Distributed-Order Fractional Spiking Neural Network

Chengjie Ge, Yufeng Peng, Qiyu Kang, Xueyang Fu, Zheng-Jun Zha*
University of Science and Technology of China
{cjge, pengyufenguni}@mail.ustc.edu.cn
{qiyukang, xyfu, zhazj}@ustc.edu.cn
https://github.com/cjge-ustc/DF-SNN-official

Abstract

Most existing Spiking Neural Networks (SNNs) are based on *first-order* ordinary differential equations (ODEs) to model neuron potential dynamics. Recently, fractional-order spiking neural networks (f-SNNs) extend conventional SNNs by incorporating fractional-order ODEs, enabling the modeling of long-range temporal neuron potential state dependencies. However, existing first-order SNNs and f-SNNs are typically restricted to a single derivative order, which limits their capacity to represent heterogeneous temporal dynamics across different time scales. In this work, we propose Distributed-Order Fractional Spiking Neural Networks (*DF*-SNNs), which generalize single-order neuron dynamics by integrating multiple fractional orders through learnable distributed weights. This formulation enables neurons to accumulate historical information across *multiple temporal scales* within a unified framework. We develop practical numerical implementations based on the Grünwald–Letnikov discretization, allowing efficient training of *DF*-SNNs within standard SNN architectures. We show that the proposed distributed-order formulation induces intrinsic multi-scale temporal memory, providing a richer class of temporal representations compared to single-order fractional dynamics. Extensive experiments on neuromorphic classification, event-based object detection, static image classification, and graph learning demonstrate consistent performance gains of *DF*-SNNs over conventional SNNs, single-order f-SNNs under a matched discretization, and stronger temporal neurons such as P-LIF, TC-LIF, and TS-LIF, with negligible additional parameters and inference overhead.

1 Introduction

Spiking Neural Networks (SNNs) have emerged as a powerful paradigm for brain-inspired computation, offering energy efficiency and a natural aptitude for processing temporal information through discrete, event-driven communication [27, 36, 38, 56, 64, 74]. The foundation of most contemporary SNNs lies in the Integrate-and-Fire (IF) and Leaky Integrate-and-Fire (LIF) neuron models [51], whose dynamics are governed by first-order ordinary differential equations (ODEs) [26]. While computationally tractable, these integer-order models possess an inherent Markovian property: the future state of a neuron depends solely on its present state, discarding the rich historical context of its activity [56]. This limitation stands in stark contrast to empirical observations of biological neurons, which exhibit complex, non-Markovian behaviors such as long-range temporal correlations, spike-frequency adaptation, and fractal dendritic structures [37, 58]. To bridge this gap, fractional-order spiking neural networks (f-SNNs) [25] replace the first-order derivative with a Caputo fractional derivative of order $\alpha \in (0, 1]$. This endows neurons with a power-law memory kernel, allowing their current state to be influenced by the entire past trajectory and capture long-term dependencies beyond conventional SNNs.

*Corresponding author.

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

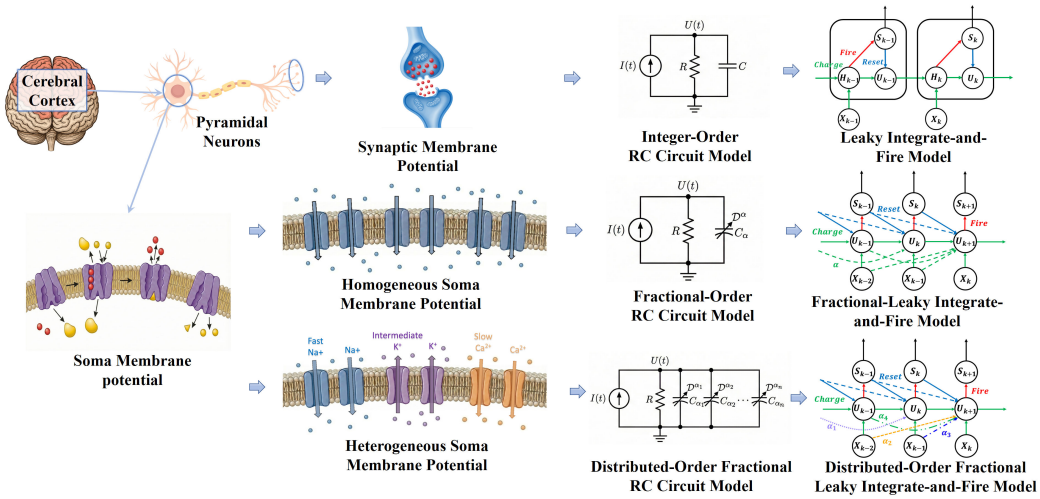


Figure 1: Biological and modeling motivations for integer-order, fractional-order, and distributed-order spiking neurons. **Left:** Illustration of biological pyramidal neurons and soma membrane potentials, highlighting homogeneous versus heterogeneous ion-channel dynamics. **Middle:** Corresponding circuit abstractions, including the standard integer-order RC model, the fractional-order RC model with power-law memory, and the distributed-order fractional RC model that aggregates multiple fractional components. **Right:** Discrete-time spiking neuron models derived from these circuits: the conventional LIF neuron, the fractional LIF (f-LIF) neuron with single-scale memory, and the distributed-order fractional LIF (*DF*-LIF) neuron that integrates membrane history across multiple temporal scales.

However, a critical constraint of existing first-order SNNs and f-SNNs is their reliance on a single, fixed order α for all neurons or across the entire network [25]. In biological systems, temporal processing is inherently heterogeneous, operating across a continuum of time scales simultaneously [5]. A single α value can only characterize one specific type of memory decay, thus limiting the representational capacity of the network to model this multi-scale temporal complexity [14]. To address this fundamental limitation, we propose Distributed-Order Fractional Spiking Neural Networks (*DF*-SNNs), a novel framework that generalizes fractional neuron dynamics by allowing each neuron to operate over a distribution of fractional orders rather than a single fixed value. As illustrated in Figure 1, this distributed-order formulation endows neurons with the ability to simultaneously integrate information across multiple temporal scales, offering a more flexible and biologically plausible memory mechanism [14].

We show that this design naturally induces multi-scale temporal memory, enabling richer representations of spatio-temporal patterns than is possible with either conventional or single-order f-SNNs. Crucially, our approach remains compatible with standard deep SNN architectures and can be trained efficiently using existing spike-based learning algorithms. In all, our primary contributions are threefold:

- We propose *DF*-SNNs, the first spiking neural network framework leveraging distributed-order fractional calculus. Unlike previous fractional models limited by temporal homogeneity, *DF*-SNNs enable diverse memory scales, providing a unified framework that encompasses existing integer and fractional-order models.
- We provide a practical and scalable implementation that enables end-to-end training within modern SNN pipelines. We complement this with a theoretical analysis highlighting the unique advantages of distributed-order dynamics over traditional SNNs.
- *DF*-SNNs consistently outperform integer-order and single-order fractional SNNs on neuromorphic classification, event-based object detection, static image classification, and graph learning. Under a matched Grünwald–Letnikov (GL) discretization and five random seeds, the gains persist across model scales, hold with non-negative distributed weights, and transfer to richer base neurons (P-LIF, TC-LIF, and TS-LIF).

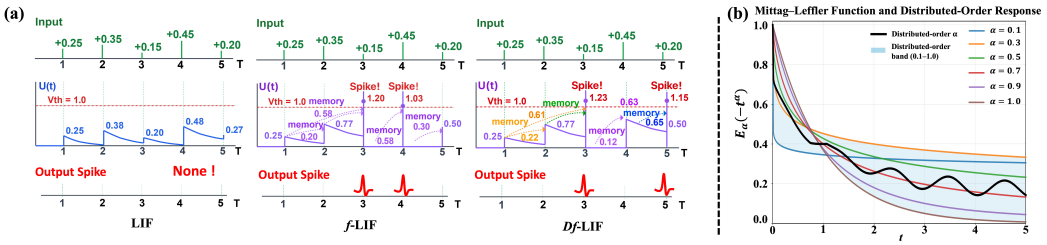


Figure 2: Illustration of single-scale and multi-scale temporal integration in spiking neurons. **(a)** Membrane potential dynamics and spike generation of LIF, f-LIF, and DF-LIF neurons under identical inputs. LIF exhibits single-scale exponential decay, f-LIF incorporates power-law memory with a single fractional order, and DF-LIF integrates historical inputs across multiple temporal scales. **(b)** Mittag–Leffler relaxation functions $E_\alpha(-t^\alpha)$ for different fractional orders, alongside a schematic distributed-order response.

2 Related Work

2.1 Spiking Neural Networks

SNNs are often regarded as the third generation of neural network models [38], drawing direct inspiration from the computational principles of biological nervous systems. Unlike Artificial Neural Networks (ANNs) that communicate via continuous-valued activations, SNNs transmit information through sparse, asynchronous spikes, which aligns with the event-driven nature of neuromorphic hardware and leads to significant energy savings [8]. The core of most SNNs is the spiking neuron model, with the LIF model being the most prevalent due to its balance between biological plausibility and computational simplicity [26, 51]. The LIF dynamics are described by a first-order ODE, $\tau \frac{\mathrm{d}U}{\mathrm{d}t} = -U + RI_{in}$, where the membrane potential U decays exponentially towards a resting state in the absence of input [57]. This exponential decay implies a Markovian process with a characteristic, fixed time constant τ, meaning the system has no "memory" beyond its current state [59]. Beyond classification, SNNs have also been applied to decision making and control, e.g., spiking policy networks whose synaptic connectivity is optimized by evolutionary search [66]. Numerous neuron variants [22, 29, 57] have been proposed—such as learnable membrane time constants (P-LIF) [20], adaptive thresholds, nonlinear dynamics, and multi-compartment models. Some of them do enrich temporal processing: TC-LIF [68] and TS-LIF [23] couple dendritic and somatic compartments to capture longer-range or frequency-specific dependencies. Nevertheless, each compartment in these models is still governed by first-order (exponential) local dynamics, so their memory is built from a small number of exponential time scales rather than the power-law, non-local memory of fractional operators [25]. Fractional neurons instead replace the first-order operator with one that is non-local in time, embedding history-dependent memory into the neuron's core dynamics, as discussed next. This change is orthogonal to the above designs: in Section 4.5, we show that our distributed-order formulation can be plugged into several of them and consistently improves their performance.

2.2 Fractional SNNs and Biological Neuron Modeling

The application of fractional calculus to neuroscience is motivated by empirical evidence of anomalous diffusion and long-memory effects in neuronal dynamics [2, 37]. At the single-neuron level, the Fractional Leaky Integrate-and-Fire (f-LIF) model, defined by $\tau D^\alpha U(t) = -U(t) + RI_{in}(t)$ where D^α is the Caputo derivative, has been shown to successfully explain phenomena like spike-frequency adaptation in pyramidal neurons [10, 58]. The solution to the f-LIF model involves the Mittag-Leffler function [45], which exhibits a slow, algebraic (power-law) decay instead of the fast exponential decay of the classical LIF, reflecting a persistent influence of past inputs [7, 15]. This property makes f-LIF neurons more robust to noisy inputs and better at generating reliable spike patterns. [13, 25, 47] The distinct spiking behaviors resulting from these different memory mechanisms are contrasted in Figure 2, which shows how LIF, f-LIF, and our proposed Distributed-Order Fractional Leaky Integrate-and-Fire (DF-LIF) neurons respond to an identical input sequence (the black distributed-order curve in panel (b) is an illustrative demo rather than an exact relaxation solution). On the network level, prior work has explored shallow fractional Hopfield-type spiking networks, focusing

on their dynamical system properties like equilibrium points and attractivity [69]. However, these studies were primarily theoretical and did not address the learning problem in deep, trainable architectures [60, 73]. The recent f-SNN framework [25] establishes an effective approach for integrating fractional dynamics into deep SNNs with end-to-end training, yielding consistent gains over integer-order models. Beyond spiking networks, distributed-order fractional operators have been introduced into continuous graph neural networks [71], where mixing several fractional orders enriches the dynamics beyond any single order. Building on these foundations, *DF*-SNN extends the single fractional order to a distribution over α, enabling the modeling of heterogeneous, multi-scale temporal processing central to biological neural computation [5].

3 *DF*-SNN Framework

We present the *DF*-SNN framework, which extends fractional spiking neural networks by endowing spiking neurons with *distributed-order* fractional dynamics. In contrast to conventional integer-order SNNs (e.g., IF/LIF) [26, 38] and single-order f-SNNs, *DF*-SNNs integrate multiple fractional orders within a unified formulation, enabling neurons to model heterogeneous temporal dependencies across multiple time scales. This distributed-order design substantially enhances neuronal expressivity beyond single-timescale dynamics.

To enable efficient training and inference, we discretize the resulting fractional dynamics following Section B.3, yielding explicit iterative update rules compatible with standard SNN architectures. A theoretical analysis in Section 3.2 shows that the induced multi-scale temporal memory leads to strictly richer neuronal representations, providing a principled explanation for the consistent performance improvements observed in our experiments.

