d\omega}, \tag{7}$$

where $\tau_{\mathrm{FS}}(\omega)$ is measured in samples and indicates how much frequency components near $\omega$ are delayed by the FS module.

Once the FS module parameters are fixed, the induced delay pattern is fixed as well. Thus, the neuron cannot tune frequency selectivity and threshold-crossing timing separately. The TS module introduces a complementary mechanism for temporal shaping.

#### 3.2.1 All-Pass Filter Cascade for Group-Delay Shaping

To reshape the group-delay pattern induced by the FS module, we introduce an auxiliary discrete-time all-pass cascade. All-pass filters preserve magnitude while modifying phase, making them natural building blocks for temporal shaping. Specifically, we use an $M$-stage cascade of first-order all-pass filters:

$$A_m(z) = \frac{z^{-1} - \beta_m}{1 - \beta_m z^{-1}}, \qquad A^{(M)}(z) = \prod_{m=1}^{M} A_m(z), \qquad |\beta_m| < 1 \tag{8}$$

where each stage has unit magnitude on the unit circle, and thus so does the full cascade.

Applying the inverse $z$-transform to $A_m(z)$ gives the recurrence

$$V_m[k+1] = \beta_m\big(V_m[k] - V_{m-1}[k+1]\big) + V_{m-1}[k], \qquad m = 1, \ldots, M, \tag{9}$$

where $V_{m-1}$ and $V_m$ denote the input and output voltages of the $m$-th stage, respectively. Applying this recurrence stage by stage yields the cascade, with $V_0[k+1]$ as its input.

The total group delay of the cascade is the sum of the stage-wise delays. For $|\beta_m| < 1$, each stage contributes a strictly positive group delay for all $\omega$. Consequently, pure cascade composition only augments the group delay induced by the FS module, rather than reshaping it more generally.

#### 3.2.2 Group-Delay Modulation via $\lambda$-Mixing

To enable richer group-delay modulation beyond pure all-pass cascade composition, the TS module recursively mixes the original FS module pathway with the outputs of the all-pass stages. With

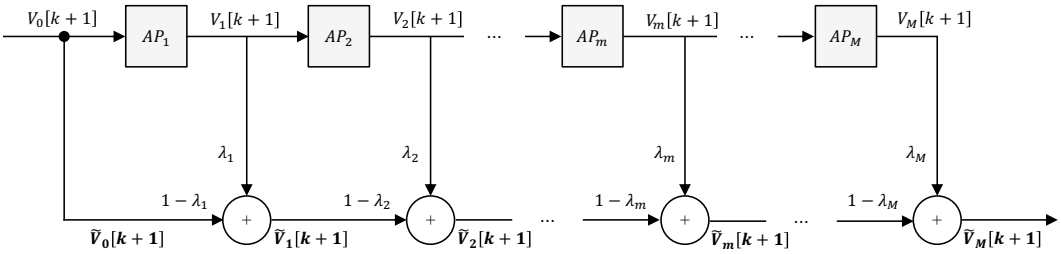



Figure 1: **TS module.** The FS output $V_0$ passes through an $M$-stage all-pass cascade and is recursively mixed with stage outputs using learnable $\lambda_m$ parameters. Each $\mathrm{AP}_m$ block denotes a first-order all-pass filter.

$\widetilde{V}_0[k+1] \equiv V_0[k+1]$, we define

$$\widetilde{V}_m[k+1] = (1-\lambda_m)\widetilde{V}_{m-1}[k+1] + \lambda_m V_m[k+1], \qquad m = 1, \ldots, M, \tag{10}$$

where $\lambda_m \in [0,1]$ controls the contribution of the $m$-th all-pass stage. The final mixed output $\widetilde{V}_M[k+1]$ serves as the effective pre-reset voltage, as illustrated in Figure 1.

The update in (10) forms a convex mixture of the direct pathway and all-pass stage outputs, so the induced response remains magnitude-bounded and does not introduce arbitrary amplification. However, because the phase of a mixture is not a mixture of phases, the induced group delay need not remain a positive sum of the constituent all-pass delays. This allows $\lambda$-mixing to realize group-delay behavior beyond pure all-pass cascade composition.

**Proposition 1** (Negative group-delay shift under $\lambda$-mixing)**.** *In the single-stage case, $\lambda$-mixing can induce a negative group-delay shift, which is impossible under pure all-pass cascade composition.*

The exact condition and proof are provided in Appendix A.4. Since every $M$-stage TS module contains the single-stage case by setting $\lambda_2 = \cdots = \lambda_M = 0$, this possibility extends to any $M \geq 1$.

More generally, the TS module recovers the FS-only model as a special case while enabling learnable group-delay shaping through the parameters $\{\beta_m, \lambda_m\}_{m=1}^{M}$. With only $2M$ additional neuron-wise parameters, it provides a compact mechanism for temporal shaping within the neuron. To satisfy the required constraints during optimization, we parameterize $\beta_m$ with a hyperbolic tangent and use sigmoid-constrained mixing coordinates. The full discrete-time update is summarized as Algorithm 1 in Appendix A.5.

# 4 Experiments

## 4.1 Datasets

We evaluate FiTS on three auditory classification datasets: two spike-based benchmarks, Spiking Heidelberg Digits (SHD) and Spiking Speech Commands (SSC), and one non-spiking benchmark, Google Speech Commands (GSC).

SHD and SSC were introduced for consistent quantitative comparisons of spiking neural networks and share a common audio-to-spike conversion pipeline [6]. SHD is a 20-class spiking speech benchmark derived from the Heidelberg Digits corpus, with spoken digits in English and German. Unlike SSC and GSC, SHD has no official validation split. We therefore report two SHD protocols in Table 1. One follows the commonly used SHD protocol that reports the best test accuracy over training only for comparability with prior work. The other holds out 20% of the training set for validation and reports test accuracy at the best validation epoch, following the validation-based SHD reporting used in SE-adLIF [2] and DelRec [30].

SSC is a 35-class spiking keyword-spotting benchmark derived from Google Speech Commands v0.02, with 75,466, 9,981, and 20,382 examples in the train, validation, and test splits, respectively. We also evaluate FiTS on Google Speech Commands (GSC) v0.02 [37], a non-spiking keyword-spotting benchmark built from one-second spoken commands. For GSC, we use conventional acoustic features rather than spike-based inputs.

Table 1: **Performance comparison on spiking auditory benchmarks.** Results are reported on SHD and SSC. We indicate whether each model uses explicit recurrent connections (Rec.) or explicit network-level delay mechanisms (Del.), together with temporal resolution (Res.), number of parameters, and top-1 accuracy. Results not marked as ours are taken from the corresponding original papers.

| Dataset | Method | Rec. | Del. | Res. | # Params | Acc. |
|---|---|---|---|---|---|---|
| SHD | RadLIF [4] | ✓ | × | 10 ms | 3.9 M | 94.62% |
| | S5-RF [21] | ✓ | × | 4 ms | 0.2 M | 91.86% |
| | SE-adLIF [2] | ✓ | × | 4 ms | 0.45 M | 95.81 ± 0.56% |
| | DCLS-Delays (3L-2KC) [16] | × | ✓ | 10 ms | 0.2 M | 95.07 ± 0.24% |
| | DelRec (rec. delays only) [30] | ✓ | ✓ | 10 ms | 0.17 M | 93.39 ± 0.45% |
| | Plain LIF (ours) | × | × | 4 ms | 0.04 M | 81.74 ± 0.76% |
| | TC-LIF [44] | × | × | 5.6 ms | 0.11 M | 83.08% |
| | cAdLIF [8] | × | × | 10 ms | 0.04 M | 94.19% |
| | DH-SFNN [46] | × | × | 1 ms | 0.05 M | 92.1% |
| | C-SiLIF [11] | × | × | 4 ms | 0.35 M | 89.30 ± 1.20% |
| | **FiTS (ours)** | × | × | **4 ms** | **0.04 M** | **95.31 ± 0.21%** |
| SHD* | SE-adLIF [2] | ✓ | × | 4 ms | 0.45 M | 93.79 ± 0.76% |
| | DelRec (rec. delays only) [30] | ✓ | ✓ | 10 ms | 0.24 M | 93.73 ± 0.69% |
| | Plain LIF (ours) | × | × | 4 ms | 0.11 M | 79.34 ± 0.67% |
| | **FiTS (ours)** | × | × | **4 ms** | **0.11 M** | **94.38 ± 0.12%** |
| SSC | RadLIF [4] | ✓ | × | 10 ms | 3.9 M | 77.4% |
| | S5-RF [21] | ✓ | × | 4 ms | 1.75 M | 78.8% |
| | SE-adLIF [2] | ✓ | × | 4 ms | 1.6 M | 80.44 ± 0.26% |
| | DCLS-Delays (3L-2KC) [16] | × | ✓ | 10 ms | 2.5 M | 80.69 ± 0.21% |
| | DelRec (rec. delays only) [30] | ✓ | ✓ | 5.6 ms | 0.37 M | 82.58 ± 0.08% |
| | Plain LIF (ours) | × | × | 4 ms | 0.35 M | 69.17 ± 0.18% |
| | TC-LIF [44] | × | × | 5.6 ms | 0.11 M | 63.46% |
| | cAdLIF [8] | × | × | 10 ms | 0.35 M | 77.5% |
| | DH-SFNN [46] | × | × | 1 ms | 0.27 M | 81.03% |
| | C-SiLIF† [11] | × | × | 4 ms | 0.35 M | 81.59 ± 0.31% |
| | **FiTS (ours)** | × | × | **4 ms** | **0.35 M** | **78.23 ± 0.16%** |

* SHD* uses a 20% hold-out validation split from the SHD training set for model selection. Test accuracy is reported at the best validation epoch.

† Uses a learnable neuron-wise discretization timestep instead of the standard shared timestep tied to the input-bin resolution.