3.1 Framework

In the *DF*-SNN framework, we further generalize fractional spiking neural networks by replacing the single fractional derivative D^α used in conventional f-SNNs (16) and (17) with a *distributed-order* fractional operator, which integrates a spectrum of fractional orders to capture heterogeneous temporal dynamics within a unified formulation. This construction enables individual neurons to accumulate and process historical information across multiple time scales, substantially enriching their temporal representational capacity beyond single-order fractional dynamics. As f-IF and f-LIF neurons are the fundamental building blocks of existing fractional SNNs, we present their distributed-order extensions as canonical instantiations of the *DF*-SNN framework. The proposed formulation is generic and can be readily extended to other fractional neuron variants. We therefore begin by introducing distributed-order f-IF and f-LIF neurons.

Distributed-order fractional IF and LIF neurons. Let $U(t)$ denote the membrane potential and $I_{\text{in}}(t)$ the input current. The distributed-order fractional dynamics of IF and LIF neurons are governed by the following integro-differential equations:

***DF*-IF neuron dynamics.**

$$\tau \int_a^b D^\alpha U(t)\, \mathrm{d}\mu(\alpha) = R\, I_{\text{in}}(t). \tag{1}$$

***DF*-LIF neuron dynamics.**

$$\tau \int_a^b D^\alpha U(t)\, \mathrm{d}\mu(\alpha) = -U(t) + R\, I_{\text{in}}(t). \tag{2}$$

where μ denotes a measure over the fractional order domain $\alpha \in (0, 1]$. Spike generation and reset follow the same rules as in the integer-order and single-order fractional cases, with spikes emitted according to $S(t) = H(U(t^-) - \theta)$. The distributed-order operator aggregates fractional dynamics of different orders, inducing multi-scale temporal dependence in the membrane potential. It *strictly generalizes* the aforementioned single-order neuron dynamics and recovers standard IF/LIF dynamics when $\mu(\alpha) = \delta(\alpha - 1)$, where δ is the Dirac delta function. It reduces to f-IF/f-LIF dynamics when $\mu(\alpha) = \delta(\alpha - \alpha_0)$ for $\alpha_0 \in (0, 1]$.

***DF*-SNN based on distributed-order fractional neuron dynamics.** To enable efficient training and inference, we discretize the distributed-order fractional dynamics in (1) and (2) following the

time-discretization procedure in Section B.3. Unlike single-order f-SNNs, the distributed-order formulation results in a multi-term fractional equation [12, 71], which we discretize using the GL scheme. By aggregating the resulting fractional kernels into a single history-dependent convolution with distributed weights, we obtain an explicit iterative update rule that is fully compatible with standard SNN architectures.

Specifically, let $\{\alpha_j\}_{j=0}^{n} \subset (0, 1]$ denote a set of discrete fractional orders for the integration of the distributed operator and $\{w_j\}_{j=0}^{n}$ the associated quadrature weights approximating the distributed-order measure μ. Here, the Caputo derivative in (1) and (2) defines the continuous-time neuron model, whereas the GL scheme is used only to derive the discrete recurrence for training and inference; the two therefore play modeling and implementation roles, respectively, rather than representing two different solvers. Applying the GL discretization to each fractional derivative and summing over all orders results in aggregated GL coefficients

$$\tilde{c}_m \;=\; \sum_{j=0}^{n} w_j h^{-\alpha_j} c_m^{(\alpha_j)}, \qquad c_m^{(\alpha_j)} = (-1)^m \binom{\alpha_j}{m}, \qquad m = 0, 1, \ldots, M-1, \tag{3}$$

where j indexes the fractional orders, m is the discrete GL lag index (the number of time steps into the past), and M is the memory-truncation length ($M = k$ when the full history up to step k is used). These coefficients encode the contributions of multiple fractional orders into a single stationary convolution kernel. More details about the discretization are given in Section C.

Parameterization of the distributed weights. To enable the measure μ to be learnable, it is equivalent and convenient to set the weights w_j to be learnable. We consider two parameterizations with learnable parameters $\{\phi_j\}$: (i) ***Softplus***, $w_j = \log(1 + e^{\phi_j}) \geq 0$, which retains a learnable overall magnitude; and (ii) ***Softmax***, $w_j = e^{\phi_j} / \sum_i e^{\phi_i}$, which additionally enforces $\sum_j w_j = 1$. Both are consistent with the non-negative measure interpretation of (1) and (2); normalization is not required by the distributed-order formulation and is only an additional constraint of (ii). Each order term adds a single learnable scalar, so *DF*-LIF adds only $n+1$ parameters per layer configuration (three in most of our settings), which is negligible relative to the backbone.

Using these coefficients, the discrete charge dynamics of distributed-order fractional neurons are given by

$$\textit{DF}\text{-IF charge:} \quad U_k = \frac{1}{\tilde{c}_0} \left(\frac{X_k}{\tau} - \sum_{m=1}^{k-1} \tilde{c}_m V_{k-m} \right), \tag{4}$$

$$\textit{DF}\text{-LIF charge:} \quad U_k = \frac{1}{\tilde{c}_0} \left(\frac{-V_{k-1} + X_k}{\tau} - \sum_{m=1}^{k-1} \tilde{c}_m V_{k-m} \right), \tag{5}$$

where X_k is the input, U_k is the charged potential, and V_k is the post-reset potential. The sum is empty at $k = 1$; all experiments use the full available history. The distributed coefficients $\{\tilde{c}_m\}_{m \geq 0}$ combine historical post-reset membrane states across multiple temporal scales.

Spike generation and reset follow the same rules as in conventional SNNs:

$$S_k = H(U_k - \theta), \tag{6}$$

$$V_k = \begin{cases} U_k - \theta S_k, & \text{(soft reset)}, \\ (1 - S_k) U_k + S_k U_{\text{reset}}, & \text{(hard reset)}. \end{cases} \tag{7}$$

Training proceeds using the surrogate-gradient method [64], which preserves discrete spike generation in the forward pass while enabling gradient-based optimization. Our detailed theoretical proof is placed in the Section D.

3.2 Theoretical Analysis

We next analyze the expressive advantages of distributed-order fractional neurons. By aggregating multiple fractional orders, the *DF*-SNN discretization constructs a memory kernel that transcends the limitations of single power-law decays. We formalize this by analyzing the properties of the aggregated kernel $\tilde{c}_m$ defined in our model.

Proposition 1 (Multi-scale Decay of Aggregated Kernels)**.** *Consider the aggregated kernel $\tilde{c}_m$ defined in* (3)*, with $h > 0$, at least two distinct orders $0 < \alpha_0 < \cdots < \alpha_N < 1$, and positive weights $w_j > 0$ (as ensured by Softplus for finite parameters). Unlike a single-order kernel $c_m^{(\alpha)}$ which obeys a strict asymptotic power-law $m^{-(1+\alpha)}$, the aggregated kernel exhibits a* time-dependent *local decay rate. Specifically, for the continuous leading-order asymptotic extension $K(m) = \sum_{j=0}^{N} w_j h^{-\alpha_j} m^{-(1+\alpha_j)}/\Gamma(-\alpha_j)$, the local log-slope $s(m) = d\ln|K(m)|/d\ln m$ varies with m and satisfies*

$$\lim_{m\to\infty} s(m) = -(1+\alpha_{\min}), \tag{8}$$

where $\alpha_{\min} = \alpha_0$; $K(m)$ describes the leading asymptotic behavior of the discrete kernel. This implies that short-term dynamics are influenced by higher orders (faster decay), while long-term memory is strictly dominated by the lowest order term (slowest decay), creating an intrinsic multi-scale representation.

Proposition 2 (Irreducibility and Linear Independence)**.** *The family of discrete power-law kernels $\mathcal{K} = \{\{m^{-(1+\alpha)}\}_{m\geq 1} \mid \alpha \in (0,1)\}$ is linearly independent. Consequently, for any aggregated kernel $\tilde{c}_m = \sum_{j=0}^{N} w_j h^{-\alpha_j} c_m^{(\alpha_j)}$ formed by at least two distinct orders with non-zero weights, there exists no single fractional order $\beta \in (0,1)$ and scalar $C \in \mathbb{R}$ such that $\tilde{c}_m = C c_m^{(\beta)}$ for all m. Thus, the temporal dynamics induced by a DF-*SNN *are strictly more expressive than, and irreducible to, those of a standard f-*SNN.

Remark 1 (Scope of the theory)**.** *Proposition 1 applies to positive-weight mixtures of at least two distinct orders in $(0,1)$, including the Softplus and Softmax parameterizations. Proposition 2 only requires non-zero weights on at least two distinct orders; it therefore also covers non-degenerate signed mixtures. Proposition 2 establishes a representational distinction between* DF-*LIF and* f-*LIF; whether this additional capacity is learnable and useful is a data- and optimization-dependent question that we examine empirically in Section 4.*

Remark 2. *Proposition 1 justifies the "dynamic gating" capability: unlike single-order models bound by a rigid memory-decay trade-off, the distributed kernel decouples timescales, employing large α_j for transients and small α_j for long-term history. Furthermore, Proposition 2 guarantees a strictly larger hypothesis space, proving that DF-*SNNs *can capture temporal dependencies mathematically inaccessible to single-order baselines. Full proofs are provided in Section D.*

4 Experiments

This section presents a systematic experimental evaluation of the proposed *DF*-SNN, i.e., spiking neural networks equipped with distributed-order fractional IF/LIF neuron dynamics. The evaluation covers four representative task settings: (1) neuromorphic event-driven classification, (2) static image classification, (3) event-based object detection, and (4) graph learning.

To ensure fairness and controlled comparisons, all experiments follow a unified evaluation protocol. Specifically, all comparisons are conducted under identical network architectures and training settings, where we only replace the LIF neuron dynamics, comparing the integer-order LIF neuron, the single-order fractional f-LIF neuron, and the proposed distributed-order *DF*-LIF neuron. In all experiments, f-LIF uses the same GL recurrence as *DF*-LIF (full history, no memory truncation), so the two differ only in the number of fractional orders. Multi-seed results are mean $\pm$ sample standard deviation: Table 1 reports three seeds, Table 2 five seeds, and Table 3 single runs.

4.1 Neuromorphic Data Classification Tasks

Datasets and setup. Neuromorphic data are acquired in an event-driven manner and are characterized by strong spatiotemporal correlations and complex, heterogeneous temporal dynamics; recent event-based vision systems exploit these properties for tasks such as RGB-event video recognition [63] and event-intensity stereo matching [72]. We evaluate the classification performance of *DF*-SNNs on multiple standard neuromorphic vision benchmarks, including N-MNIST [42], DVS128Gesture [1], N-Caltech101 [42], DVS-Lip [55], and the large-scale dataset HarDVS [62]. N-MNIST (10 digit classes) and DVS128Gesture (1,342 recordings, 11 gestures) are comparatively simple and close to saturation; N-Caltech101 (101 object classes) and DVS-Lip (lip-reading, fine-grained lip motion) are more challenging; HarDVS is a large-scale human-action benchmark with over 100,000 samples and

Table 1: Neuromorphic data classification accuracy (%, $\text{mean}_{\pm\text{std}}$ over three seeds) on multiple datasets. CNN128 denotes the SpikingJelly DVSNet backbone with 128 channels, and Spikformer256 denotes Spikformer [75] with embedding dimension 256. LIF (SJ; SpikingJelly) [21], LIF (snnTorch) [18], f-LIF [25], and *DF*-LIF are all trained with the codebase of Ge et al. [25] under the same settings. Results with larger models are given in Table 2. The best results are **boldfaced**, while the runner-ups are underlined.

Dataset	Architecture	T	LIF (SJ)	LIF (snnTorch)	f-LIF	*DF*-LIF
N-MNIST	CNN128	16	$99.09_{\pm 0.16}$	$98.58_{\pm 0.47}$	$99.34_{\pm 0.21}$	$\mathbf{99.62}_{\pm 0.09}$
DVS-Lip	CNN128	16	$41.58_{\pm 0.81}$	$32.85_{\pm 1.75}$	$43.23_{\pm 0.36}$	$\mathbf{45.28}_{\pm 1.01}$
DVS128Gesture	CNN128	16	$93.40_{\pm 0.35}$	$90.31_{\pm 1.34}$	$94.22_{\pm 0.53}$	$\mathbf{95.14}_{\pm 0.70}$
	Spikformer256	16	$94.10_{\pm 0.35}$	$88.81_{\pm 1.57}$	$94.56_{\pm 0.53}$	$\mathbf{95.95}_{\pm 0.53}$
N-Caltech101	CNN128	16	$66.38_{\pm 0.53}$	$65.63_{\pm 1.20}$	$70.76_{\pm 0.49}$	$\mathbf{71.66}_{\pm 0.15}$
	Spikformer256	16	$71.86_{\pm 0.75}$	$65.77_{\pm 0.38}$	$74.31_{\pm 1.05}$	$\mathbf{77.82}_{\pm 0.56}$
HarDVS	CNN128	8	$45.80_{\pm 0.44}$	$46.86_{\pm 0.62}$	$47.37_{\pm 0.48}$	$\mathbf{48.44}_{\pm 0.37}$
	Spikformer256	8	$45.59_{\pm 0.47}$	$46.44_{\pm 0.28}$	$47.18_{\pm 0.28}$	$\mathbf{48.49}_{\pm 0.26}$

300 classes, whose long and diverse motion patterns make it the most demanding test of temporal modeling among these datasets. We further evaluate event-based object detection on the Prophesee Gen1 automotive dataset [9] in Section 4.3. Detailed dataset statistics are given in Section E.

Except for N-MNIST, all datasets follow the standard preprocessing pipeline provided by the `SpikingJelly` framework [21], where event streams are converted into frame-based representations with a fixed number of time steps. Specifically, DVS128Gesture, N-Caltech101, and DVS-Lip are evaluated using 16 time steps, while HarDVS uses 8 time steps due to its larger scale. All input frames are resized to a spatial resolution of 128×128, and models are trained using the Adam optimizer. Additional dataset details and experimental settings are provided in Section E.

Classification performance. The classification results on neuromorphic datasets are summarized in Table 1. Across all evaluated datasets and both CNN- and Transformer-based backbones, *DF*-LIF achieves the best accuracy, followed by single-order f-LIF, while integer-order LIF yields the lowest accuracy. Fractional dynamics thus provide a stable improvement in temporal modeling, and explicitly modeling multiple temporal scales through the distributed-order formulation yields further gains on neuromorphic data with heterogeneous temporal dynamics.

Model scales. Table 2 reports five-seed results on N-Caltech101 with Spikformer at three model scales (embedding dimension 256/384/512, i.e., 2.6M/5.8M/10.3M parameters; $T = 16$, batch size 32). All neurons share the same data split, optimizer, training protocol, and seeds. The best *DF*-LIF improves over f-LIF by 3.36/4.31/4.56 points at the three scales, with both weight constraints consistently outperforming the single-order baseline. Softplus and Softmax constrain the weights to be non-negative, so the gain is attainable within the non-negative distributed-order measure covered by Proposition 1. *DF*-LIF also exhibits a smaller variance across seeds.

Table 2: Accuracy (%, mean $\pm$ sample std. over five seeds) on N-Caltech101 with Spikformer at three model scales ($T = 16$, batch size 32). The best results are **boldfaced**, while the runner-ups are underlined.

Embed. dim.	Params	LIF	f-LIF	*DF*-LIF (Softplus)	*DF*-LIF (Softmax)
256	2.6M	$68.10_{\pm 3.00}$	$73.52_{\pm 1.89}$	$\mathbf{76.88}_{\pm 0.30}$	$76.86_{\pm 0.36}$
384	5.8M	$72.12_{\pm 1.08}$	$74.38_{\pm 1.17}$	$78.41_{\pm 0.53}$	$\mathbf{78.69}_{\pm 0.44}$
512	10.3M	$73.14_{\pm 1.33}$	$75.78_{\pm 0.84}$	$\mathbf{80.34}_{\pm 0.68}$	$79.77_{\pm 0.25}$

4.2 Static Image Classification Tasks

Datasets and setup. To evaluate the applicability of the proposed approach in non-event-driven scenarios, we further conduct experiments on static image classification benchmarks, including **CIFAR-10**, **CIFAR-100** [34], and **ImageNet-1K** [35]. In this setting, static images are replicated along the temporal dimension for T time steps to construct a sequential input representation, which is a standard practice when applying SNNs to static vision tasks. All experiments use $T = 4$ time steps and adopt **Spiking-ResNet-18** [19] and **Spikformer** [75] as backbone architectures.

Classification performance. Although replicated static inputs contain no intrinsic temporal dynamics, neuron dynamics still determine how information is accumulated across time steps. As shown in Table 3, f-LIF consistently improves over LIF, and *DF*-LIF further achieves the best accuracy across datasets and architectures. With a Bernoulli *spike encoder* on ImageNet-1K, which introduces substantial temporal randomness, *DF*-LIF shows a slightly larger advantage, suggesting more robust membrane-potential evolution under stochastic spike inputs. Beyond static image classification, *DF*-LIF also consistently improves performance on event-based tasks (Sections 4.1 and 4.3). Results with larger models are in Section E.2.

Table 3: Static image classification accuracy (%, single runs, $T = 4$) with Spiking-ResNet-18 [19] on CIFAR and Spikformer [75] on ImageNet-1K (160×160 inputs). †Bernoulli spike encoder input.

Datasets	Architecture	LIF (SJ)	LIF (snnTorch)	f-LIF	*DF*-LIF
CIFAR-10	Spiking-ResNet-18	91.34	90.26	<u>91.98</u>	**92.61**
CIFAR-100	Spiking-ResNet-18	68.13	64.45	<u>68.83</u>	**69.15**
ImageNet-1K	Spikformer (16.7M)	66.37	65.84	<u>66.76</u>	**68.42**
ImageNet-1K†	Spikformer (16.7M)	55.49	54.32	<u>57.24</u>	**59.56**

4.3 Event-based Object Detection

Datasets and setup. To examine whether distributed-order dynamics remain effective on complex real-world event streams and larger networks, we evaluate object detection on the Prophesee Gen1 automotive dataset [9]. Gen1 contains 39 hours of driving recordings captured by an event camera at 304×240 resolution, with more than 255,000 bounding-box annotations of two categories (cars and pedestrians). We adopt the EMS-YOLO framework [52] with the EMS-Res10 (6.20M parameters) and EMS-Res34 (14.40M parameters) backbones and follow its original protocol: the event stream of each sample is split into $T = 5$ temporal bins with a constant window and accumulated into frame-like inputs, and the models are trained with SGD (learning rate 10^{-2}) for 100 epochs with a batch size of 128. LIF, f-LIF, and *DF*-LIF share identical data processing, training protocols, and backbones; the neuron model is the only variable and does not change the parameter count.

Detection performance. As shown in Table 4, *DF*-LIF achieves the best detection accuracy on both backbones. On EMS-Res10, it improves mAP@0.5:0.95 over f-LIF and LIF by 1.8 and 2.7 points and mAP@0.5 by 1.9 and 3.7 points. On EMS-Res34, the corresponding gains are 1.9/3.7 points (mAP@0.5:0.95) and 2.8/6.2 points (mAP@0.5). The gains grow with the deeper backbone.

Table 4: Object detection on Prophesee Gen1 with EMS-YOLO [52] ($T = 5$, single runs); only the neuron is changed. Δ: gain over LIF with the same backbone.

Backbone	Neuron	Params	mAP@0.5:0.95	Δ	mAP@0.5	Δ
EMS-Res10	LIF	6.20M	26.7	–	54.7	–
	f-LIF	6.20M	<u>27.6</u>	+0.9	<u>56.5</u>	+1.8
	DF-LIF	6.20M	**29.4**	+2.7	**58.4**	+3.7
EMS-Res34	LIF	14.40M	31.0	–	59.0	–
	f-LIF	14.40M	<u>32.8</u>	+1.8	<u>62.4</u>	+3.4
	DF-LIF	14.40M	**34.7**	+3.7	**65.2**	+6.2

Table 5: Node classification accuracy (%, mean$_{\pm\text{std}}$ over 20 runs). SpikingJelly and snnTorch denote the corresponding LIF neurons; Best results are **bolded**; second-best results are underlined.

	Method	Cora	Citeseer	PubMed	Photo	Computers	Arxiv
SGCN	SpikingJelly	$81.81_{\pm 0.69}$	$71.83_{\pm 0.23}$	$86.79_{\pm 0.32}$	$87.72_{\pm 0.25}$	$70.86_{\pm 0.24}$	$50.26_{\pm 0.11}$
	snnTorch	$83.12_{\pm 1.41}$	$71.68_{\pm 0.95}$	$59.82_{\pm 1.07}$	$83.34_{\pm 0.89}$	$74.88_{\pm 0.87}$	$21.55_{\pm 0.13}$
	f-SNN	$88.08_{\pm 0.58}$	$73.80_{\pm 0.51}$	$87.17_{\pm 0.28}$	$92.49_{\pm 0.32}$	$89.12_{\pm 0.21}$	$51.08_{\pm 0.08}$
	DF-SNN (Ours)	$\mathbf{91.48}_{\pm 0.12}$	$\mathbf{79.51}_{\pm 0.23}$	$\mathbf{87.54}_{\pm 0.11}$	$\mathbf{92.94}_{\pm 0.33}$	$\mathbf{89.17}_{\pm 0.66}$	$\mathbf{51.87}_{\pm 0.14}$
DRSGNN	SpikingJelly	$83.30_{\pm 0.64}$	$72.72_{\pm 0.24}$	$87.13_{\pm 0.34}$	$88.31_{\pm 0.15}$	$76.55_{\pm 0.17}$	$50.13_{\pm 0.14}$
	snnTorch	$80.98_{\pm 1.71}$	$68.00_{\pm 0.69}$	$59.56_{\pm 1.05}$	$82.28_{\pm 0.93}$	$76.78_{\pm 0.81}$	$28.46_{\pm 0.25}$
	f-SNN	$88.51_{\pm 0.62}$	$75.11_{\pm 0.45}$	$87.29_{\pm 0.32}$	$91.93_{\pm 0.20}$	$88.77_{\pm 0.20}$	$53.13_{\pm 0.13}$
	DF-SNN (Ours)	$\mathbf{90.01}_{\pm 0.54}$	$\mathbf{83.43}_{\pm 0.32}$	$\mathbf{88.08}_{\pm 0.05}$	$\mathbf{92.42}_{\pm 0.22}$	$\mathbf{89.11}_{\pm 0.72}$	$\mathbf{53.89}_{\pm 0.11}$

4.4 Graph Learning

DF-SNN also applies beyond vision. Following the protocol of f-SNN [25], we evaluate graph learning on two tasks: node classification and link prediction.

Datasets and baselines. Node classification is performed with two spiking graph neural networks, SGCN [76] and DRSGNN [70], on Cora [39], Citeseer [48], Pubmed [61], Amazon Photo, Amazon Computers, and ogbn-arxiv [28]. Link prediction uses the MSG framework [53] on Computers, Photo, and the CS and Physics co-authorship graphs [49]. To ensure a fair comparison, we only replace the spiking neuron modules of the baseline models with the *DF*-LIF neuron and leave all other architectural and training components unchanged, so all variants have the same backbone and differ only in the neuron dynamics. We compare against SpikingJelly [21], snnTorch [18], and f-SNN [25]. Dataset statistics are given in Section E.

Experimental setup. For node classification we use Poisson spike encoding with 100–128 simulation time steps, and the data are split into training/validation/test sets with ratio $0.7/0.2/0.1$. For DRSGNN, the positional-encoding dimension is 32–128, using Laplacian (LSPE) [17] or random-walk (RWPE) [16] encodings. For link prediction we follow the original MSG configuration (IF neurons with the Lorentz dynamics model, representation dimension 32, 16 time steps, Adam with learning rate 0.01) and report AUC. All results are the mean and standard deviation over 20 independent runs. Other settings are given in Section E.

Node classification. As shown in Table 5, *DF*-SNN achieves the best accuracy on all six datasets with both backbones. Compared with f-SNN, the gains are largest on the more challenging citation graphs: 3.40/5.71 points on Cora/Citeseer with SGCN, and 1.50/8.32 points on Cora/Citeseer with DRSGNN. On the larger Photo, Computers, and ogbn-arxiv graphs the improvements are smaller but consistent (0.05–0.79 points). Compared with the integer-order SpikingJelly baseline, *DF*-SNN improves accuracy by up to 18.31 points (SGCN on Computers). These results indicate that mixing several fractional orders gives graph neurons a richer temporal memory than a single order.

Link prediction. *DF*-SNN also obtains the highest AUC on all four datasets, with the largest gains on CS (97.37 vs. 96.53) and Physics (97.23 vs. 96.57); the full results, including snnTorch baselines, are listed in Section E.

4.5 Computational Efficiency and Generality

We further examine what distributed-order memory costs in practice in terms of throughput and GPU memory, and whether it also benefits neuron models richer than LIF.

Computational Efficiency. Fractional neurons pay for long-range memory with a history convolution at every step, and *DF*-LIF combines several orders; we examine whether this adds a practical cost. On HarDVS, we train Spikformer-256 ($T = 8$, batch size 64) with LIF, f-LIF, and *DF*-LIF with fixed or learnable weights, and report training and test throughput, peak GPU memory, and accuracy over five runs (Table 6). Fixed-weight *DF*-LIF matches the test throughput and peak memory of f-LIF

Table 6: Efficiency and accuracy on HarDVS (Spikformer-256, $T = 8$, five runs, one RTX 4090).

Neuron	Train FPS	Test FPS	Mem. (GB)	Acc. (%)
LIF	2193.0	3760.0	8.02	45.37
f-LIF	1661.6	3326.6	10.06	46.47
DF-LIF (fixed w_j)	1649.0	3330.8	10.08	<u>47.28</u>
DF-LIF (Softplus)	1365.4	3352.0	14.05	**48.38**

while improving accuracy by 0.81 points. Learning $\{w_j\}$ with Softplus further raises accuracy to 48.38% (+1.91 over f-LIF, +3.01 over LIF); the additional gradient computation for the coefficients lowers training throughput and increases memory, while test throughput is unaffected. Overall, the practical cost of distributed-order memory is dominated by the history convolution already present in f-LIF, rather than by the number of combined orders.

Table 7: Five-seed accuracy (%) with richer base neurons on N-Caltech101 (Spikformer-256, $T = 16$). Δ: gain of *DF* over the single-order fractional variant; FPS drop: relative test-throughput reduction of *DF* vs. the fractional variant. *DF* variants use Softplus weights.