### 4.2 Experimental Setup

**Input representation.** For SHD and SSC, we follow the standard preprocessing used in recent SNN benchmarks [2, 8, 11, 16, 21, 30]. We spatially bin the 700 afferent channels in groups of five, yielding 140 input features. Each 1.0 s sample is discretized into $T = 250$ time bins with $\Delta t = 4$ ms. For GSC, we use 40-band log-Mel spectrograms with $T = 100$ acoustic frames per utterance.

**Model configuration.** We evaluate FiTS in a feedforward fully connected architecture with two hidden layers and a linear classifier, without explicit recurrent connections or network-level delay mechanisms, so that its contribution can be isolated at the neuron level. We use a fixed firing threshold $V_{\text{th}} = 1.0$ for all benchmarks and $\eta = \gamma$. We report results for multiple hidden widths and TS cascade orders $M \in \{0, 1, 2\}$, where $M = 0$ denotes the FS-only model. For each layer, target frequencies are initialized on a log scale.

**Training details.** We train all FiTS and LIF-based ablation models for 100 epochs with cross-entropy loss, Adam [25], cosine annealing [27], a piecewise-linear triangular surrogate gradient [3, 10], and subtractive soft reset. We use a batch size of 128 and do not use weight decay, normalization layers, learnable thresholds, auxiliary losses, or data augmentation. For our experiments, we report the mean and standard deviation across 5 random seeds. Baseline results not marked as ours are taken from the corresponding original papers. All our experiments use a single NVIDIA RTX A5000 GPU. Appendix B reports the remaining neuron constants, selected hyperparameters, and training times.

Table 2: **Performance comparison on GSC with non-spiking acoustic inputs.** We report whether each model uses explicit recurrent connections (Rec.) or explicit network-level delay mechanisms (Del.), together with input frequency resolution (# Mel), parameter count, and top-1 accuracy. All entries use a 10 ms frame hop with approximately 100 timesteps. Results not marked as ours are taken from the corresponding original papers.

| Dataset | Method | Rec. | Del. | # Mel | # Params | Acc. |
|---|---|---|---|---|---|---|
| GSC | RadLIF [4] | ✓ | × | 40 | 0.83 M | 94.51% |
| | DCLS-Delays (3L-2KC) [16] | × | ✓ | 140 | 2.5 M | 95.35 ± 0.04% |
| | Plain LIF (ours) | × | × | 40 | 0.30 M | 80.02 ± 0.16% |
| | cAdLIF [8] | × | × | 40 | 0.30 M | 94.67% |
| | SiLIF† [11] | × | × | 40 | 1.1 M | 95.49 ± 0.09% |
| | **FiTS (ours)** | × | × | **40** | **0.30 M** | **94.48 ± 0.12%** |

† Uses a learnable neuron-wise discretization timestep instead of the standard shared timestep tied to the input-bin resolution.

### 4.3 Main Results

Tables 1 and 2 summarize the main results. Overall, FiTS performs strongly in the constrained feedforward setting without recurrence or network-level delays. On SHD, FiTS achieves $95.31 \pm 0.21\%$ under the best-test protocol and $94.38 \pm 0.12\%$ under the 20% validation-split protocol. On SSC and GSC, it achieves $78.23 \pm 0.16\%$ and $94.48 \pm 0.12\%$, respectively.

On SHD, FiTS is competitive under both reporting protocols. Under the best-test protocol, it remains competitive with recurrent and delay-based baselines. Under the 20% validation-split protocol, FiTS obtains the highest mean accuracy among the validation-based SHD reports in Table 1. This result is obtained with fewer parameters and without recurrence or network-level delays, indicating that neuron-level frequency selectivity and temporal shaping are effective on SHD.

On SSC, FiTS remains competitive under the same feedforward setting without recurrence or network-level delays. It improves over plain LIF, TC-LIF, and cAdLIF, while remaining below stronger feedforward baselines such as DH-SFNN and C-SiLIF. These results support the usefulness of FiTS's neuron-level temporal parameterization on SSC, although stronger neuron designs, including learnable timestep parameterizations, remain important for closing the gap.

On GSC, FiTS improves over the plain LIF baseline from $80.02 \pm 0.16\%$ to $94.48 \pm 0.12\%$ under the same feedforward setting without recurrence or network-level delays, while remaining close to stronger adaptive baselines. This result shows that FiTS can also be applied to conventional acoustic features, not only spike-based auditory inputs.

## 5 Discussion

### 5.1 FiTS learns layer-dependent frequency and timing organization

Figure 2 summarizes learned shifts in target frequency $\Delta f^\star$ and group delay $\Delta\tau$ across 20 random seeds. In both cases, the learned parameters exhibit clear layer-dependent organization after training.

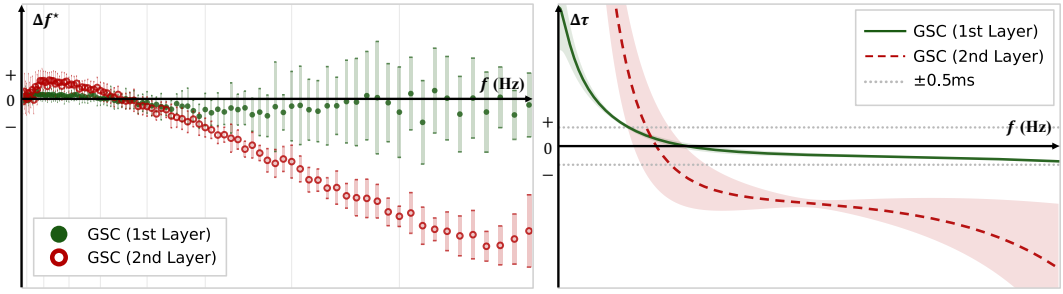



Figure 2: **Layer-dependent organization learned by FiTS on GSC.** Left: target frequency shifts learned by the FS module across random seeds. Right: group-delay shifts learned by the TS module across random seeds.

Table 3: **Factorized ablation on SSC across hidden sizes.** We compare baselines with FiTS variants initialized on a log-scale target frequency range through the inverse map. The FiTS rows separate frozen target frequencies, learnable target frequencies, and TS modules.

| Model | Hidden Size | | | |
|---|---|---|---|---|
| | **64** | **128** | **256** | **512** |
| ***Baselines*** | | | | |
| Plain LIF | $60.91 \pm 0.28\%$ | $65.91 \pm 0.16\%$ | $68.52 \pm 0.15\%$ | $69.17 \pm 0.18\%$ |
| Adapt. LIF (Frozen zero init.) | $61.18 \pm 0.22\%$ | $65.96 \pm 0.14\%$ | $68.38 \pm 0.13\%$ | $68.88 \pm 0.13\%$ |
| ***FiTS with log-scale target frequency init.*** | | | | |
| FS (Frozen target frequency) | $69.46 \pm 0.30\%$ | $73.75 \pm 0.15\%$ | $76.15 \pm 0.20\%$ | $77.13 \pm 0.22\%$ |
| FS (Learnable target frequency) | $71.72 \pm 0.12\%$ | $74.94 \pm 0.16\%$ | $76.81 \pm 0.08\%$ | $77.78 \pm 0.15\%$ |
| FS + TS ($M=1$) | $72.58 \pm 0.08\%$ | $75.69 \pm 0.12\%$ | $77.28 \pm 0.06\%$ | $78.22 \pm 0.12\%$ |
| FS + TS ($M=2$) | $\mathbf{73.16 \pm 0.30\%}$ | $\mathbf{75.94 \pm 0.16\%}$ | $\mathbf{77.50 \pm 0.18\%}$ | $\mathbf{78.23 \pm 0.16\%}$ |

The left panel of Figure 2 shows how target frequencies shift under the FS module. Both layers are initialized from the same log-scale distribution of target frequencies, but their target frequencies reorganize differently after training. Across random seeds, the second layer shifts more strongly toward lower target frequencies than the first, showing that the target frequencies do not simply remain near initialization. Appendix C.3 probes these targets with reset and shuffle perturbations.

The right panel of Figure 2 shows the learned group-delay shift induced by the TS module. In both layers, the learned delay shift is frequency-dependent and sign-indefinite, but its magnitude and shape differ across layers. Across random seeds, the second layer learns stronger and more structured group-delay shifts than the first. This suggests that the learned timing parameters summarize layer-dependent temporal organization at the parameter level. The learned distributions of $\beta$ and $\lambda$ are provided in Appendix C.4.

### 5.2 Ablation highlights complementary contributions of FS and TS

Table 3 shows that the largest gain comes from log-scale initialization of nonzero response-level target frequencies, enabled by the inverse map in Theorem 1. The adaptation-only baseline with zero-initialized adaptation shows no consistent improvement over plain LIF, whereas frozen target frequencies initialized on a log scale using (5) yield a significant improvement. Making these frequencies learnable through the FS module further improves accuracy at every tested hidden width. Thus, (5) supports both learning and frequency-domain initialization by allowing the initial target frequency range to be set directly based on the simulation timestep.

On SSC, the TS module provides a consistent additional improvement over FS-only. For both $M=1$ and $M=2$, adding TS improves accuracy at every tested hidden width. Since the TS module adds only $2M$ learnable parameters per neuron on top of the FS module, the gain is not driven by a large increase in learnable parameters. Appendix C.1 extends the SSC sweep to $M=3,4,5$, where larger cascade orders remain above FS-only without yielding monotonic gains.

We next characterize the accuracy–computation trade-off relative to plain LIF. We compute theoretical event-driven energy from standard AC/MAC counts using the 45 nm CMOS costs of Horowitz [18]. The estimates use $E_{\mathrm{AC}} = 0.9\,\mathrm{pJ}$, $E_{\mathrm{MAC}}=4.6\,\mathrm{pJ}$, and mean firing rates measured on the SSC validation split. They should be interpreted as theoretical cost estimates rather than measured hardware energy. Table 4 shows that FiTS reduces the spike-driven linear-layer cost relative to plain LIF by lowering firing rates, but shifts computation into neuron-internal updates. At hidden width 512, FS+TS ($M=2$) reduces the layer cost from 7.3 to $5.8\,\mu$J, while increasing the neuron-internal cost from 1.6 to $14.5\,\mu$J. The total estimated cost increases from 8.9 to $20.3\,\mu$J. This should be interpreted as an explicit accuracy–computation trade-off. FS provides a moderate-cost improvement over plain LIF, whereas TS trades additional neuron-internal computation for extra timing capacity.

Table 4: **Estimated theoretical energy ($\mu$J) on SSC at width 512.** Values are computed from AC/MAC counts and firing rates.

| **Model** | $\mathbf{E}_{\text{layer}}$ | $\mathbf{E}_{\text{neuron}}$ | $\mathbf{E}_{\text{total}}$ |
|---|---|---|---|
| Plain LIF | 7.3 | 1.6 | 8.9 |
| Adapt. (Frozen zero init.) | 7.3 | 5.6 | 13.0 |
| FS (Frozen target freq.) | 6.2 | 5.6 | 11.9 |
| FS (Learnable target freq.) | 6.2 | 5.6 | 11.8 |
| FS+TS ($M=1$) | 5.9 | 10.1 | 16.0 |
| FS+TS ($M=2$) | 5.8 | 14.5 | 20.3 |

We further estimate theoretical inference energy following the framework in [26] to account for arithmetic operations together with memory accesses and addressing. Table 5 reports the estimated energy per sample for the same SSC configuration. We assume 32-bit storage with memory costs determined by SRAM size and access FiTS states and coefficients at each timestep. Under these assumptions, FS and FS+TS require between approximately 81% and 85% of the energy of plain LIF. Memory accesses dominate the estimate and reduced spike-driven accesses to weights and membrane states offset the additional neuron-internal costs. The operation counts and arithmetic-cost coefficients differ from those used in Table 4. The relative advantage depends on the assumed memory costs and can disappear for FS+TS ($M = 2$) when the energy cost of accessing weights is reduced. These results are theoretical estimates rather than measured hardware energy.

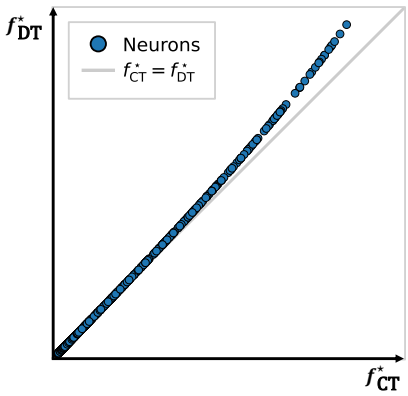
Table 5: **Estimated memory-aware energy ($\mu$J) on SSC at width 512.**

| **Model** | $\mathbf{E}_{\text{total}}$ |
|---|---|
| Plain LIF | 839.2 |
| Adapt. (Frozen zero init.) | 858.6 |
| FS (Frozen target freq.) | 711.5 |
| FS (Learnable target freq.) | 708.3 |
| FS+TS ($M = 1$) | 681.1 |
| FS+TS ($M = 2$) | 683.7 |

### 5.3 CT-to-DT fidelity of the FS parameterization

The FS module is parameterized by a continuous-time target frequency, but training and inference are carried out in discrete time through semi-implicit Euler discretization. A natural question is therefore whether the learned continuous-time target frequency remains a meaningful parameter in the implemented discrete-time neuron. To measure this, we sweep the input frequency for each neuron in a model trained on SHD and identify the realized discrete-time frequency that maximizes its magnitude response. Figure 3 shows the relation between the learned continuous-time target frequency and the realized discrete-time target frequency across neurons. In most cases the learned continuous-time target frequency closely matches the realized discrete-time target frequency, but the gap increases toward the high-frequency regime.

Figure 3: **CT vs. DT frequency.**




The gap observed in Figure 3 can be understood as a consequence of using semi-implicit Euler discretization rather than an exact discretization. Appendix A.3 accounts for this gap by deriving closed-form stationary candidates for the implemented discrete-time response. We denote this closed-form discrete-time estimate by $\widehat{f}^\star_{\mathrm{DT}}$.

As shown in Table 6, $\widehat{f}^\star_{\mathrm{DT}}$ reduces the mean absolute error from $0.7605\,\mathrm{Hz}$ to $0.0019\,\mathrm{Hz}$ and the maximum error from $7.4020\,\mathrm{Hz}$ to $0.0038\,\mathrm{Hz}$ relative to $f^\star_{\mathrm{DT}}$. Thus, $f^\star_{\mathrm{CT}}$ remains practical for learning, while $\widehat{f}^\star_{\mathrm{DT}}$ provides a near-exact post-training interpretation of the implemented discrete-time neuron.

Table 6: **CT-to-DT alignment.** Errors are measured against the realized discrete-time target frequency.

| **Comparison** | **MAE (Hz)** | **Max Err. (Hz)** |
|---|---|---|
| CT target $f^\star_{\mathrm{CT}}$ vs. realized DT $f^\star_{\mathrm{DT}}$ | 0.7605 | 7.4020 |
| closed-form DT $\widehat{f}^\star_{\mathrm{DT}}$ vs. realized DT $f^\star_{\mathrm{DT}}$ | **0.0019** | **0.0038** |

## 6 Conclusion

We introduced FiTS, a spiking neuron that factorizes neuron-level temporal computation into Frequency Selectivity (FS) and Temporal Shaping (TS). The FS module parameterizes the target frequency that maximizes the subthreshold magnitude response. The TS module uses group-delay modulation to shape when frequency components contribute to membrane voltage before spike generation.

Across SHD, SSC, and GSC, FiTS consistently improves over a plain LIF baseline in feedforward networks without explicit recurrence or network-level delay mechanisms. The learned target frequencies and group-delay shifts show clear layer-dependent organization. The CT-to-DT analysis further shows that the continuous-time target frequency remains practical for learning while allowing accurate discrete-time interpretation after training. These results suggest that FiTS provides a useful neuron-level temporal parameterization with parameter-level summaries of learned frequency selectivity and group-delay modulation.

This work focuses primarily on auditory benchmarks and isolates neuron-level temporal computation from stronger network-level mechanisms. Appendix C.5 reports additional results on sMNIST and Dynamic Vision Sensor (DVS) Gesture 128 [1], but broader evaluation beyond auditory tasks and interactions with recurrence, network-level delays, and richer structural neuron models remain important next steps. Finally, the TS module introduces additional neuron-internal computation while highlighting the underexplored problem of learning when frequency-specific responses contribute to pre-spike membrane voltage accumulation. This motivates more efficient implementations and broader neuron-level timing mechanisms.

## Acknowledgments and Disclosure of Funding

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00212845, Multimodal Speech Processing for Human-Computer Interaction).

## References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, et al. A low power, fully event-based gesture recognition system. In *Proc. CVPR*, 2017.

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

[3] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *NeurIPS*, 2018.

[4] Alexandre Bittar and Philip N. Garner. A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, Volume 16 - 2022, 2022.

[5] Xinyi Chen, Jibin Wu, Chenxiang Ma, Yinsong Yan, Yujie Wu, and Kay Chen Tan. PMSN: A parallel multi-compartment spiking neuron for multi-scale temporal processing. *arXiv preprint arXiv:2408.14917*, 2024.

[6] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Trans. Neural Networks*, 33:2744–2757, 2020.

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

[8] 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.

[9] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *Proc. ICLR*, 2022.

[10] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113:11441–11446, 2016.

[11] Maxime Fabre, Lyubov Dudchenko, and Emre Neftci. Structured state space model dynamics and parametrization for spiking neural networks. *arXiv preprint arXiv:2506.06374*, 2025.

[12] 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 *Proc. ICCV*, 2021.

[13] Wei Fang, Zhaofei Yu, Zhaokun Zhou, Ding Chen, Yanqi Chen, Zhengyu Ma, Timothée Masquelier, and Yonghong Tian. Parallel spiking neurons with high efficiency and ability to learn long-term dependencies. In *NeurIPS*, 2023.

[14] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, et al. Event-based vision: A survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 44:154–180, 2020.

[15] Yufei Guo, Yuanpei Chen, Zecheng Hao, Weihang Peng, Zhou Jie, Yuhan Zhang, Xiaode Liu, and Zhe Ma. Take a shortcut back: Mitigating the gradient vanishing for training spiking neural networks. In *NeurIPS*, 2024.

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

[17] Saya Higuchi, Sebastian Kairat, Sander M. Bohté, and Sebastian Otte. Balanced resonate-and-fire neurons. In *Proc. ICML*, 2024.

[18] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

[19] Yulong Huang, Xiaopeng Lin, Hongwei Ren, Haotian Fu, Yue Zhou, Zunchang Liu, Biao Pan, and Bojun Cheng. CLIF: Complementary leaky integrate-and-fire neuron for spiking neural networks. In *Proc. ICLR*, 2024.

[20] Yulong Huang, Zunchang Liu, Changchun Feng, Xiaopeng Lin, Hongwei Ren, Haotian Fu, Yue Zhou, Hong Xing, and Bojun Cheng. PRF: Parallel resonate and fire neuron for long sequence learning in spiking neural networks. *arXiv preprint arXiv:2410.03530*, 2024.

[21] Thomas E Huber, Jules Lecomte, Borislav Polovnikov, and Axel von Arnim. Scaling up resonate-and-fire networks for fast deep learning. In *Proc. ECCV*, 2024.

[22] Bruce Hutcheon and Yosef Yarom. Resonance, oscillation and the intrinsic frequency preferences of neurons. *Trends in Neurosciences*, 23:216–222, 2000.

[23] Eugene M. Izhikevich. Resonate-and-fire neurons. *Neural Networks*, pages 883–894, 2001.

[24] Sanja Karilanova, Subhrakanti Dey, and Ayça Özçelikkale. Delays in spiking neural networks: A state space model approach. *arXiv preprint arXiv:2512.01906*, 2025.

[25] Diederik P Kingma, Jimmy Ba, Y Bengio, and Y LeCun. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

[26] Edgar Lemaire, Loïc Cordone, Andrea Castagnetti, Pierre-Emmanuel Novac, Jonathan Courtois, and Benoît Miramond. An analytical estimation of spiking neural networks energy efficiency. In *International conference on neural information processing*, 2022.