Base neuron	Base	Fractional	*DF* (ours)	Δ	FPS drop
LIF	$68.10_{\pm 3.00}$	$73.52_{\pm 1.89}$	$\mathbf{76.88}_{\pm 0.30}$	+3.36	1.6%
P-LIF [20]	$67.64_{\pm 0.68}$	$72.65_{\pm 0.56}$	$\mathbf{76.42}_{\pm 0.49}$	+3.77	0.7%
TC-LIF [68]	$23.66_{\pm 1.01}$	$57.23_{\pm 0.65}$	$\mathbf{66.99}_{\pm 0.38}$	+9.76	1.0%
TS-LIF [23]	$62.33_{\pm 0.77}$	$67.38_{\pm 1.63}$	$\mathbf{71.25}_{\pm 0.61}$	+3.87	4.3%

Richer temporal neurons. Distributed-order memory is not tied to the LIF neuron. Following the same procedure used for LIF, we extend P-LIF [20], which learns its membrane time constant, and the two-compartment neurons TC-LIF [68] and TS-LIF [23]. For the two-compartment models we identify a coupled continuous-time system whose unit-step semi-implicit Euler discretization exactly recovers the native recurrence, and replace only its first-order derivatives with fractional ones, keeping all other mechanisms unchanged (Section E.4). Under the same Spikformer, data, training protocol, and five seeds (Table 7), distributed-order memory improves over the single-order fractional variant by 3.36–9.76 points for all four base neurons, while reducing test throughput by only 0.7%–4.3%. The largest gain appears on TC-LIF, whose native recurrence reaches only 23.66% in this setting: fractional memory lifts it to 57.23%, and distributed-order memory adds a further 9.76 points. Distributed-order memory also reduces the standard deviation across seeds, e.g., from ± 1.89 to ± 0.30 for LIF. Full results are given in Table 11 (Section E.3).

5 Limitation and Future Work

Our implementation shares one distributed-order configuration across layers, with fixed orders and learnable weights. Future work will explore layer-specific and learnable orders. Full-history GL convolution incurs increasing per-step cost as sequences grow; short-memory approximations warrant evaluation on longer sequences. Further directions include extending *DF* to additional neuron models and measuring energy consumption on neuromorphic hardware.

6 Conclusion

We propose Distributed-Order Fractional Spiking Neural Networks (*DF*-SNNs), which combine multiple fractional orders with learnable weights to model multi-scale temporal memory. Theoretically, *DF*-SNNs encompass integer-order and single-order fractional models and yield memory kernels that a single fractional order cannot reproduce. Experiments show consistent gains across neuromorphic and static image classification, event-based detection, and graph learning. The consistent improvements across richer neuron models further highlight the value of multi-scale memory for spiking architectures across diverse learning tasks.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62225207, 62436008, 62422609, 62576326 and GG2100060211, in part by the CAS Project for Young Scientists in Basic Research under Grant YSBR-122, and in part by the Fundamental Research Funds for the Central Universities under Grant WK2102026001. The authors declare no competing interests.

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, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

[2] Thomas J Anastasio. The fractional-order dynamics of brainstem vestibulo-oculomotor neurons. *Biological cybernetics*, 72(1):69–79, 1994.

[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. *Advances in neural information processing systems*, 31, 2018.

[4] Jan Benda. Neural adaptation. *Current Biology*, 31(3):R110–R116, 2021.

[5] György Buzsáki. *The brain from inside out*. Oxford University Press, 2019.

[6] Michele Caputo. Linear models of dissipation whose q is almost frequency independent—ii. *Geophysical journal international*, 13(5):529–539, 1967.

[7] Jiejie Chen, Zhigang Zeng, and Ping Jiang. Global mittag-leffler stability and synchronization of memristor-based fractional-order neural networks. *Neural Networks*, 51:1–8, 2014.

[8] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[9] Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive. *arXiv preprint arXiv:2001.08499*, 2020.

[10] Yabin Deng, Bijing Liu, Zenan Huang, Xiaojie Liu, Shan He, Qiuhong Li, and Donghui Guo. Fractional spiking neuron: Fractional leaky integrate-and-fire circuit described with dendritic fractal model. *IEEE Transactions on Biomedical Circuits and Systems*, 16(6):1375–1386, 2022.

[11] Kai Diethelm. *The analysis of fractional differential equations: an application-oriented exposition using differential operators of Caputo type*, volume 2004. Lect. Notes Math., 2010.

[12] Kai Diethelm and Neville J Ford. Numerical analysis for distributed-order differential equations. *J. Comput. Appl. Math.*, 225(1):96–104, 2009.

[13] Jianhao Ding, Zhaofei Yu, Jian K. Liu, and Tiejun Huang. Neuromorphic computing paradigms enhance robustness through spiking neural networks. *Nature Communications*, 16(1), 2025.

[14] Wei Ding, Sansit Patnaik, Sai Sidhardh, and Fabio Semperlotti. Applications of distributed-order fractional operators: A review. *Entropy*, 23(1):110, 2021.

[15] Zhixia Ding, Yi Shen, and Leimin Wang. Global mittag-leffler synchronization of fractional-order neural networks with discontinuous activations. *Neural Networks*, 73:77–85, 2016.

[16] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

[17] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

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

[19] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34:21056–21069, 2021.

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

[21] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023.

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

[23] Shibo Feng, Wanjin Feng, Xingyu Gao, Peilin Zhao, and Zhiqi Shen. TS-LIF: A temporal segment spiking neuron network for time series forecasting. In *International Conference on Learning Representations (ICLR)*, 2025.

[24] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022.

[25] Chengjie Ge, Yufeng Peng, Zihao Li, Qiyu Kang, Xueyang Fu, Xuhao Li, Qixin Zhang, Junhao Ren, and Zheng-Jun Zha. Fractional-order spiking neural network. In *International Conference on Learning Representations*, 2026.

[26] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[27] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[28] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133, 2020.

[29] 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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 19949–19972, 2024.

[30] Murat Isik, Karn Tiwari, Muhammed Burak Eryilmaz, and I Can Dikmen. Accelerating sensor fusion in neuromorphic computing: A case study on loihi-2. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2024.

[31] Qiyu Kang, Kai Zhao, Qinxu Ding, Feng Ji, Xuhao Li, Wenfei Liang, Yang Song, and Wee Peng Tay. Unleashing the potential of fractional calculus in graph neural networks with FROND. In *Proc. International Conference on Learning Representations*, Vienna, Austria, 2024.

[32] Shujaat Khan, Jawwad Ahmad, Imran Naseem, and Muhammad Moinuddin. A novel fractional gradient-based learning algorithm for recurrent neural networks. *Circuits, Systems, and Signal Processing*, 37(2):593–612, 2018.

[33] Christof Koch, Moshe Rapp, and Idan Segev. A brief history of time (constants). *Cerebral cortex*, 6(2):93–101, 1996.

[34] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[36] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[37] Brian N Lundstrom, Matthew H Higgs, William J Spain, and Adrienne L Fairhall. Fractional differentiation by neocortical pyramidal neurons. *Nature neuroscience*, 11(11):1335–1342, 2008.

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

[39] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

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

[41] Gabriel Nobis, Maximilian Springenberg, Marco Aversa, Michael Detzel, Rembert Daems, Roderick Murray-Smith, Shinichi Nakajima, Sebastian Lapuschkin, Stefano Ermon, Tolga Birdal, Manfred Opper, Christoph Knochenhauer, Luis Oala, and Wojciech Samek. Generative fractional diffusion models. In *Advances in Neural Information Processing Systems*, volume 37, pages 25469–25509, 2024.

[42] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9: 437, 2015.

[43] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.*, 41(4):A2603–A2626, 2019.

[44] Paul F. Pinsky and John Rinzel. Intrinsic and network rhythmogenesis in a reduced Traub model for CA3 neurons. *Journal of Computational Neuroscience*, 1(1):39–60, 1994.

[45] Igor Podlubny. *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications*, volume 198 of *Mathematics in Science and Engineering*. Academic Press, 1999.

[46] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. In *CVPR*, pages 3857–3866, 2019.

[47] Ran Rubin, L. F. Abbott, and Haim Sompolinsky. Balanced excitation and inhibition are required for high-capacity, noise-robust neuronal selectivity. *Proceedings of the National Academy of Sciences*, 114(44), 2017. doi: 10.1073/pnas.1705841114.

[48] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[49] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learn. Workshop, Advances Neural Inf. Process. Syst.,*, 2018.

[50] Yeonjong Shin, Jérôme Darbon, and George Em Karniadakis. Accelerating gradient descent and adam via fractional gradients. *Neural Networks*, 161:185–201, 2023.

[51] Richard B Stein. Some models of neuronal variability. *Biophysical journal*, 7(1):37–68, 1967.

[52] Qiaoyi Su, Yuhong Chou, Yifan Hu, Jianing Li, Shijie Mei, Ziyang Zhang, and Guoqi Li. Deep directly-trained spiking neural networks for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6555–6565, 2023.

[53] Li Sun, Zhenhao Huang, Qiqi Wan, Hao Peng, and Philip S. Yu. Spiking graph neural network on riemannian manifolds. In *Advances in Neural Information Processing Systems*, volume 37, pages 34025–34055, 2024.

[54] OI Tacha, JM Munoz-Pacheco, E Zambrano-Serrano, IN Stouboulos, and V-T Pham. Determining the chaotic behavior in a fractional-order finance system with negative parameters. *Nonlinear Dynamics*, 94(2):1303–1317, 2018.

[55] Ganchao Tan, Yang Wang, Han Han, Yang Cao, Feng Wu, and Zheng-Jun Zha. Multi-grained spatio-temporal features perceived network for event-based lip-reading. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20094–20103, 2022.

[56] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

[57] Corinne Teeter, Ramakrishnan Iyer, Vilas Menon, Nathan Gouwens, David Feng, Jim Berg, Aaron Szafer, Nicholas Cain, Hongkui Zeng, Michael Hawrylycz, Christof Koch, and Stefan Mihalas. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature Communications*, 9(1):709, 2018. doi: 10.1038/s41467-017-02717-4.

[58] Wondimu Teka, Toma M Marinov, and Fidel Santamaria. Neuronal spike timing adaptation described with a fractional leaky integrate-and-fire model. *PLoS computational biology*, 10(3): e1003526, 2014.

[59] Anton Trushechkin. Long-term behaviour in an exactly solvable model of pure decoherence and the problem of markovian embedding. *Mathematics*, 12(1):1, 2023.

[60] Gal Vardi. On the implicit bias in deep-learning algorithms. *Communications of the ACM*, 66 (6):86–93, 2023.

[61] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019.

[62] Xiao Wang, Zongzhen Wu, Bo Jiang, Zhimin Bao, Lin Zhu, Guoqi Li, Yaowei Wang, and Yonghong Tian. Hardvs: Revisiting human activity recognition with dynamic vision sensors. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 5615–5623, 2024.

[63] Xiao Wang, Yao Rong, Shiao Wang, Yuan Chen, Zhe Wu, Bo Jiang, Yonghong Tian, and Jin Tang. Unleashing the power of CNN and transformer for balanced RGB-event video recognition. *Machine Intelligence Research*, 22(6):1031–1047, 2025. doi: 10.1007/s11633-025-1555-3.

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

[65] Chengting Yu, Zheming Gu, Da Li, Gaoang Wang, Aili Wang, and Erping Li. Stsc-snn: Spatio-temporal synaptic connection with temporal convolution and attention for spiking neural networks. *Frontiers in Neuroscience*, 16:1079357, 2022.

[66] Duzhen Zhang, Tielin Zhang, Shuncheng Jia, Qingyu Wang, and Bo Xu. Tuning synaptic connections instead of weights by genetic algorithm in spiking policy network. *Machine Intelligence Research*, 21(5):906–918, 2024. doi: 10.1007/s11633-023-1481-1.

[67] Jiqing Zhang, Malu Zhang, Yuanchen Wang, Qianhui Liu, Baocai Yin, Haizhou Li, and Xin Yang. Spiking neural networks with adaptive membrane time constant for event-based tracking. *IEEE Transactions on Image Processing*, 2025.

[68] Shimin Zhang, Qu Yang, Chenxiang Ma, Jibin Wu, Haizhou Li, and Kay Chen Tan. TC-LIF: A two-compartment spiking neuron model for long-term sequential modelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16838–16847, 2024.

[69] Shuo Zhang, Lu Liu, Chunhua Wang, Xiaomeng Zhang, and Rong Ma. Multistability and global attractivity for fractional-order spiking neural networks. *Applied Mathematics and Computation*, 508:129617, 2026.

[70] Han Zhao, Xu Yang, Cheng Deng, and Junchi Yan. Dynamic reactive spiking graph neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16970–16978, 2024.

[71] Kai Zhao, Xuhao Li, Qiyu Kang, Feng Ji, Qinxu Ding, Yanan Zhao, Wenfei Liang, and Wee Peng Tay. Distributed-order fractional graph operating network. In *Advances in Neural Information Processing Systems*, volume 37, pages 103442–103475, Vancouver, Canada, Dec. 2024.

[72] Qinghao Zhao and Jiangtao Xu. Multi-modal fusion and wavelet feature extraction for event-intensity stereo matching. *Machine Intelligence Research*, 23(3):662–676, 2026. doi: 10.1007/s11633-025-1567-z.

[73] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11): 3212–3232, 2019.

[74] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11062–11070, 2021.