[27] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proc. ICLR*, 2017.

[28] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, pages 1659–1671, 1997.

[29] Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Towards memory-and time-efficient backpropagation for training spiking neural networks. In *Proc. ICCV*, 2023.

[30] Alexandre Queant, Ulysse Rançon, Benoit R Cottereau, and Timothée Masquelier. DelRec: Learning delays in recurrent spiking neural networks. *arXiv preprint arXiv:2509.24852*, 2025.

[31] Magnus J. E. Richardson, Nicolas Brunel, and Vincent Hakim. From subthreshold to firing-rate resonance. *Journal of Neurophysiology*, 89:2538–2554, 2003.

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

[33] Shuaijie Shen, Chao Wang, Renzhuo Huang, Yan Zhong, Qinghai Guo, Zhichao Lu, Jianguo Zhang, and Luziwei Leng. SpikingSSMs: Learning long sequences with sparse and parallel spiking state space models. In *NeurIPS*, 2025.

[34] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In *NeurIPS*, 2018.

[35] Pengfei Sun, Longwei Zhu, and Dick Botteldooren. Axonal delay as a short-term memory for feed forward deep spiking neural networks. In *Proc. ICASSP*, 2022.

[36] Kexin Wang, Yuhong Chou, Di Shang, Shijie Mei, Jiahong Zhang, Yanbin Huang, Man Yao, Bo Xu, and Guoqi Li. MMDEND: Dendrite-inspired multi-branch multi-compartment parallel spiking neuron for sequence modeling. In *Proc. ACL*, 2025.

[37] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[38] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. In *NeurIPS*, 2022.

[39] Xingting Yao, Fanrong Li, Zitao Mo, and Jian Cheng. GLIF: A unified gated leaky integrate-and-fire neuron for spiking neural networks. In *NeurIPS*, 2022.

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

[41] Kairong Yu, Tianqing Zhang, Hongwei Wang, and Qi Xu. FSTA-SNN: Frequency-based spatial-temporal attention module for spiking neural networks. In *Proc. AAAI*, 2025.

[42] Friedemann Zenke and Surya Ganguli. SuperSpike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30:1514–1541, 2018.

[43] Dehao Zhang, Malu Zhang, Shuai Wang, Jingya Wang, Wenjie Wei, Zeyu Ma, Guoqing Wang, Yang Yang, and Haizhou Li. Dendritic resonate-and-fire neuron for effective and efficient long sequence modeling. In *NeurIPS*, 2026.

[44] Shimin Zhang, Qu Yang, Chenxiang Ma, Jibin Wu, Haizhou Li, and Kay Chen Tan. TC-LIF: A two-compartment spiking neuron model for long-term sequential modelling. In *Proc. AAAI*, 2024.

[45] Tianqing Zhang, Kairong Yu, Jian Zhang, and Hongwei Wang. DA-LIF: Dual adaptive leaky integrate-and-fire model for deep spiking neural networks. In *Proc. ICASSP*, 2025.

[46] Hanle Zheng, Zhong Zheng, Rui Hu, Bo Xiao, Yujie Wu, Fangwen Yu, Xue Liu, and Guoqi Li. Temporal dendritic heterogeneity incorporated with spiking neural networks for learning multi-timescale dynamics. *Nature Communications*, 15, 2024.

# Supplementary Material: FiTS


**A Theoretical Details** **14**

A.1 Proof of Theorem 1 . . . . . 14

A.2 Comparison of Semi-Implicit and Explicit Euler Discretization . . . . . 15

A.3 Exact CT-to-DT Mapping for the FS Module . . . . . 15

A.4 Proof of Proposition 1 . . . . . 16

A.5 Algorithmic Form of the FiTS Update . . . . . 17

A.6 Response-Level Target Frequency versus Intrinsic Pole Frequency . . . . . 17

**B Experimental Setup and Reproducibility** **18**

B.1 Additional experimental settings . . . . . 18

B.2 Training time . . . . . 19

**C Additional Results** **20**

C.1 Additional Ablation . . . . . 20

C.2 Sensitivity to Target-Frequency Initialization . . . . . 21

C.3 Target Frequency Perturbation . . . . . 21

C.4 Learned Distributions of $\beta$ and $\lambda$ . . . . . 22

C.5 Additional Temporal Benchmark Results . . . . . 22

# A Theoretical Details

## A.1 Proof of Theorem 1

We prove Theorem 1 by characterizing the nonzero stationary point of the magnitude response in (3), and then solving the resulting relation for $\kappa$ to obtain the inverse parameterization in (5).

Starting from (3), the squared magnitude response is

$$|H(j\Omega)|^2 = \frac{\rho^2 + \Omega^2}{(\mu\rho + \kappa - \Omega^2)^2 + (\mu + \rho)^2\Omega^2}. \tag{11}$$

Letting $x := \Omega^2$ with $x \geq 0$, define

$$M(x) := |H(j\Omega)|^2 = \frac{N(x)}{D(x)},$$

where $N(x)$ and $D(x)$ are the numerator and denominator of (11), respectively. A nonzero maximizer of $|H(j\Omega)|$ over $\Omega > 0$ is equivalent to a maximizer of $M(x)$ over $x > 0$. Differentiating gives

$$M'(x) = \frac{N'(x)D(x) - N(x)D'(x)}{\{D(x)\}^2}.$$

Substituting the derivatives of $N$ and $D$ and simplifying yields

$$x^2 + 2\rho^2 x - \Big[\kappa(2\rho^2 + 2\rho\mu + \kappa) - \rho^4\Big] = 0. \tag{12}$$

Hence the unique nonnegative stationary point is

$$x^\star = -\rho^2 + \sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)}. \tag{13}$$

It is strictly positive if and only if

$$\sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)} > \rho^2.$$

To establish uniqueness, observe that the numerator of $M'(x)$ is the negative of the quadratic polynomial in (12). Since this polynomial is strictly increasing on $x \geq 0$, it has at most one nonnegative root. Under the condition above, that root is $x^\star > 0$. Therefore $M'(x)$ changes sign from positive to negative at $x^\star$, which shows that $x^\star$ is the unique global maximizer of $M(x)$ over $x > 0$. Returning to $\Omega$, this gives exactly (4).

Conversely, let a desired target frequency $\Omega^\star > 0$ be given and set $x^\star = (\Omega^\star)^2$. Then (13) implies

$$x^\star + \rho^2 = \sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)}.$$

Squaring both sides gives

$$\kappa^2 + 2\rho(\rho + \mu)\kappa - (x^\star + \rho^2)^2 = 0. \tag{14}$$

Its unique positive solution is

$$\kappa^\star = -\rho(\rho + \mu) + \sqrt{\rho^2(\rho + \mu)^2 + (x^\star + \rho^2)^2}.$$

Substituting $x^\star = (\Omega^\star)^2$ and factoring out $\rho(\rho + \mu)$ yields

$$\kappa^\star = \rho(\rho + \mu)\left[\sqrt{1 + \frac{\left(1 + (\Omega^\star/\rho)^2\right)^2}{(1 + \mu/\rho)^2}} - 1\right],$$

which is exactly (5). Since the bracketed term is strictly positive for every $\Omega^\star > 0$, we have $\kappa^\star > 0$, and uniqueness follows from (14).

Finally, setting $\kappa = 0$ recovers the standard LIF case. Then (13) gives $x^\star = -\rho^2 < 0$, and therefore the magnitude response is maximized only at $\Omega = 0$. Thus the standard LIF neuron cannot realize a nonzero target frequency. □

### A.2 Comparison of Semi-Implicit and Explicit Euler Discretization

We compare the semi-implicit Euler discretization used in Section 3.1.2 with the fully explicit Euler discretization of (2). We focus on zero-input subthreshold dynamics, since they determine whether a given target frequency initialization yields a stable discrete-time state.

With $I(t) \equiv 0$, the fully explicit Euler discretization of (2) gives

$$\begin{bmatrix} V[k+1] \\ a[k+1] \end{bmatrix} = \begin{bmatrix} 1-\mu\Delta t & -\eta\Delta t \\ \gamma\Delta t & 1-\rho\Delta t \end{bmatrix} \begin{bmatrix} V[k] \\ a[k] \end{bmatrix},$$

whereas the semi-implicit discretization used in (6), with $V[k+1] \equiv V_0[k+1]$, gives

$$\begin{bmatrix} V[k+1] \\ a[k+1] \end{bmatrix} = \begin{bmatrix} 1-\mu\Delta t & -\eta\Delta t \\ \gamma\Delta t(1-\mu\Delta t) & 1-\rho\Delta t-\kappa\Delta t^2 \end{bmatrix} \begin{bmatrix} V[k] \\ a[k] \end{bmatrix}.$$

For either state-update matrix above, let

$$\lambda^2 - T\lambda + D,$$

denote its characteristic polynomial. The second-order Jury stability criterion then requires

$$1 - T + D > 0, \qquad 1 + T + D > 0, \qquad 1 - D > 0.$$

In our parameter regime, the active condition is the third inequality for the explicit discretization and the second inequality for the semi-implicit discretization. Solving these conditions for $\kappa$ yields

$$\kappa < \begin{cases} \dfrac{\mu+\rho}{\Delta t} - \mu\rho, & \text{(explicit)} \\ \mu\rho + \dfrac{4}{\Delta t^2} - \dfrac{2(\mu+\rho)}{\Delta t}, & \text{(semi-implicit)} \end{cases} \tag{15}$$

In the SSC setting used in our experiments, $\tau_m = 0.04$, $\tau_a = 0.2$, and $\Delta t = 0.004$. By Theorem 1, the target frequency increases monotonically with $\kappa$. Therefore, the stability bounds in (15) correspond to approximate target frequency limits of 13.8 Hz for the explicit discretization and 77.2 Hz for the semi-implicit discretization.

Figure A.1: **Zero-input trajectories.**

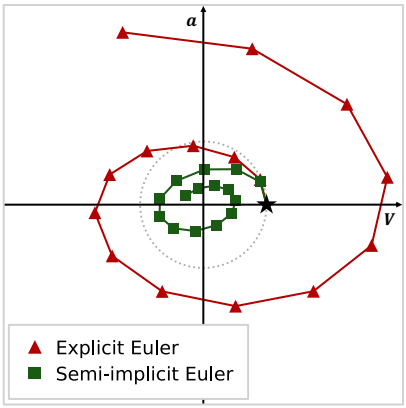



Figure A.1 shows the different dynamical behavior induced by the two discretization methods. Under the same zero-input initialization, the explicit discretization drives the discrete state $(V, a)$ along an oscillatory divergent trajectory and across the reference unit circle in the phase plane, whereas the semi-implicit discretization shows a convergent trajectory. This is consistent with the wider stable target frequency range of the semi-implicit discretization. We therefore use the semi-implicit discretization throughout the paper.

### A.3 Exact CT-to-DT Mapping for the FS Module

In this subsection, we derive the exact discrete-time transfer function induced by the semi-implicit Euler discretization (6) and characterize the realized discrete-time target frequency $\omega^\star_{\mathrm{DT}}$, defined in Section 3.1.2 as the unique nonzero global maximizer of its magnitude response. Throughout, we interpret (6) under $V[k+1] \equiv V_0[k+1]$.

**Theorem 2** (Closed-form stationary candidates for the realized discrete-time target frequency)**.** *Interpret* (6) *under the subthreshold identification* $V[k+1] \equiv V_0[k+1]$*, and define*

$$\bar{\mu} := 1 - \mu\Delta t, \qquad \bar{\rho} := 1 - \rho\Delta t, \qquad \bar{\kappa} := \eta\gamma\Delta t^2.$$

*Let* $H_d(e^{j\omega})$ *denote the discrete-time frequency response induced by* (6)*. Then the stationary candidates of* $|H_d(e^{j\omega})|$ *over* $\omega \in (0, \pi)$ *are*

$$x_\pm = \frac{1 + \bar{\rho}^2 \pm \sqrt{\frac{\bar{\kappa}}{\bar{\mu}}\Big((1-\bar{\rho}^2)(1-\bar{\mu}\bar{\rho}) + \bar{\kappa}\bar{\rho}\Big)}}{2\bar{\rho}}, \tag{16}$$

*with corresponding stationary frequencies*

$$\omega_\pm = \arccos(x_\pm), \qquad \textit{whenever } x_\pm \in (-1,1). \tag{17}$$

*Proof.* Applying the $z$-transform to (6) under zero initial conditions, together with the subthreshold identification $V[k+1] \equiv V_0[k+1]$, yields

$$H_d(z) = \frac{V(z)}{I(z)} = \frac{z-\bar{\rho}}{(z-\bar{\mu})(z-\bar{\rho})+\bar{\kappa}z}.$$

Evaluating on the unit circle gives

$$H_d(e^{j\omega}) = \frac{e^{j\omega}-\bar{\rho}}{(e^{j\omega}-\bar{\mu})(e^{j\omega}-\bar{\rho})+\bar{\kappa}e^{j\omega}}, \qquad \omega \in [0,\pi]. \tag{18}$$

Let $x := \cos\omega$. Then

$$M(x) := \left|H_d(e^{j\omega})\right|^2 = \frac{1+\bar{\rho}^2-2\bar{\rho}x}{1+(\bar{\kappa}-\bar{\mu}-\bar{\rho})^2+\bar{\mu}^2\bar{\rho}^2-2\bar{\mu}\bar{\rho}+2(\bar{\kappa}-\bar{\mu}-\bar{\rho})(1+\bar{\mu}\bar{\rho})x+4\bar{\mu}\bar{\rho}x^2}. \tag{19}$$

Since $x = \cos\omega$ is strictly decreasing on $(0,\pi)$, maximizing $|H_d(e^{j\omega})|$ over $\omega \in (0,\pi)$ is equivalent to maximizing $M(x)$ over $x \in (-1,1)$. Hence any interior maximizer must satisfy $M'(x) = 0$. Differentiating (19) with respect to $x$ and simplifying yields

$$4\bar{\mu}\bar{\rho}^2x^2 - 4\bar{\mu}\bar{\rho}(1+\bar{\rho}^2)x + \bar{\mu}(1+\bar{\rho}^2)^2 - \bar{\kappa}\Big((1-\bar{\rho}^2)(1-\bar{\mu}\bar{\rho})+\bar{\kappa}\bar{\rho}\Big) = 0. \tag{20}$$

Under $0<\bar{\mu}<1$, $0<\bar{\rho}<1$, and $\bar{\kappa}>0$, the factor $(1-\bar{\rho}^2)(1-\bar{\mu}\bar{\rho})+\bar{\kappa}\bar{\rho}$ is strictly positive. Therefore the discriminant of (20) is positive, and the stationary candidates are real. Solving (20) gives (16), and (17) follows by setting $\omega_\pm = \arccos(x_\pm)$ whenever $x_\pm \in (-1,1)$. □

Theorem 2 provides a closed-form characterization of the stationary candidates of the realized discrete-time target frequency. Given a continuous-time target frequency, the induced $\bar{\kappa}$ defines the exact discrete-time frequency response. The realized discrete-time target frequency is then obtained by selecting the valid maximizer among the stationary candidates.

### A.4 Proof of Proposition 1

We consider the single-stage case, since the multi-stage TS module contains it by setting $\lambda_2 = \cdots = \lambda_M = 0$. Let

$$A_1(e^{j\omega}) = e^{j\phi_{A_1}(\omega)}$$

denote the first-stage all-pass response, and let

$$G_1(e^{j\omega}) := (1-\lambda_1) + \lambda_1 A_1(e^{j\omega}) \tag{21}$$

be the corresponding single-stage $\lambda$-mixed response.

For pure all-pass composition, each stage contributes a nonnegative all-pass delay

$$\tau_{A_1}(\omega) := -\frac{d}{d\omega}\phi_{A_1}(\omega) \geq 0.$$

Thus pure all-pass cascade composition can only produce nonnegative stage-wise group-delay shifts. Under $\lambda$-mixing, this restriction no longer holds.

Writing

$$G_1(e^{j\omega}) = \big((1-\lambda_1) + \lambda_1\cos\phi_{A_1}(\omega)\big) + j\,\lambda_1\sin\phi_{A_1}(\omega),$$

the group delay of $G_1(e^{j\omega})$ is obtained by differentiating its phase. A direct calculation yields

$$\tau_{G_1}(\omega) = \frac{\lambda_1\big(\lambda_1 + (1-\lambda_1)\cos\phi_{A_1}(\omega)\big)}{\lambda_1^2 + (1-\lambda_1)^2 + 2\lambda_1(1-\lambda_1)\cos\phi_{A_1}(\omega)}\tau_{A_1}(\omega). \tag{22}$$

The denominator in (22) is

$$\left|(1-\lambda_1) + \lambda_1 e^{j\phi_{A_1}(\omega)}\right|^2,$$

and is therefore nonnegative, with equality only in the degenerate cancellation case. Since $\lambda_1 \in (0,1)$ and $\tau_{A_1}(\omega) \geq 0$, the sign of $\tau_{G_1}(\omega)$ is determined by

$$\lambda_1 + (1-\lambda_1)\cos\phi_{A_1}(\omega).$$

Hence

$$\tau_{G_1}(\omega) < 0 \quad \Longleftrightarrow \quad \lambda_1 + (1-\lambda_1)\cos\phi_{A_1}(\omega) < 0. \tag{23}$$

Solving this inequality yields the exact condition

$$\lambda_1 < \frac{1}{2} \qquad \text{and} \qquad \cos\phi_{A_1}(\omega) < -\frac{\lambda_1}{1-\lambda_1}. \tag{24}$$

This proves the single-stage claim. The extension to any $M \geq 1$ follows immediately by setting $\lambda_2 = \cdots = \lambda_M = 0$.

### A.5 Algorithmic Form of the FiTS Update

Algorithm 1 summarizes the full discrete-time update of the FiTS neuron. The update proceeds in three stages: the FS module computes the subthreshold voltage and adaptation update, the TS module applies stage-wise all-pass filtering and $\lambda$-mixing, and the final mixed response is used for spike generation and soft reset.

**Algorithm 1** Discrete-Time Update of the FiTS Neuron

**Input:** $V[k]$, $a[k]$, $\{V_m[k]\}_{m=0}^{M}$, $I[k]$
**Output:** $V[k+1]$, $a[k+1]$, $\{V_m[k+1]\}_{m=0}^{M}$, $S[k+1]$

**FS module**

$V_0[k+1] \leftarrow (1-\mu\Delta t)V[k] - \eta\Delta t\, a[k] + I[k]$
$a[k+1] \leftarrow (1-\rho\Delta t)a[k] + \gamma\Delta t\, V_0[k+1]$

**TS module**

$\widetilde{V}_0[k+1] \leftarrow V_0[k+1]$
**for** $m = 1, \ldots, M$ **do**
    $V_m[k+1] \leftarrow \beta_m\big(V_m[k] - V_{m-1}[k+1]\big) + V_{m-1}[k]$
    $\widetilde{V}_m[k+1] \leftarrow (1-\lambda_m)\widetilde{V}_{m-1}[k+1] + \lambda_m V_m[k+1]$
**end for**

**Spike Generation and Reset**

$S[k+1] \leftarrow \Theta(\widetilde{V}_M[k+1] - V_{\mathrm{th}})$
$V[k+1] \leftarrow \widetilde{V}_M[k+1] - S[k+1]V_{\mathrm{th}}$

### A.6 Response-Level Target Frequency versus Intrinsic Pole Frequency

This section clarifies the distinction between the FiTS target frequency and an intrinsic frequency defined from the homogeneous dynamics. Starting from the input-to-voltage response in (3), the squared magnitude response is

$$|H(j\Omega)|^2 = \frac{\rho^2 + \Omega^2}{(\mu\rho + \kappa - \Omega^2)^2 + (\mu+\rho)^2\Omega^2}. \tag{25}$$

FiTS defines the target frequency as the nonzero maximizer of this input-to-voltage magnitude response. By (4), this response-level target satisfies

$$(\Omega^\star)^2 = \sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)} - \rho^2. \tag{26}$$

By contrast, an intrinsic pole frequency is determined by the homogeneous part of the same linear subsystem. The denominator of (3) gives the characteristic polynomial

$$s^2 + (\mu+\rho)s + (\mu\rho + \kappa) = 0. \tag{27}$$

When the poles are complex, the magnitude of their imaginary part is

$$\Omega_{\mathrm{pole}} = \frac{1}{2}\sqrt{4(\mu\rho+\kappa) - (\mu+\rho)^2}, \tag{28}$$

and hence

$$\Omega_{\mathrm{pole}}^2 = \mu\rho + \kappa - \frac{(\mu+\rho)^2}{4}. \tag{29}$$

This quantity characterizes the natural mode of the homogeneous subsystem. It is not, in general, the same as the frequency at which input current most strongly drives membrane voltage.