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

[76] Zulun Zhu, Jiaying Peng, Jintang Li, Liang Chen, Qi Yu, and Siqiang Luo. Spiking graph convolutional networks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pages 2434–2440, 2022.

A More Related Works

A.1 Fractional Calculus on Deep Learning and Training

Beyond neuron modeling, fractional calculus has found other applications in the broader field of deep learning. One line of research focuses on using fractional operators to enhance the expressivity of continuous ANNs. For instance, fractional graph neural networks have been developed to improve performance and robustness on graph-structured data [31], and fractional diffusion processes have been used in generative models to promote diversity [41]. Another prominent area is the development of Physics-Informed Neural Networks (PINNs) for solving fractional partial differential equations (f-PINNs), which treat the neural network as a universal function approximator to find solutions to given physical laws [43]. A distinct but related approach involves using fractional derivatives to define novel gradient operators for the training process itself. Methods like fractional gradient descent replace the standard integer-order update rule with a fractional counterpart, which can smooth the optimization landscape and help escape local minima [32, 50]. It is critical to distinguish our work from these approaches. Unlike f-PINNs, which solve an external equation, our *DF*-SNN embeds the fractional dynamics as the core computational engine of the neuron itself. Furthermore, unlike fractional gradient methods that alter the optimizer, our work changes the network architecture's fundamental dynamics. Our goal is not to solve a PDE or to design a new optimizer, but to create a new class of spiking neurons with intrinsically richer, multi-scale temporal memory capabilities.

A.2 Event Camera

Event cameras are bio-inspired vision sensors that break away from the traditional frame-based paradigm. Instead of capturing redundant, synchronous frames at a fixed rate, they operate asynchronously: each pixel independently monitors its local brightness and emits an event—a tuple (x, y, t, p)—only when the log intensity change exceeds a threshold. Here, (x, y) denotes the pixel location, t is a high-precision timestamp (typically with microsecond resolution), and $p \in \{+1, -1\}$ indicates the polarity of the change (brightening or dimming) [24]. This novel sensing mechanism endows event cameras with several remarkable properties: extremely high temporal resolution, a dynamic range exceeding 140 dB, negligible motion blur, and inherent data sparsity [24]. These advantages make them particularly suitable for high-speed, low-latency, and high-dynamic-range visual perception tasks in robotics, autonomous driving, and augmented reality. The unique data format of event cameras has spurred significant research interest. Early foundational work established their potential for optical flow estimation and visual odometry [24]. More recently, deep learning approaches have been adapted to process event streams. For instance, methods like E2VID [46] focus on reconstructing intensity images from events, while others directly perform recognition or tracking on the raw event data. To better leverage the temporal structure of events, specialized SNN architectures have been proposed. Notably, Yu et al. [65] introduced the Spatio-Temporal Saliency Coding SNN (STSC-SNN), which enhances feature extraction by explicitly modeling spatial and temporal saliency in the event stream. Furthermore, Isik et al. [30] demonstrated the practical efficiency of processing event data on neuromorphic hardware like Intel's Loihi 2 chip, achieving substantial power savings over conventional platforms. In our work, we adopt this asynchronous event stream as the primary input. The events are directly converted into spikes and fed into our *DF*-SNN. This approach capitalizes on the natural synergy between the event-driven sensor and the event-driven processor, while the distributed-order fractional dynamics of our model provide a powerful mechanism to capture the complex, multi-scale temporal correlations present in real-world event data.

B Preliminaries

This section outlines the foundational components required for fractional-order spiking neural networks. We first introduce fractional calculus as a mathematical framework for modeling non-local temporal dynamics, followed by the Grünwald–Letnikov discretization for its numerical solution. Finally, we revisit single-term f-SNNs, detailing how fractional-order dynamics are incorporated into standard spiking neuron architectures.

B.1 Fractional Calculus

Let $y(t)$ denote a real-valued function defined with respect to time t. In classical calculus, the first-order derivative of $y(t)$ is defined as the instantaneous rate of change,

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} \coloneqq \lim_{\Delta t \to 0} \frac{y(t+\Delta t) - y(t)}{\Delta t}. \tag{9}$$

While this local definition is sufficient for modeling memoryless dynamics, it is inadequate for systems whose evolution depends on their historical states.

Fractional calculus generalizes classical differentiation by extending the order of differentiation to non-integer values, thereby providing a principled framework for modeling temporal memory and non-local dependencies [45].

Definition 1 (Fractional Derivative)**.** *Let $y(t)$ be defined on an interval $[0, T]$. The Caputo fractional derivative of order $\alpha \in (0, 1]$ is defined as [6, 11]*

$$D^{\alpha} y(t) \coloneqq \frac{1}{\Gamma(1-\alpha)} \int_0^t (t-\tau)^{-\alpha}\, y'(\tau)\, \mathrm{d}\tau, \tag{10}$$

where $y'(\tau)$ denotes the first-order derivative of $y(\tau)$.

For $\alpha \in (0, 1)$, the Caputo derivative incorporates the entire history of $y(t)$ through a power-law kernel, endowing the system with long-range temporal memory. As $\alpha \to 1$, the Caputo derivative continuously recovers the classical first-order derivative.

An integer-order ordinary differential equation and its fractional-order counterpart can be expressed as

$$\text{integer-order ODE:} \quad \frac{\mathrm{d}y(t)}{\mathrm{d}t} = f(t, y(t)), \tag{11}$$

$$\text{fractional-order ODE:} \quad D^{\alpha} y(t) = f(t, y(t)), \tag{12}$$

where $f(\cdot)$ specifies the system dynamics and the initial condition $y(0) = y_0$ is defined identically in both cases.

B.2 Grünwald–Letnikov Discretization

To numerically handle fractional-order differential equations, a common approach is to discretize the non-local fractional derivative into a tractable finite-difference form. Among existing methods, the Grünwald–Letnikov (GL) [45, 54] formulation provides a simple and widely used discretization.

Given a time-dependent function $y(t)$, the fractional derivative $D^{\alpha} y(t)$ can be approximated in the Grünwald–Letnikov sense as [54]:

$$D^{\alpha} y(t) \approx \frac{1}{h^{\alpha}} \sum_{j=0}^{\lfloor t/h \rfloor} (-1)^j \binom{\alpha}{j} y(t - jh), \tag{13}$$

where h denotes the temporal step size and $\binom{\alpha}{j}$ is the generalized binomial coefficient.

For practical implementation, the coefficients $(-1)^j \binom{\alpha}{j}$ are represented by a sequence of Grünwald–Letnikov weights $\{c_j^{(\alpha)}\}$, which can be computed recursively as

$$c_0^{(\alpha)} = 1, \qquad c_j^{(\alpha)} = \left(1 - \frac{1+\alpha}{j}\right) c_{j-1}^{(\alpha)}, \quad j \geq 1. \tag{14}$$

On a uniform temporal grid $t_k = kh$ with $y_k \approx y(t_k)$, the GL discretization yields an explicit time-stepping scheme in which the current state depends on a weighted sum of all past states:

$$y_{k+1} = h^{\alpha} f(t_k, y_k) - \sum_{j=1}^{k+1} c_j^{(\alpha)}\, y_{k+1-j}, \tag{15}$$

This formulation makes the non-Markovian memory effect of fractional dynamics explicit and provides a convenient foundation for incorporating fractional-order neuron models into spiking neural networks.

B.3 Single-order Fractional Spiking Neuron and f-SNN

Existing SNN models [3, 4, 33, 67] predominantly describe neuronal membrane-potential dynamics using discretized first-order ODEs with derivative $\mathrm{d}/\,\mathrm{d}t$, including the widely adopted IF and LIF neurons [26, 51]. To incorporate long-range temporal memory into spiking neurons, the work [25] replaces the first-order derivative $\mathrm{d}/\,\mathrm{d}t$ with a fractional-order derivative D^α and obtains the following fractional neuron dynamics:

$$f\text{-IF neuron:} \quad \tau\, D^\alpha U(t) = R\, I_{\mathrm{in}}(t), \tag{16}$$

$$f\text{-LIF neuron:} \quad \tau\, D^\alpha U(t) = -U(t) + R\, I_{\mathrm{in}}(t), \tag{17}$$

where $U(t)$ denotes the membrane potential, $I_{\mathrm{in}}(t)$ the input current, $\tau > 0$ the membrane time constant, and $R > 0$ the membrane resistance. Spike generation and reset follow the same mechanism as in the integer-order case: $S(t) = H(U(t^-) - \theta)$, where $H(\cdot)$ denotes the Heaviside step function and θ is the firing threshold. Upon spiking, either a soft or hard reset is applied. The fractional order α controls the strength of temporal memory: $\alpha = 1$ recovers the standard IF/LIF models, whereas $\alpha < 1$ induces power-law memory and enhanced temporal correlations in the membrane potential.

In this paper, the single-order and distributed-order neurons use the same GL discretization. Let $t_k = kh$ with $h > 0$, set $R = 1$, and assume the zero initial state $V_0 = 0$. We denote the charged potential by U_k, the post-reset potential by V_k, and the presynaptic input by X_k. Using the coefficients $c_m^{(\alpha)} = (-1)^m \binom{\alpha}{m}$ from (14), the full-history charge updates are

$$f\text{-IF charge:} \quad U_k = h^\alpha \frac{X_k}{\tau} - \sum_{m=1}^{k-1} c_m^{(\alpha)} V_{k-m}, \tag{18}$$

$$f\text{-LIF charge:} \quad U_k = h^\alpha \frac{-V_{k-1} + X_k}{\tau} - \sum_{m=1}^{k-1} c_m^{(\alpha)} V_{k-m}. \tag{19}$$

The history sum is empty at $k = 1$. Spike generation and reset are

$$S_k = H(U_k - \theta), \tag{20}$$

$$V_k = \begin{cases} U_k - \theta S_k, & \text{soft reset,} \\ (1 - S_k)U_k + S_k U_{\mathrm{reset}}, & \text{hard reset.} \end{cases}$$

For the integer-order case $\alpha = 1$, the GL coefficients satisfy $c_0^{(1)} = 1$, $c_1^{(1)} = -1$, and $c_m^{(1)} = 0$ for $m \geq 2$. The same GL formulation therefore reduces to the standard explicit Euler charge updates

$$\text{IF:} \quad U_k = V_{k-1} + hX_k/\tau,$$

$$\text{LIF:} \quad U_k = V_{k-1} + h(-V_{k-1} + X_k)/\tau.$$

Thus, integer-order LIF, single-order f-LIF, and distributed-order *DF*-LIF share a consistent discretization and reset convention. Under the surrogate-gradient approach [40, 64], the forward pass uses the hard threshold, while backpropagation uses a smooth surrogate.

C Numerical Discretization for Distributed Operator

We consider the distributed-order system governed by the following integro-differential equation:

$$\int_a^b D^\alpha y(t)\, \mathrm{d}\mu(\alpha) = f(t, y(t)), \tag{21}$$

where μ represents a measure over the fractional order α (which may be parameterized as a learnable distribution). A standard numerical approach approximates this integral using a quadrature rule, resulting in a multi-term fractional differential equation (FDE):

$$\sum_{j=0}^{n} w_j D^{\alpha_j} y(t) = f(t, y(t)), \tag{22}$$

subject to the initial condition $y(0) = y_0$. Here, w_j denotes the discretization weights (coefficients) and $\alpha_j \in (0, 1)$ represents the discrete fractional orders.

We work on a uniform grid $t_k = kh$ with step size $h > 0$. For each fractional order α_j we approximate the fractional derivative at t_k by the GL formula (13):

$$D^{\alpha_j} y(t_k) \approx h^{-\alpha_j} \sum_{m=0}^{k} c_m^{(\alpha_j)}\, y_{k-m}, \qquad k = 1, \ldots, N. \tag{23}$$

Substituting this into the multi-term FDE

$$\sum_{j=0}^{n} w_j D^{\alpha_j} y(t) = f(t, y(t)),$$

and evaluating at t_k with an explicit right-hand side $f(t_{k-1}, y_{k-1})$ gives

$$\sum_{j=0}^{n} w_j h^{-\alpha_j} \sum_{m=0}^{k} c_m^{(\alpha_j)} y_{k-m} = f(t_{k-1}, y_{k-1}), \qquad k = 1, \ldots, N. \tag{24}$$

Interchanging the sums, we obtain a single convolution with *distributed GL weights*

$$\sum_{m=0}^{k} \tilde{c}_m\, y_{k-m} = f(t_{k-1}, y_{k-1}), \qquad \tilde{c}_m := \sum_{j=0}^{n} w_j h^{-\alpha_j} c_m^{(\alpha_j)}, \quad m \geq 0. \tag{25}$$

Using $c_0^{(\alpha_j)} = 1$ for all j from (14), we have

$$\tilde{c}_0 = \sum_{j=0}^{n} w_j h^{-\alpha_j} > 0, \tag{26}$$

so we can solve for y_k as

$$y_k = \frac{1}{\tilde{c}_0} \left(f(t_{k-1}, y_{k-1}) - \sum_{m=1}^{k} \tilde{c}_m\, y_{k-m} \right), \qquad k = 1, 2, \ldots, N. \tag{27}$$

Equivalently, reindexing to align with the notation in (15), we write

$$y_{k+1} = \frac{1}{\tilde{c}_0} \left(f(t_k, y_k) - \sum_{m=1}^{k+1} \tilde{c}_m\, y_{k+1-m} \right), \qquad k = 0, 1, \ldots, N-1. \tag{28}$$

In particular, when μ collapses to a single atom (i.e., $n = 0$, $w_0 = 1$, $\alpha_0 = \alpha$), we have $\tilde{c}_m = h^{-\alpha} c_m^{(\alpha)}$ and (28) reduces to the single-order GL scheme in Section B.2 (up to the trivial scaling by $\tilde{c}_0 = h^{-\alpha}$).