The difference follows directly from Eqs. (26) and (29). Equating the response-level target frequency and the pole-imaginary frequency would require

$$\sqrt{\kappa(2\rho^2 + 2\rho\mu + \kappa)} - \rho^2 = \mu\rho + \kappa - \frac{(\mu+\rho)^2}{4}, \tag{30}$$

which is not satisfied in general. The response-level target frequency depends on both the numerator and denominator of the input-to-voltage transfer function, whereas the imaginary part of the pole depends only on the homogeneous denominator.

This distinction also separates response-level frequency selectivity from intrinsic oscillation. A nonzero imaginary part of the pole exists only when

$$4(\mu\rho + \kappa) > (\mu+\rho)^2. \tag{31}$$

However, this condition is not equivalent to the existence of a nonzero maximizer of $|H(j\Omega)|$. For example, with $\mu=4$, $\rho=1$, and $\kappa=2$, we have

$$4(\mu\rho + \kappa) = 24 < 25 = (\mu+\rho)^2,$$

so the poles are real and $\Omega_{\mathrm{pole}}$ is not a real-valued intrinsic oscillation frequency. Nevertheless, (26) gives

$$(\Omega^\star)^2 = \sqrt{24} - 1 > 0,$$

so the input-to-voltage response has a nonzero target frequency $\Omega^\star \approx 1.97$. Thus, a neuron can exhibit a response-level frequency preference even when its homogeneous dynamics do not have an oscillatory eigenmode.

This response-level definition is relevant for spiking neurons because spike generation is determined by threshold crossing of the membrane voltage. FiTS therefore parameterizes the frequency at which input current most strongly contributes to membrane voltage, rather than only the natural frequency of the autonomous linear subsystem.

# B Experimental Setup and Reproducibility

## B.1 Additional experimental settings

Table A.1 reports the dataset-specific neuron constants used in the benchmark experiments. Shared training settings are given in the main text. For all datasets, we set the adaptation current scale ratio to $\eta/\gamma = 1.0$. The TS parameters are initialized in unconstrained coordinates. We initialize the all-pass coordinate as $\hat{\beta}_m = 0$, which gives $\beta_m = \tanh(\hat{\beta}_m) = 0$. The implementation mixes stage outputs from the last stage toward the FS pathway using $q_m = \sigma(\hat{q}_m)$ and optimizes $\hat{q}_m$ initialized at $-3.0$, giving $q_m \approx 0.047$. This is equivalent to (10) under $\lambda_m = (P_m - P_{m+1})/(1 - P_{m+1})$, where $P_m = \prod_{i=1}^{m} q_i$ and $P_{M+1} = 0$. Thus, the TS module is initialized close to the direct FS pathway and learns the amount of all-pass mixing during training.

Table A.1: **Dataset-specific neuron constants.** The firing threshold and shared training settings are reported in the main text.

| Setting | SHD | SSC | GSC |
|---|---|---|---|
| $\tau_m$ | 0.04 s | 0.04 s | 0.10 s |
| $\tau_a$ | 0.20 s | 0.20 s | 0.50 s |
| $\Delta t$ | 0.004 s | 0.004 s | 0.010 s |
| Target frequency range $f^\star$ | $[1, 50]$ Hz | $[1, 50]$ Hz | $[1, 30]$ Hz |

For reproducibility, Table A.2 reports the selected learning rate and dropout probability for each dataset, cascade order, and hidden width. These values correspond to the final configurations used in the benchmark experiments.

Table A.2: **Selected learning rate and dropout.** Each entry reports learning rate $lr$ and dropout probability $p$ by dataset, cascade order, and hidden width.

| Dataset | Order | $\mathbf{W = 64}$ | $\mathbf{W = 128}$ | $\mathbf{W = 256}$ | $\mathbf{W = 512}$ |
|---|---|---|---|---|---|
| SHD | $M = 0$ | $lr = 3\text{e}{-3},\ p = 0.5$ | $lr = 6.5\text{e}{-3},\ p = 0.4$ | $lr = 2\text{e}{-3},\ p = 0.3$ | $lr = 1.5\text{e}{-3},\ p = 0.4$ |
| | $M = 1$ | $lr = 9\text{e}{-3},\ p = 0.2$ | $lr = 7.5\text{e}{-3},\ p = 0.5$ | $lr = 7.5\text{e}{-4},\ p = 0.2$ | $lr = 2\text{e}{-3},\ p = 0.3$ |
| | $M = 2$ | $lr = 5.5\text{e}{-3},\ p = 0.6$ | $lr = 6.5\text{e}{-3},\ p = 0.6$ | $lr = 4.5\text{e}{-3},\ p = 0.3$ | $lr = 2\text{e}{-2},\ p = 0.2$ |
| SSC | $M = 0$ | $lr = 2\text{e}{-3},\ p = 0.3$ | $lr = 1.5\text{e}{-3},\ p = 0.3$ | $lr = 6\text{e}{-4},\ p = 0.45$ | $lr = 6\text{e}{-4},\ p = 0.55$ |
| | $M = 1$ | $lr = 1.25\text{e}{-3},\ p = 0.2$ | $lr = 1\text{e}{-3},\ p = 0.35$ | $lr = 5.5\text{e}{-4},\ p = 0.5$ | $lr = 5.5\text{e}{-4},\ p = 0.6$ |
| | $M = 2$ | $lr = 3\text{e}{-3},\ p = 0.3$ | $lr = 1.5\text{e}{-3},\ p = 0.45$ | $lr = 8\text{e}{-4},\ p = 0.55$ | $lr = 4.5\text{e}{-4},\ p = 0.55$ |
| GSC | $M = 0$ | $lr = 4\text{e}{-3},\ p = 0.15$ | $lr = 2.5\text{e}{-3},\ p = 0.20$ | $lr = 2.5\text{e}{-3},\ p = 0.3$ | $lr = 9\text{e}{-4},\ p = 0.45$ |
| | $M = 1$ | $lr = 3.5\text{e}{-3},\ p = 0.2$ | $lr = 3\text{e}{-3},\ p = 0.25$ | $lr = 3\text{e}{-3},\ p = 0.4$ | $lr = 3\text{e}{-3},\ p = 0.65$ |
| | $M = 2$ | $lr = 4\text{e}{-3},\ p = 0.2$ | $lr = 3.5\text{e}{-3},\ p = 0.3$ | $lr = 2\text{e}{-3},\ p = 0.55$ | $lr = 1\text{e}{-3},\ p = 0.45$ |

## B.2 Training time

FiTS is evaluated in feedforward SNNs without recurrence or network-level delays. This allows layer-wise temporal processing. Once the spike sequence from the previous layer is available, the temporal update of the next layer can be evaluated over the full sequence. We use this property to implement the FiTS temporal update with a fused Triton kernel.

The fused implementation processes temporal chunks within each layer and combines the FS module update, TS module update, spike generation, soft reset, and backward pass. To isolate the implementation effect, we compare it with a PyTorch timestep loop under a setting with two hidden layers and SSC-like tensor shapes. This benchmark uses the same temporal update in both cases and changes only the implementation. The two implementations produce identical spike outputs in all tested settings.

Table A.3: **Runtime of the temporal FiTS update.** We compare a PyTorch timestep loop with the fused Triton implementation under a setting with two hidden layers and SSC-like tensor shapes. Times include forward and backward passes and are reported as median milliseconds per iteration on a single NVIDIA RTX A5000 GPU. Bold values in parentheses report speedup over the PyTorch loop.

| $W$ | $M$ | PyTorch (ms) | Triton (ms) |
|---|---|---|---|
| 64 | $M = 0$ | 426.7 | 8.3 (**51.16×**) |
| | $M = 1$ | 627.1 | 7.3 (**86.35×**) |
| | $M = 2$ | 861.8 | 6.7 (**129.04×**) |
| 128 | $M = 0$ | 521.8 | 7.3 (**71.13×**) |
| | $M = 1$ | 704.2 | 12.8 (**54.81×**) |
| | $M = 2$ | 876.6 | 10.4 (**84.35×**) |
| 256 | $M = 0$ | 537.8 | 17.3 (**31.09×**) |
| | $M = 1$ | 933.3 | 6.9 (**134.56×**) |
| | $M = 2$ | 1099.9 | 20.6 (**53.30×**) |
| 512 | $M = 0$ | 1079.4 | 33.5 (**32.20×**) |
| | $M = 1$ | 1385.9 | 25.0 (**55.41×**) |
| | $M = 2$ | 1437.2 | 33.0 (**43.51×**) |

Table A.3 shows that the fused Triton implementation reduces forward and backward runtime by $31\times$ to $135\times$ relative to the PyTorch timestep loop across the tested hidden widths and TS cascade orders. This speedup comes from evaluating the temporal update within a fused layer-wise implementation rather than changing the mathematical operations of the neuron model. These measurements are implementation-level runtime results. They are separate from the theoretical operation-level energy estimates in Table 4, and should not be interpreted as speedups for architectures with inter-layer temporal recurrence.

Table A.4 reports the resulting per-run training times used in our benchmark experiments. Times are measured in minutes on a single NVIDIA RTX A5000 GPU and summarized by the median over completed runs. The measured times are similar across $M = 0, 1, 2$ in the tested settings, indicating that the fused implementation keeps feedforward FiTS training practical when TS cascade orders are used.

Table A.4: **Training time across experimental settings.** Entries are median per-run times in minutes on a single NVIDIA RTX A5000 GPU.

| Dataset | Order | $\mathbf{W = 64}$ | $\mathbf{W = 128}$ | $\mathbf{W = 256}$ | $\mathbf{W = 512}$ |
|---|---|---|---|---|---|
| SHD | $M = 0$ | 6.6 | 6.6 | 6.0 | 8.4 |
| | $M = 1$ | 6.0 | 6.6 | 6.6 | 7.8 |
| | $M = 2$ | 6.6 | 6.6 | 6.6 | 8.4 |
| SSC | $M = 0$ | 51.0 | 50.4 | 52.2 | 68.4 |
| | $M = 1$ | 51.6 | 53.4 | 52.2 | 69.0 |
| | $M = 2$ | 50.4 | 54.0 | 52.8 | 69.0 |
| GSC | $M = 0$ | 19.8 | 19.8 | 21.6 | 22.2 |
| | $M = 1$ | 21.0 | 20.4 | 21.0 | 24.0 |
| | $M = 2$ | 21.0 | 21.0 | 21.0 | 24.0 |

# C Additional Results

## C.1 Additional Ablation

**Factorized ablations on GSC and SHD$^*$.** Tables A.5 and A.6 provide the full ablations for GSC and SHD$^*$. They follow the same factorized comparison as the SSC ablation in the main text. Log-scale initialization of target frequencies gives a large improvement over plain LIF on both benchmarks. The learnable FS module and TS module further refine this baseline, although the TS effect depends on the dataset, hidden width, and cascade order.

Table A.5: **Factorized ablation on GSC across hidden sizes.** We compare baselines with FiTS variants initialized on a log-scale target frequency range through the inverse map. The FiTS rows separate frozen target frequencies, learnable target frequencies, and TS modules.

| Model | Hidden Size | | | |
|---|---|---|---|---|
| | **64** | **128** | **256** | **512** |
| ***Baselines*** | | | | |
| Plain LIF | $74.17 \pm 0.32\%$ | $78.02 \pm 0.18\%$ | $77.61 \pm 0.37\%$ | $80.02 \pm 0.16\%$ |
| Adapt. LIF (Frozen zero init.) | $74.45 \pm 0.38\%$ | $78.05 \pm 0.25\%$ | $77.91 \pm 0.15\%$ | $79.89 \pm 0.19\%$ |
| ***FiTS with log-scale target frequency init.*** | | | | |
| FS (Frozen target frequency) | $84.99 \pm 0.31\%$ | $89.72 \pm 0.15\%$ | $91.82 \pm 0.14\%$ | $92.50 \pm 0.15\%$ |
| FS (Learnable target frequency) | $89.84 \pm 0.06\%$ | $92.01 \pm 0.10\%$ | $92.89 \pm 0.10\%$ | $93.27 \pm 0.12\%$ |
| FS + TS ($M = 1$) | $90.89 \pm 0.16\%$ | $92.95 \pm 0.10\%$ | $\mathbf{94.08 \pm 0.20\%}$ | $\mathbf{94.48 \pm 0.12\%}$ |
| FS + TS ($M = 2$) | $\mathbf{90.94 \pm 0.18\%}$ | $\mathbf{93.06 \pm 0.04\%}$ | $93.94 \pm 0.12\%$ | $93.96 \pm 0.14\%$ |

Table A.6: **Factorized ablation on SHD$^*$ across hidden sizes.** All results use the validation-split SHD protocol, where 20% of the training set is held out for validation and test accuracy is reported at the best validation epoch. We compare baselines with FiTS variants initialized on a log-scale target frequency range through the inverse map. The FiTS rows separate frozen target frequencies, learnable target frequencies, and TS modules.

| Model | Hidden Size | | | |
|---|---|---|---|---|
| | **64** | **128** | **256** | **512** |
| ***Baselines*** | | | | |
| Plain LIF | $77.80 \pm 0.69\%$ | $78.82 \pm 0.78\%$ | $79.34 \pm 0.67\%$ | $79.70 \pm 0.35\%$ |
| Adapt. LIF (Frozen zero init.) | $77.50 \pm 0.67\%$ | $78.16 \pm 0.84\%$ | $79.47 \pm 0.57\%$ | $79.47 \pm 0.36\%$ |
| ***FiTS with log-scale target frequency init.*** | | | | |
| FS (Frozen target frequency) | $90.43 \pm 0.56\%$ | $91.55 \pm 0.39\%$ | $92.30 \pm 0.38\%$ | $92.30 \pm 0.35\%$ |
| FS (Learnable target frequency) | $93.01 \pm 0.18\%$ | $93.70 \pm 0.14\%$ | $93.99 \pm 0.12\%$ | $94.02 \pm 0.21\%$ |
| FS + TS ($M = 1$) | $\mathbf{93.47 \pm 0.20\%}$ | $\mathbf{94.12 \pm 0.30\%}$ | $93.99 \pm 0.15\%$ | $\mathbf{94.07 \pm 0.11\%}$ |
| FS + TS ($M = 2$) | $93.25 \pm 0.38\%$ | $93.39 \pm 0.15\%$ | $\mathbf{94.38 \pm 0.12\%}$ | $94.04 \pm 0.24\%$ |

**Larger TS cascade orders on SSC.** To test whether the gains from the TS module are explained by a larger cascade order alone, we extend the SSC ablation to $M = 3, 4, 5$. Table A.7 reports the

extended sweep at the selected hidden widths. This experiment varies only the TS cascade order while keeping the rest of the architecture and training protocol unchanged.

The TS module consistently improves over FS-only across all tested cascade orders. However, the improvement is not monotonic in $M$. Thus, the cascade order is better viewed as a hyperparameter controlling temporal shaping capacity rather than as a simple scaling axis. More systematic order selection and broader within-neuron mechanisms for shaping pre-spike membrane voltage accumulation remain useful directions for future work.

Table A.7: **Extended TS cascade order sweep on SSC across hidden sizes.** We extend the SSC ablation by evaluating larger TS cascade orders $M = 3, 4, 5$. FiTS variants are initialized on a log-scale target frequency range through the inverse map. Results are reported as mean $\pm$ standard deviation across random seeds.

| Model | Hidden Size | | | |
|---|---|---|---|---|
| | 64 | 128 | 256 | 512 |
| ***Baselines*** | | | | |
| Plain LIF | $60.91 \pm 0.28\%$ | $65.91 \pm 0.16\%$ | $68.52 \pm 0.15\%$ | $69.17 \pm 0.18\%$ |
| Adapt. LIF (Frozen zero init.) | $61.18 \pm 0.22\%$ | $65.96 \pm 0.14\%$ | $68.38 \pm 0.13\%$ | $68.88 \pm 0.13\%$ |
| ***FiTS with log-scale target frequency init.*** | | | | |
| FS (Frozen target frequency) | $69.46 \pm 0.30\%$ | $73.75 \pm 0.15\%$ | $76.15 \pm 0.20\%$ | $77.13 \pm 0.22\%$ |
| FS (Learnable target frequency) | $71.72 \pm 0.12\%$ | $74.94 \pm 0.16\%$ | $76.81 \pm 0.08\%$ | $77.78 \pm 0.15\%$ |
| FS + TS ($M = 1$) | $72.58 \pm 0.08\%$ | $75.69 \pm 0.12\%$ | $77.28 \pm 0.06\%$ | $78.22 \pm 0.12\%$ |
| FS + TS ($M = 2$) | $\mathbf{73.16 \pm 0.30\%}$ | $75.94 \pm 0.16\%$ | $\mathbf{77.50 \pm 0.18\%}$ | $78.23 \pm 0.16\%$ |
| FS + TS ($M = 3$) | $73.01 \pm 0.10\%$ | $75.93 \pm 0.13\%$ | $77.47 \pm 0.08\%$ | $78.23 \pm 0.15\%$ |
| FS + TS ($M = 4$) | $73.06 \pm 0.13\%$ | $\mathbf{75.97 \pm 0.21\%}$ | $77.47 \pm 0.16\%$ | $\mathbf{78.25 \pm 0.14\%}$ |
| FS + TS ($M = 5$) | $73.16 \pm 0.08\%$ | $75.87 \pm 0.05\%$ | $77.46 \pm 0.09\%$ | $78.12 \pm 0.10\%$ |

## C.2 Sensitivity to Target-Frequency Initialization

We compare target-frequency initialization schemes on SHD* using FS-only networks ($M = 0$) with hidden width 256. Target frequencies are evenly spaced on a log scale or a linear scale over $[1, 50]$ Hz. We also evaluate constant initializations that assign all neurons a target frequency of $\sqrt{50}$ Hz or 25 Hz. The value $\sqrt{50}$ Hz is the geometric mean of the interval endpoints. For each initialization, we evaluate both frozen and learnable target frequencies. All configurations use 100 training epochs with a learning rate of $2 \times 10^{-3}$ and dropout of 0.3. The learning rate and dropout were selected for log-scale initialization and kept unchanged across all initialization schemes.

Table A.8: **Target-frequency initialization on SHD*.** Test accuracy at the best validation epoch is reported as mean $\pm$ standard deviation over five random seeds.

| Initialization | Frozen target freq. | Learnable target freq. |
|---|---|---|
| Log-scale | $92.30 \pm 0.67\%$ | $92.98 \pm 0.40\%$ |
| Linear-scale | $91.65 \pm 1.02\%$ | $91.04 \pm 0.44\%$ |
| Constant $\sqrt{50}$ Hz | $91.08 \pm 0.83\%$ | $92.47 \pm 0.31\%$ |
| Constant 25 Hz | $57.57 \pm 1.09\%$ | $70.56 \pm 0.27\%$ |

Table A.8 shows that log-scale initialization yields the highest observed mean accuracy in both the frozen and learnable settings. Learning the target frequencies improves the mean accuracy for log-scale initialization and both constant initializations but decreases it for linear-scale initialization. Constant initialization at 25 Hz performs substantially worse than the other initializations in both settings. These results support log-scale initialization under the evaluated training configuration and show that learning target frequencies does not remove sensitivity to initialization.