- **short memory version:** As in the single-order case, we may truncate the convolution to a finite memory length M and obtain a short-memory distributed GL solver. For $k + 1 \leq M$, (28) is unchanged. For $k + 1 > M$, we keep only the M most recent history terms, which yields

$$y_{k+1} = \frac{1}{\tilde{c}_0} \left(f(t_k, y_k) - \sum_{m=1}^{\min\{k+1, M\}} \tilde{c}_m\, y_{k+1-m} \right) \tag{29}$$

$$= \frac{1}{\tilde{c}_0} \left(f(t_k, y_k) - \sum_{j=\max\{0, k+1-M\}}^{k} \tilde{c}_{k+1-j}\, y_j \right), \qquad k = 0, 1, \ldots, N-1, \tag{30}$$

  which mirrors the short-memory GL scheme but with the aggregated weights $\{\tilde{c}_m\}_{m \geq 0}$ encoding the distributed-order measure μ through the quadrature coefficients $\{(w_j, \alpha_j)\}_{j=0}^{n}$.

D Proofs for Theoretical Analysis

D.1 Preliminaries

Grünwald–Letnikov Discretization. To numerically solve fractional-order differential equations, it is necessary to discretize the non-local fractional derivative in a tractable form. Among existing approaches, the GL formulation provides a direct finite-difference approximation that naturally exposes the temporal memory structure induced by fractional dynamics.

GL definition and difference scheme. For a time-dependent function $y(t)$, the fractional operator D^α of order $\alpha \in (0,1)$ is approximated by the Grünwald–Letnikov limit [54]:

$$D^\alpha y(t) = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{j=0}^{\lfloor t/h \rfloor} (-1)^j \binom{\alpha}{j} y(t-jh). \tag{S.1}$$

Let $\{t_k\}_{k\geq 0}$ be a uniform temporal grid with step size h, and denote $y_k \approx y(t_k)$. By truncating the limit, we obtain the discretized difference equation for the system dynamics $D^\alpha y(t) = f(t, y(t))$:

$$\sum_{j=0}^{k} c_j^{(\alpha)}\, y_{k-j} = h^\alpha f(t_{k-1}, y_{k-1}), \tag{S.2}$$

where $c_j^{(\alpha)} := (-1)^j \binom{\alpha}{j}$ are the GL coefficients. Solving for the current state y_k yields the explicit iterative scheme:

$$y_{k+1} = h^\alpha f(t_k, y_k) - \sum_{j=1}^{k+1} c_j^{(\alpha)}\, y_{k+1-j}. \tag{S.3}$$

Derivation of the recursive coefficients. Direct computation of binomial coefficients for large j is numerically unstable. We derive the recursive relation used in (S.3) as follows. By definition:

$$\binom{\alpha}{j} = \frac{\alpha(\alpha-1)\cdots(\alpha-j+1)}{j!}. \tag{S.4}$$

Considering the ratio of consecutive terms:

$$\begin{aligned} \frac{c_j^{(\alpha)}}{c_{j-1}^{(\alpha)}} &= \frac{(-1)^j \binom{\alpha}{j}}{(-1)^{j-1}\binom{\alpha}{j-1}} = -\frac{\alpha(\alpha-1)\cdots(\alpha-j+1)/j!}{\alpha(\alpha-1)\cdots(\alpha-j+2)/(j-1)!} \\ &= -\frac{\alpha-j+1}{j} = \frac{j-\alpha-1}{j} = 1 - \frac{1+\alpha}{j}. \end{aligned} \tag{S.5}$$

Thus, starting with $c_0^{(\alpha)} = 1$, the weights can be computed efficiently via $c_j^{(\alpha)} = (1 - \frac{1+\alpha}{j}) c_{j-1}^{(\alpha)}$.

Gamma function representation and proofs. To analyze the memory properties, we express $c_j^{(\alpha)}$ using the Gamma function. Recall the property of the generalized binomial coefficient:

$$\binom{\alpha}{j} = \frac{\Gamma(\alpha+1)}{\Gamma(j+1)\Gamma(\alpha-j+1)}. \tag{S.6}$$

Using the reflection formula or the property $\Gamma(z+1) = z\Gamma(z)$, we can rewrite the numerator terms of the binomial expansion. Observe that:

$$\begin{aligned} c_j^{(\alpha)} &= (-1)^j \frac{\alpha(\alpha-1)\cdots(\alpha-j+1)}{j!} \\ &= \frac{(-\alpha)(1-\alpha)\cdots(j-1-\alpha)}{j!} \\ &= \frac{\Gamma(j-\alpha)/\Gamma(-\alpha)}{\Gamma(j+1)}. \end{aligned} \tag{S.7}$$

Here, we utilized the identity $\Gamma(j-\alpha) = (j-1-\alpha)\cdots(-\alpha)\Gamma(-\alpha)$. Thus, we arrive at the closed form:

$$c_j^{(\alpha)} = \frac{\Gamma(j-\alpha)}{\Gamma(-\alpha)\Gamma(j+1)}. \tag{S.8}$$

Asymptotic analysis (Stirling's approximation). We now determine the decay rate of the memory kernel as $j \to \infty$. Stirling's approximation states that for large x, $\Gamma(x+b) \sim \sqrt{2\pi} x^{x+b-1/2} e^{-x}$. A useful corollary for the ratio of Gamma functions is:

$$\lim_{j\to\infty} \frac{\Gamma(j+a)}{\Gamma(j+b)} \cdot j^{b-a} = 1. \tag{S.9}$$

Applying this to (S.8) with $a = -\alpha$ and $b = 1$:

$$\begin{aligned} c_j^{(\alpha)} &= \frac{1}{\Gamma(-\alpha)} \cdot \frac{\Gamma(j-\alpha)}{\Gamma(j+1)} \\ &\sim \frac{1}{\Gamma(-\alpha)} \cdot j^{-\alpha-1} \qquad \text{as } j \to \infty. \end{aligned} \tag{S.10}$$

Since $\alpha \in (0,1)$, the term $\Gamma(-\alpha)$ is negative, confirming that $c_j^{(\alpha)}$ is eventually negative. The decay rate $j^{-(1+\alpha)}$ indicates a heavy-tailed memory distribution, characteristic of fractional dynamics.

Generating functions and Aggregated Kernels. The sequence $c_j^{(\alpha)}$ corresponds to the coefficients of the Taylor expansion of the function $(1-z)^\alpha$:

$$\sum_{j=0}^{\infty} c_j^{(\alpha)} z^j = \sum_{j=0}^{\infty} (-1)^j \binom{\alpha}{j} z^j = (1-z)^\alpha, \quad |z| < 1. \tag{S.11}$$

In our *DF*-SNN discretization, we extend this by aggregating distinct fractional orders $\{\alpha_n\}_{n=1}^N$ with learnable weights $\{w_n\}_{n=1}^N$:

$$\tilde{c}_j := \sum_{n=1}^{N} w_n h^{-\alpha_n} c_j^{(\alpha_n)}. \tag{S.12}$$

To quantify the multi-scale nature of this aggregated kernel, we define the *local log-slope*:

$$s(j) := \frac{d \ln |\tilde{c}_j|}{d \ln j}. \tag{S.13}$$

For a single order, the asymptotic relation (S.10) implies $\ln |c_j^{(\alpha)}| \approx -(1+\alpha) \ln j + \text{const}$, yielding a constant slope $s(j) \approx -(1+\alpha)$. The aggregated kernel $\tilde{c}_j$, however, enables a time-varying $s(j)$, allowing the model to capture complex decay patterns beyond a single power law.

D.2 Proof of Proposition 1 (Multi-scale Decay)

Proof. We analyze the local log-slope $s(m)$ of the leading-order continuous extension of $\tilde{c}_m$ and its limit for the smallest fractional order.

1. Asymptotic expansion of the aggregated kernel. Let $N \geq 1$, $h > 0$, $w_j > 0$, and let the distinct fractional orders satisfy $0 < \alpha_0 < \alpha_1 < \cdots < \alpha_N < 1$. Recall the asymptotic behavior of the single-order GL coefficient derived in the main text:

$$c_m^{(\alpha_j)} = \frac{1}{\Gamma(-\alpha_j)} m^{-(1+\alpha_j)} \left(1 + \mathcal{O}(m^{-1})\right), \quad \text{as } m \to \infty. \tag{S.14}$$

Substituting this into the definition of the aggregated kernel $\tilde{c}_m = \sum_{j=0}^N w_j h^{-\alpha_j} c_m^{(\alpha_j)}$, we obtain:

$$\tilde{c}_m = \sum_{j=0}^{N} \frac{w_j h^{-\alpha_j}}{\Gamma(-\alpha_j)} m^{-(1+\alpha_j)} + \text{h.o.t.}, \tag{S.15}$$

where h.o.t. denotes higher-order terms. Assuming $w_0 \neq 0$, we factor out the term corresponding to the smallest order $\alpha_{\min} = \alpha_0$:

$$\tilde{c}_m = \frac{w_0 h^{-\alpha_0}}{\Gamma(-\alpha_0)} m^{-(1+\alpha_0)} \left[1 + \sum_{j=1}^{N} \frac{w_j h^{-\alpha_j} \Gamma(-\alpha_0)}{w_0 h^{-\alpha_0} \Gamma(-\alpha_j)} m^{-(\alpha_j - \alpha_0)}\right]. \tag{S.16}$$

2. Derivation of the local log-slope. For the leading-order continuous extension $K(m)$ in the proposition, the local log-slope is $s(m) = \frac{d\ln|K(m)|}{d\ln m} = \frac{mK'(m)}{K(m)}$. Treating m as a continuous variable for the asymptotic analysis, let $K_j = \frac{w_j h^{-\alpha_j}}{\Gamma(-\alpha_j)}$. The derivative of the j-th term $K_j m^{-(1+\alpha_j)}$ is $-(1+\alpha_j)K_j m^{-(2+\alpha_j)}$. The derivative of the sum is approximately:

$$\frac{d\tilde{c}_m}{dm} \approx -\sum_{j=0}^{N}(1+\alpha_j)K_j m^{-(2+\alpha_j)}. \tag{S.17}$$

Substituting this into the expression for $s(m)$:

$$\begin{aligned} s(m) &\approx \frac{m}{\sum_{j=0}^{N} K_j m^{-(1+\alpha_j)}} \left(-\sum_{j=0}^{N}(1+\alpha_j)K_j m^{-(2+\alpha_j)} \right) \\ &= -\frac{\sum_{j=0}^{N}(1+\alpha_j)K_j m^{-(1+\alpha_j)}}{\sum_{j=0}^{N} K_j m^{-(1+\alpha_j)}}. \end{aligned} \tag{S.18}$$

3. Limit analysis ($m \to \infty$). We divide both the numerator and denominator by the dominant term $m^{-(1+\alpha_0)}$ (corresponding to the slowest decay):

$$s(m) \approx -\frac{(1+\alpha_0)K_0 + \sum_{j=1}^{N}(1+\alpha_j)K_j m^{-(\alpha_j-\alpha_0)}}{K_0 + \sum_{j=1}^{N} K_j m^{-(\alpha_j-\alpha_0)}}. \tag{S.19}$$

Since we ordered $\alpha_j > \alpha_0$ for all $j \geq 1$, the exponent $-(\alpha_j - \alpha_0)$ is strictly negative. Consequently, as $m \to \infty$, the terms $m^{-(\alpha_j-\alpha_0)}$ vanish. The limit is therefore:

$$\lim_{m\to\infty} s(m) = -\frac{(1+\alpha_0)K_0}{K_0} = -(1+\alpha_0) = -(1+\alpha_{\min}). \tag{S.20}$$

4. Non-constancy and multi-scale nature. For any finite m, the terms $m^{-(\alpha_j-\alpha_0)}$ are non-zero. Unless $w_j = 0$ for all $j \neq 0$ (which degenerates to the single-order case), the ratio defining $s(m)$ depends explicitly on m. Physically, for small m, terms with larger α_j (faster decay) contribute significantly, making the slope magnitude $|s(m)|$ larger. As m increases, these fast-decaying components vanish relative to the α_0 component, causing the slope to relax toward $-(1+\alpha_0)$. This time-varying slope confirms the intrinsic multi-scale property of the kernel. □

D.3 Proof of Proposition 2 (Irreducibility)

Proof. This proposition asserts that the aggregated kernel cannot be represented by a single fractional order. We prove this by establishing the linear independence of GL kernels.

Part 1: Linear Independence. We first show that the set of sequences $\mathcal{K} = \{\{c_m^{(\alpha)}\}_{m\geq 1} \mid \alpha \in (0,1)\}$ is linearly independent. Consider a finite linear combination of distinct orders $0 < \alpha_0 < \cdots < \alpha_N < 1$ that sums to zero:

$$\sum_{j=0}^{N} w_j c_m^{(\alpha_j)} = 0, \quad \forall\, m \geq 1. \tag{S.21}$$

Assume for the sake of contradiction that not all w_j are zero. Let k be the index of the smallest order with a non-zero weight, i.e., $w_k \neq 0$. Multiply the equation by $m^{1+\alpha_k}$ and take the limit $m \to \infty$. Using the asymptotic property $\lim_{m\to\infty} c_m^{(\alpha)} m^{1+\alpha} = \frac{1}{\Gamma(-\alpha)}$, we analyze the sum:

$$\lim_{m\to\infty} \sum_{j=k}^{N} w_j \frac{c_m^{(\alpha_j)}}{m^{-(1+\alpha_k)}} = \lim_{m\to\infty} \left(w_k \frac{c_m^{(\alpha_k)}}{m^{-(1+\alpha_k)}} + \sum_{j=k+1}^{N} w_j \frac{c_m^{(\alpha_j)}}{m^{-(1+\alpha_k)}} \right) = 0. \tag{S.22}$$

- The first term converges to $w_k \frac{1}{\Gamma(-\alpha_k)}$.

- For any $j > k$, we have $\alpha_j > \alpha_k$. Thus, $c_m^{(\alpha_j)}$ decays faster than $m^{-(1+\alpha_k)}$, meaning the ratio $c_m^{(\alpha_j)}/m^{-(1+\alpha_k)} \sim m^{-(\alpha_j-\alpha_k)}$ approaches 0.

The limit equation simplifies to $w_k/\Gamma(-\alpha_k) = 0$. Since the Gamma function is finite for $\alpha_k \in (0, 1)$, this implies $w_k = 0$, contradicting our assumption. By induction, all weights w_j must be zero, proving linear independence.

Part 2: Irreducibility via Generating Functions. We now prove that an aggregated kernel $\tilde{c}_m$ with at least two non-zero weights cannot be equivalent to any single kernel $Kc_m^{(\beta)}$. We utilize the generating function (Z-transform) identity $\sum_{m=0}^{\infty} c_m^{(\alpha)} z^m = (1-z)^{\alpha}$. If $\tilde{c}_m = Kc_m^{(\beta)}$ for all m, their generating functions must be identical for $|z| < 1$:

$$\sum_{j=0}^{N} w_j (1-z)^{\alpha_j} = K(1-z)^{\beta}. \tag{S.23}$$

Let $u = 1 - z$. As $z \to 1$, $u \to 0$. The condition becomes $\sum w_j u^{\alpha_j} = Ku^{\beta}$ near $u = 0$. Assume α_0 is the smallest order with $w_0 \neq 0$. Dividing both sides by u^{α_0}:

$$w_0 + w_1 u^{\alpha_1 - \alpha_0} + \cdots + w_N u^{\alpha_N - \alpha_0} = Ku^{\beta - \alpha_0}. \tag{S.24}$$

Taking the limit $u \to 0$:

- The LHS converges to w_0 (since all exponents $\alpha_j - \alpha_0 > 0$).
- The RHS converges to K if $\beta = \alpha_0$, 0 if $\beta > \alpha_0$, and ∞ if $\beta < \alpha_0$.

For the equality to hold, we must have $\beta = \alpha_0$ and $K = w_0$. Substituting this back into (S.23) yields:

$$\sum_{j=1}^{N} w_j (1-z)^{\alpha_j} = 0. \tag{S.25}$$

By the linear independence proved in Part 1, this requires $w_j = 0$ for all $j \geq 1$. This contradicts the premise that the aggregated kernel has at least two non-zero components. Thus, the dynamics of a *DF*-SNN are strictly irreducible to those of a single-order f-SNN. □

E Additional Experimental Details and Results

E.1 Neuromorphic Data Classification Tasks

Datasets and baselines. We evaluate the proposed method on five representative neuromorphic visual classification benchmarks: N-MNIST [42], DVS128Gesture [1], N-Caltech101 [42], DVS-Lip [55], and the large-scale HarDVS dataset [62]. These datasets span a wide range of task complexities and temporal characteristics, providing a comprehensive testbed for assessing the effectiveness of distributed-order spiking neuron dynamics.

N-MNIST is a neuromorphic adaptation of the classical MNIST dataset, where handwritten digit images are recorded by a Dynamic Vision Sensor (DVS) under controlled camera motions, producing event streams corresponding to ten digit classes. DVS128Gesture is designed for dynamic gesture recognition and is collected using an event-based camera with microsecond-level temporal resolution. It contains 1,342 samples across 11 gesture categories, capturing fine-grained spatiotemporal motion patterns. N-Caltech101 extends the static Caltech101 dataset into the neuromorphic domain by recording event streams generated from various camera motions over static object images, resulting in event-based representations of 101 object categories. DVS-Lip focuses on visual speech recognition and leverages the high temporal sensitivity of event cameras to capture subtle lip movements. HarDVS is a large-scale event-based human action recognition dataset comprising over 100,000 samples and 300 activity classes, posing significant challenges in both temporal modeling and scalability. Prophesee Gen1 [9] is an automotive event-based object-detection dataset recorded with an ATIS sensor (304×240) while driving, with bounding-box annotations for cars and pedestrians; we use it for the detection experiments in Section 4.3 following the EMS-YOLO protocol [52].

To ensure fair and controlled comparisons, we adopt two commonly used backbone architectures—CNN-based and Transformer-based SNNs—and vary only the spiking neuron dynamics across different methods. Specifically, baseline models employ the `neuron.LIFNode` from the SpikingJelly framework [21] and the `snn.leaky` neuron from snnTorch [18]. In contrast, our approach replaces the standard neuron modules with the proposed distributed-order fractional neuron dynamics, instantiated using the *DF*-LIF formulation described in (5). For the CNN-based setting, we follow the DVSNet architecture provided by SpikingJelly, while the Transformer-based experiments are conducted using Spikformer [75] as the backbone. For the additional comparisons in Section 4.5, P-LIF, TC-LIF, and TS-LIF are implemented following Fang et al. [20], Zhang et al. [68], and Feng et al. [23], respectively.

Training and inference settings. For the N-MNIST dataset, we use a batch size of 512 and simulate the network for $T = 16$ timesteps. Models are optimized with the Adam optimizer and trained for 100 epochs.

For the remaining neuromorphic datasets, we adopt the standard preprocessing pipeline provided by the SpikingJelly framework, where raw event streams are converted into frame-based representations. The temporal resolution is set to 16 timesteps for DVS128Gesture, N-Caltech101, and DVS-Lip, while HarDVS is evaluated with 8 timesteps to balance computational cost and performance. For N-Caltech101, the dataset is split into training and test sets with an 80%/20% ratio. For the remaining neuromorphic datasets, models are trained with a batch size of 16 for CNN128 and 32 for Spikformer256 (64 for the five-run HarDVS experiments in Section 4.5), and input frames are uniformly resized to a spatial resolution of 128×128. For these datasets, CNN128 models are trained for 400 epochs using the Adam optimizer. Spikformer256 models are trained for 500 epochs on DVS-Lip and HarDVS and for 1000 epochs on DVS128Gesture and N-Caltech101, under the same optimization settings.

Computational resources. All experiments are run on single NVIDIA RTX 4090 GPUs (24 GB). Table 8 reports the GPU hours of one training run, including training and a full test pass after every epoch.

Table 8: Epochs and GPU hours (single RTX 4090) of one training run, including a full test pass per epoch.

Dataset	T	CNN128	Spikformer256
HarDVS	8	400 ep., 84–90 h	500 ep., ≈93 h
DVS128Gesture	16	400 ep., ≈1.6 h	1000 ep., ≈2.1 h
N-Caltech101	16	400 ep., ≈10.7 h	1000 ep., ≈15.3 h
DVS-Lip	16	400 ep., ≈22.8 h	500 ep., ≈13.7 h

The software environment is Python 3.9.23, PyTorch 2.7.1 (CUDA 12.6), CuPy 13.6.0, and timm 0.5.4. Each GPU is paired with 8 logical CPU cores, with 4 data-loading workers and one computation thread. All runs use the CuPy implementation of the *DF*-LIF neuron with full history and automatic mixed precision (AMP), with $T = 8$ on HarDVS and $T = 16$ on the other datasets.

Robustness analysis. Fractional-order spiking neural networks are known to exhibit improved robustness over conventional integer-order SNNs due to their intrinsic non-Markovian dynamics [25]. Building on this observation, we further evaluate the robustness of the proposed *DF*-SNNs by comparing them against single-order f-SNNs and traditional SNN baselines under a range of realistic perturbations.

As illustrated in Figure 3, *DF*-SNNs consistently achieve performance that is comparable to, and in several cases slightly better than, single-order f-SNNs across all evaluated robustness settings, while clearly outperforming integer-order SNN baselines. In particular, under **temporal truncation**, both *DF*-SNNs and f-SNNs exhibit graceful degradation as the available temporal context is reduced, indicating strong resilience to incomplete temporal information. Under **noise injection**, the two fractional models maintain stable performance even at higher noise levels, whereas traditional SNNs suffer from pronounced accuracy drops. Similarly, in the presence of **central occlusions**, *DF*-SNNs

preserve robustness comparable to f-SNNs, demonstrating effective integration of partial spatial information.

Overall, these results suggest that introducing distributed-order, multi-scale temporal dynamics does not compromise the inherent robustness of fractional neuron models. Instead, *DF*-SNNs retain the stability advantages of f-SNNs while offering increased modeling flexibility through multi-scale temporal integration.

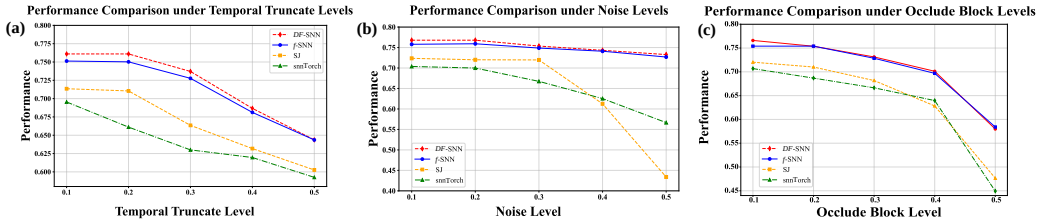


Figure 3: Robustness comparison under different perturbation settings on N-Caltech101 with Spikformer (embedding dimension 256, $T = 16$). (a) Temporal truncation, (b) Gaussian noise injection, and (c) central occlusion. *DF*-SNNs exhibit robustness comparable to single-order f-SNNs across all perturbations, while consistently outperforming integer-order SNN baselines.

E.2 Static Image Classification on ImageNet-1K

Model scales. The gains persist with larger models. With the 29.7M-parameter Spikformer-8-512 [75] on ImageNet-1K ($T = 4$, Table 9), *DF*-LIF improves over LIF and f-LIF by 1.45 and 0.73 points, and with a longer horizon of $T = 8$ it again outperforms both baselines under repeated-input and spike-encoder settings (Table 10).

Table 9: ImageNet-1K top-1 accuracy (%, single runs) with the 29.7M-parameter Spikformer-8-512 ($T = 4$).

Neuron	Top-1	Params
LIF	73.38	29.7M
f-LIF	74.10	29.7M
DF-LIF	**74.83**	29.7M

Table 10: ImageNet-1K top-1 accuracy (%, single runs) with $T = 8$ (Spikformer, 16.7M).

Setting	LIF (SJ)	LIF (snnTorch)	f-LIF	*DF*-LIF
Repeated input	65.42	65.21	66.80	**67.34**
Spike encoder	53.84	53.75	56.34	**57.65**

E.3 Richer Temporal Neurons

Table 11 lists the full five-seed results for the richer temporal neurons summarized in Table 7, including training and test throughput, peak GPU memory, and the Euler-update controls for TC-LIF and TS-LIF. The construction of the fractional and distributed-order extensions is detailed in Section E.4. In Tables 2, 6, 7 and 11, the LIF baseline uses the same `LIFNeuronFDE` implementation, with its distributed-order update replaced by a first-order Euler update.

E.4 Distributed-order Extensions of Multi-compartment Neurons

TC-LIF [68] and TS-LIF [23] are originally presented as discrete-time dual-compartment recurrences. For each model we identify a coupled continuous-time system whose structure-preserving discretization recovers the native recurrence, and then replace only its first-order derivatives with fractional- or distributed-order derivatives. To avoid a clash with the fractional orders α_j used in the main text, we denote the compartment decay factors by λ_1, λ_2 and the fractional orders by ν (single order) or $\{\nu_k\}_{k=1}^{K}$ with weights $\{w_k\}$ (distributed order).

Table 11: Five-seed comparison with richer temporal neurons on N-Caltech101 (Spikformer, embedding dim. 256, $T = 16$, batch size 32). *DF* variants use Softplus weights unless stated. Accuracy is averaged over five runs; FPS and peak GPU memory are reported from efficiency benchmarks.