## C.3 Target Frequency Perturbation

To examine whether the target frequencies learned by the FS module remain relevant to the trained computation, we perturb SSC checkpoints trained with the FS module only ($M = 0$) at inference time. For each hidden width, we evaluate 5 independently trained checkpoints. We compare the original model with two perturbations. In the reset condition, we keep all feedforward weights and other parameters fixed and restore only the target frequency parameters to their initial log-scale values. In the shuffle condition, we randomly permute the target frequency parameters within each layer.

This preserves the layer-wise target frequency distribution but breaks the assignment between target frequencies and individual neurons.

Table A.9: **Inference-time target frequency perturbation on SSC.** For each hidden width, we evaluate 5 FS-only ($M=0$) checkpoints. Each cell reports test accuracy, with the paired change from the corresponding unperturbed checkpoint shown in parentheses.

| $W$ | Full | Reset $f^\star$ | Shuffle $f^\star$ |
|---|---|---|---|
| 64 | $71.72 \pm 0.12\%$ | $20.95 \pm 6.80\%$ $(\mathbf{-50.78 \pm 6.78\%})$ | $3.69 \pm 1.13\%$ $(\mathbf{-68.03 \pm 1.12\%})$ |
| 128 | $74.94 \pm 0.16\%$ | $44.39 \pm 7.46\%$ $(\mathbf{-30.55 \pm 7.51\%})$ | $3.49 \pm 1.12\%$ $(\mathbf{-71.45 \pm 1.10\%})$ |
| 256 | $76.81 \pm 0.08\%$ | $69.86 \pm 0.72\%$ $(\mathbf{-6.95 \pm 0.74\%})$ | $4.20 \pm 0.48\%$ $(\mathbf{-72.61 \pm 0.44\%})$ |
| 512 | $77.78 \pm 0.15\%$ | $73.43 \pm 0.49\%$ $(\mathbf{-4.34 \pm 0.43\%})$ | $3.59 \pm 0.34\%$ $(\mathbf{-74.19 \pm 0.30\%})$ |

Table A.9 reports the perturbed accuracies and their changes relative to the original checkpoints. Resetting the target frequencies degrades accuracy at every hidden width. The learned target frequencies therefore remain active components of the trained computation after optimization. The degradation is larger at smaller widths, indicating that narrower networks are more sensitive to this perturbation. The shuffle condition causes a larger degradation and brings accuracy close to chance level across all widths, even though the layer-wise target frequency distribution is preserved. Thus, both the learned distribution and the assignment of target frequencies to individual neurons are coupled to the learned feedforward weights and the resulting spiking computation.

### C.4 Learned Distributions of $\beta$ and $\lambda$

To complement the layer-dependent organization analysis in Section 5.1, we visualize the learned TS parameters $\beta$ and $\lambda$ on GSC. Figure A.2 shows one representative trained checkpoint from the 20 random seeds analyzed in Section 5.1. This figure provides a qualitative view of the learned TS parameters rather than an additional aggregate statistic.

The learned $\beta$ values do not remain concentrated near zero after training. The two hidden layers form distinct distributions within the stable range $(-1, 1)$, which indicates that the all-pass stage is used for group-delay modulation rather than remaining at the initial pure-delay setting. The learned $\lambda$ values also do not collapse to a trivial endpoint. They occupy a nontrivial subset of $(0, 1)$, with the second hidden layer placing substantial mass on intermediate mixing values. These distributions indicate that the TS module uses both the direct FS pathway and the all-pass pathway after training.

These checkpoint-level distributions are consistent with the group-delay patterns reported in Figure 2. The aggregate group-delay analysis in Section 5.1 remains the main evidence for layer-dependent timing organization, while this figure shows how the underlying TS parameters are distributed in a representative trained model.

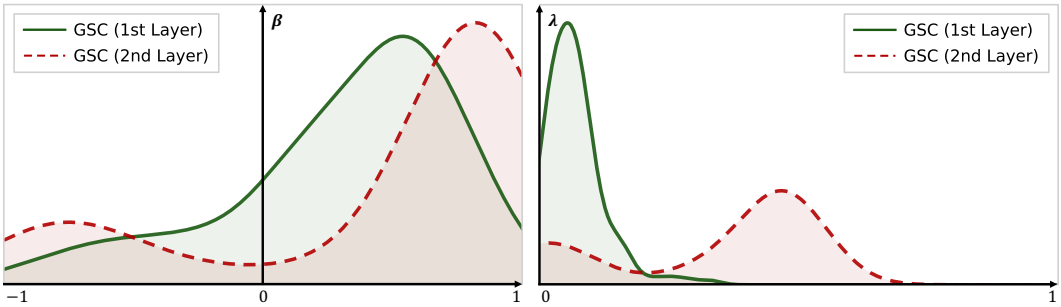



Figure A.2: **Representative learned distributions of TS parameters on GSC.** The distributions are shown for one trained checkpoint selected from the 20 random seeds analyzed in Section 5.1. Left: learned all-pass parameters $\beta \in (-1, 1)$. Right: learned mixing parameters $\lambda \in (0, 1)$.

### C.5 Additional Temporal Benchmark Results

**sMNIST.** We include sequential MNIST (sMNIST) as an additional non-auditory evaluation. Table A.10 reports a representative FiTS result together with previously reported baselines. FiTS reaches $98.28\%$ accuracy with $0.07$ M parameters using the $M=1$, width-256 model. This result is not intended to establish state-of-the-art performance on sMNIST, which primarily stresses long-range

temporal credit assignment. Rather, it shows that the same neuron-level frequency selectivity and group-delay modulation can be applied outside the auditory benchmarks considered in the main text.

Table A.10: **sMNIST results.** We compare FiTS with representative previously reported models on sequential MNIST. Parameter counts are reported as given in the corresponding references.

| Dataset | Method | # Params | Acc. |
|---|---|---|---|
| sMNIST | TC-LIF [44] | 0.09 M | 97.35% |
| | S5-RF [21] | 0.04 M | 98.89% |
| | DH-SRNN [46] | 0.08 M | 98.90% |
| | D-RF [43] | 0.16 M | 99.50% |
| | PMSN [5] | 0.16 M | 99.53% |
| | PRF [20] | 0.07 M | 99.18% |
| | **FiTS (ours, $M = 1, W = 256$)** | **0.07 M** | **98.28%** |

**DVS Gesture 128.** We also evaluate DVS Gesture 128 [1] to test FiTS on non-auditory event-based input. The goal is not to compete with state-of-the-art DVS Gesture models, but to test whether FiTS improves over plain LIF within the same lightweight feedforward convolutional architecture. Events are converted into $T = 50$ time bins with two polarity channels and resized to $64 \times 64$. The model has two convolutional spiking blocks, one fully connected spiking block, and a linear classifier. All variants use the same architecture and fixed training recipe, and differ only in the hidden-layer neuron model: plain LIF, FS-only ($M = 0$), or FS+TS ($M = 1, 2$).

For convolutional layers, FiTS parameters are shared across spatial locations within each channel. Each channel learns one target frequency and, for TS variants, one $M$-stage set of $\beta$ and $\lambda$ parameters. The fully connected hidden layer uses unit-wise FiTS parameters. Since DVS Gesture 128 has no official validation split, we use the official train/test split and report final-epoch test accuracy across 5 random seeds under a fixed training schedule. With $T = 50$ and $\Delta t = 20$ ms, we restrict the target frequency range to $[0.5, 5]$ Hz for stable FiTS updates.

Table A.11: **Non-auditory event-based check on DVS Gesture 128.** All models use the same lightweight feedforward convolutional architecture and the official train/test split. For convolutional layers, FiTS parameters are shared across spatial locations within each channel. We report final-epoch test accuracy across 5 random seeds under a fixed training schedule.

| Model | # Params | Test Acc. (%) |
|---|---|---|
| Plain LIF | 0.0726M | $87.01 \pm 1.34\%$ |
| FS-only ($M = 0$) | 0.0728M | $89.72 \pm 0.84\%$ |
| FS+TS ($M = 1$) | 0.0732M | $90.00 \pm 0.86\%$ |
| FS+TS ($M = 2$) | 0.0735M | $\mathbf{90.35 \pm 1.08}\%$ |

Table A.11 shows that FiTS improves over plain LIF under the same lightweight architecture. FS-only improves accuracy by 2.71 percentage points, and adding TS further improves the result to $90.35\%$. This provides an additional non-auditory event-based example where the proposed frequency and timing parameterization within each neuron is useful beyond the auditory benchmarks in the main text.

## 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: We highlight our contributions in both the abstract and the final paragraph of the introduction.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss limitations in the discussion, conclusion, and appendix.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: We state the assumptions in the theoretical results and provide detailed proofs for all theorems and propositions in the appendix.

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: The model, experimental settings, hyperparameters, random seeds, and compute resources are described in the paper and appendix. The official implementation is available at `https://github.com/kaistmm/FiTS`.

   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: The official code and instructions for reproducing the main experiments are available at `https://github.com/kaistmm/FiTS`. We use publicly available benchmark datasets and document their preparation in the paper and repository.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: We describe the experimental settings in the main paper 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: We report both the mean and standard deviation across random seeds for our results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: We report the computational resources in both the experiments section and appendix.

   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: We follow the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This work is foundational research on spiking neuron design, and its societal impacts are expected to be indirect.

    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: Our paper does not include high-risk models, datasets, or systems that require specific safeguards.

    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: We properly use open source assets following their license.

    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: We release the FiTS implementation at `https://github.com/kaistmm/FiTS`. The repository documents its setup, data preparation, training procedures, license, and known limitations.

    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: Our paper does not involve crowdsourcing experiments or new research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: Our paper does not involve new human-subject experiments, so IRB approval is not applicable.

    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: No LLM is used as an important, original, or non-standard component of the proposed method.

    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.