Base	Variant	Train / Test FPS	Mem. (GB)	Acc. (%)
LIF	LIF	3240.3 / 4575.2	8.16	68.10
	f-LIF	2455.5 / 3661.4	10.98	73.52
	DF-LIF (Softplus)	1869.3 / 3601.4	12.94	**76.88**
	DF-LIF (Softmax)	1878.8 / 3604.2	12.95	76.86
P-LIF	P-LIF	2958.4 / 3987.1	8.67	67.64
	f-P-LIF	2276.1 / 3511.6	11.35	72.65
	DF-P-LIF	1785.8 / 3488.1	12.88	**76.42**
TC-LIF	Native discrete update	2015.3 / 3804.2	14.21	23.66
	Euler update	2167.6 / 3972.2	14.36	25.17
	f-TC-LIF	1269.8 / 2531.7	17.43	57.23
	DF-TC-LIF	1007.2 / 2506.4	20.67	**66.99**
TS-LIF	Native discrete update	1892.3 / 3361.4	14.56	62.33
	Euler update	2053.1 / 3927.5	13.93	63.54
	f-TS-LIF	1111.7 / 2080.3	16.35	67.38
	DF-TS-LIF	983.2 / 1991.8	19.51	**71.25**

TC-LIF. Inspired by the dendritic–somatic structure of the Pinsky–Rinzel model [44], TC-LIF uses the recurrence

$$\begin{aligned} U^D[t] &= U^D[t-1] + \beta_1 U^S[t-1] + I[t] - \gamma S[t-1], \\ U^S[t] &= U^S[t-1] + \beta_2 U^D[t] - V_{\mathrm{th}} S[t-1], \qquad S[t] = \Theta\big(U^S[t] - V_{\mathrm{th}}\big), \end{aligned} \tag{S.26}$$

where β_1, β_2 are coupling coefficients of opposite signs and γ controls the partial dendritic reset. Excluding the reset, its continuous-time counterpart is

$$\frac{\mathrm{d}u^D}{\mathrm{d}t} = \beta_1 u^S + x(t), \qquad \frac{\mathrm{d}u^S}{\mathrm{d}t} = \beta_2 u^D. \tag{S.27}$$

A semi-implicit (symplectic) Euler step of size h, which evaluates the somatic increment at the already-updated dendritic state and carries the reset over from the previous spike s_t, gives $u^D_{t+1} = u^D_t + h(\beta_1 u^S_t + x_{t+1}) - \gamma s_t$ and $u^S_{t+1} = u^S_t + h\beta_2 u^D_{t+1} - V_{\mathrm{th}} s_t$, followed by $s_{t+1} = \Theta(u^S_{t+1} - V_{\mathrm{th}})$. At $h = 1$ this reproduces (S.26) term by term. The single-order (f-TC-LIF) and distributed-order (*DF*-TC-LIF) extensions are

$$D^\nu_t u^D = \beta_1 u^S + x, \quad D^\nu_t u^S = \beta_2 u^D; \qquad \sum_{k=1}^{K} w_k D^{\nu_k}_t u^D = \beta_1 u^S + x, \quad \sum_{k=1}^{K} w_k D^{\nu_k}_t u^S = \beta_2 u^D. \tag{S.28}$$

TS-LIF. TS-LIF adds leaky dynamics to both compartments, injects the input into both, and lets both compartments spike:

$$\begin{aligned} U^D[t] &= \lambda_1 U^D[t-1] + \beta_1 U^S[t-1] + (1-\lambda_1) I[t] - \gamma_1 S^D[t-1], \\ U^S[t] &= \lambda_2 U^S[t-1] + \beta_2 U^D[t] + (1-\lambda_2) I[t] - \gamma_2 S^S[t-1], \\ S^{\mathrm{mix}}[t] &= \kappa\, \Theta(U^D[t] - V_{\mathrm{th}}) + (1-\kappa)\, \Theta(U^S[t] - V_{\mathrm{th}}). \end{aligned} \tag{S.29}$$

Its continuous-time counterpart (excluding resets) is

$$\frac{\mathrm{d}u^D}{\mathrm{d}t} = (\lambda_1 - 1)u^D + \beta_1 u^S + (1-\lambda_1)x, \qquad \frac{\mathrm{d}u^S}{\mathrm{d}t} = (\lambda_2 - 1)u^S + \beta_2 u^D + (1-\lambda_2)x, \tag{S.30}$$

and the same unit-step semi-implicit Euler ordering recovers the native charge update, including its one-step-delayed resets. f-TS-LIF and *DF*-TS-LIF are obtained by replacing $\frac{\mathrm{d}}{\mathrm{d}t}$ with D^ν_t and $\sum_k w_k D^{\nu_k}_t$, respectively.

Implementation and controls. The dendritic and somatic potentials are maintained as two solver states, and the GL history convolution is applied separately to each state. In all variants, the compartment coupling, input injection, firing, reset, and spike read-out are unchanged; only the temporal derivative operator differs. The Euler variants serve as integer-order controls for the solver interface: in Table 11 they reach accuracy comparable to the native recurrences (differences of 1.51 and 1.21 points for TC-LIF and TS-LIF), whereas the single-order fractional and distributed-order extensions yield large additional gains. The slightly higher throughput of the Euler variants results from the `torch.compile` acceleration used in our solver wrapper rather than from a change in neuronal dynamics. The same construction applies to any neuron whose internal states and right-hand-side dynamics are explicitly defined.

E.5 Graph Learning Tasks

To rigorously evaluate the efficacy of the proposed method, we conducted comprehensive experiments on standard graph learning benchmarks. Following the experimental protocol established in f-SNNs [25], we integrated our approach into both node classification and link prediction tasks across widely adopted graph datasets.

Datasets and baselines. For node classification, our experiments were performed on two representative Spiking Graph Neural Network (SGNN) architectures: SGCN [76] and DRSGNN [70]. Six benchmark datasets were employed: Cora [39], Citeseer [48], Pubmed [61], Amazon Photo, Amazon Computers, and OGBN-Arxiv [28]. For link prediction, we adopted the Message Passing Spiking Graph (MSG) framework [53] and evaluated performance on the Amazon Computers and Photo datasets, as well as the co-authorship graphs CS and Physics from the Coauthor suite [49]. Detailed datasets information can be found in Table 12.

Table 12: Dataset statistics of the graph learning tasks.

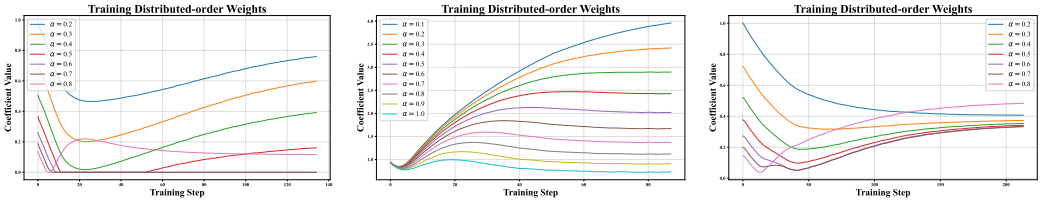
Name	**# of Nodes**	**# of Classes**	**# of Features**	**# of Edges**
Cora	2,708	7	1,433	10,556
Pubmed	19,717	3	500	88,648
Citeseer	3,327	6	3,703	9,104
Photo	7,650	8	745	238,162
Computers	13,752	10	767	491,722
Arxiv	169,343	40	128	1,166,243
CS	18,333	15	6,805	163,788
Physics	34,493	5	8,415	495,924

To ensure a fair comparison, we only replaced the integer- or fractional-order spiking neuron modules in the baseline models with our proposed distributed-order fractional spiking neuron module, leaving all other architectural and training components unchanged. Specifically, we employ the same backbone network and conduct comparative experiments using SpikingJelly [21], snnTorch [18], f-SNN [25], and our proposed *DF* neuron, respectively.

Experimental setup. Poisson spike encoding was adopted for all node classification experiments. The number of simulation timesteps N was set within the range $[100, 128]$, and batch sizes varied between 32 and 512 depending on dataset scale. Consistent with prior work [25], each dataset was partitioned into training, validation, and test sets using a 7:2:1 split ratio. For DRSGNN, positional encoding dimensions ranged from 32 to 128, employing either Laplacian Spectral Positional Encoding (LSPE) [17] or Random Walk Positional Encoding (RWPE) [16]. All reported results are averaged over 20 independent runs, with standard deviations included to reflect statistical reliability.

For link prediction, we followed the experimental configuration of the original MSG paper [53]: IF neurons coupled with the Lorentz dynamics model were used; the representation dimension was fixed at 32; the number of timesteps was set to 16; and the Adam optimizer was employed with an initial learning rate of 0.01. Again, results are reported as mean ± standard deviation over 20 trials.

Across all tasks, we set the multi-term fractional orders α as $\{0.1, \ldots, 1.0\}$ or $\{0.2, \ldots, 0.8\}$, and their associated coefficients were treated as learnable parameters initialized via logarithmic scaling.



(a) Distributed-order weights for Arxiv (b) Distributed-order weights for Photo (c) Distributed-order weights for Pubmed

Figure 4: Evolution of learnable distributed-order weights during training across three representative datasets. Each curve corresponds to the coefficient assigned to a specific fractional order $\alpha \in [0.2, 0.8]$ (or $[0.1, 1.0]$ for Photo). The smooth and convergent trajectories demonstrate that the proposed framework stably learns a data-adaptive mixture of fractional dynamics. Notably, different datasets exhibit distinct weight distributions, indicating that the model automatically emphasizes the most informative temporal scales for each task.

As illustrated in Figure 4, the framework automatically adapts these weights during gradient-based training, effectively learning an optimal composition of fractional dynamics tailored to each dataset.

Experimental results. As summarized in Table 5, our proposed *DF*-SNN consistently achieves state-of-the-art performance across diverse datasets and architectural frameworks. On the Cora and Citeseer datasets under the SGCN backbone, *DF*-SNN improves absolute accuracy by up to 3.4% and 5.7%, respectively, compared to the previous best f-SNN. More strikingly, on Citeseer with DRSGNN, *DF*-SNN attains 83.43% accuracy—surpassing f-SNN by more than 8 percentage points—highlighting the enhanced representational capacity afforded by distributed-order modeling. In particular, on the large-scale Arxiv dataset, *DF*-SNN achieves performance comparable to f-SNN, demonstrating its scalability and robustness even in challenging, high-dimensional settings.

Table 13: Link prediction performance measured by Area Under the ROC Curve (AUC) (%) across multiple datasets. Best results are **bolded**; second-best results are underlined. Standard deviations are reported as subscripts.

	Method	Computers	Photos	CS	Physics
MSG	SpikingJelly	$94.65_{\pm 0.73}$	$96.75_{\pm 0.18}$	$95.19_{\pm 0.15}$	$93.43_{\pm 0.16}$
	f-SNN	$94.91_{\pm 0.12}$	$96.80_{\pm 0.16}$	$96.53_{\pm 0.15}$	$96.57_{\pm 0.08}$
	DF-SNN (Ours)	$\mathbf{95.02}_{\pm 0.42}$	$\mathbf{96.80}_{\pm 0.09}$	$\mathbf{97.37}_{\pm 0.11}$	$\mathbf{97.23}_{\pm 0.06}$

Furthermore, as shown in Table 13, *DF*-SNN demonstrates consistent superiority in link prediction. It achieves the highest AUC scores across all four datasets, with particularly notable improvements on the CS (+0.84%) and Physics (+0.66%) co-author graphs. These results confirm that the distributed-order formulation not only benefits node-level tasks but also enhances relational inference by capturing richer temporal correlations in spiking dynamics.

E.6 Firing Rates

We compare the firing rates of the three neuron types in a trained Spikformer (embedding dimension 256, $T = 16$) on N-Caltech101. Table 14 lists the average firing rate of every spiking layer for LIF, f-LIF, and *DF*-LIF, together with the overall firing rate of the network.

The overall firing rate decreases from 6.86% for LIF to 5.11% for f-LIF and 4.89% for *DF*-LIF, i.e., *DF*-LIF fires 28.7% less than LIF and 4.2% less than f-LIF, so the accuracy gains reported in the main text are not obtained through denser activity. At the layer level, *DF*-LIF has the lowest firing rate of the three neurons in 12 of the 19 spiking layers, with the largest reductions in `patch_embed.lifs.0` and in the attention-projection and attention neurons of the second block. The exceptions are mainly the query neurons (`q_lif`) of both blocks and `patch_embed.lifs.1`, where *DF*-LIF fires more

Table 14: Layer-wise firing rates (%) of Spikformer (embedding dimension 256) on N-Caltech101 with LIF, f-LIF, and *DF*-LIF neurons. The last row is the overall firing rate of the network.

Layer	LIF	f-LIF	*DF*-LIF
`patch_embed.lifs.0`	9.08	4.96	3.19
`patch_embed.lifs.1`	1.30	0.84	4.99
`patch_embed.lifs.2`	3.39	3.73	2.47
`patch_embed.lifs.3`	1.07	1.45	1.08
`patch_embed.rpe_lif`	9.89	10.14	7.81
`blocks.0.attn.q_lif`	20.89	16.30	29.52
`blocks.0.attn.k_lif`	10.90	10.09	11.54
`blocks.0.attn.v_lif`	6.70	5.71	4.69
`blocks.0.attn.attn_lif`	13.03	7.82	14.44
`blocks.0.attn.proj_lif`	8.98	7.21	4.56
`blocks.0.mlp.fc1_lif`	2.36	2.05	1.18
`blocks.0.mlp.fc2_lif`	8.26	6.85	5.11
`blocks.1.attn.q_lif`	30.06	30.26	35.07
`blocks.1.attn.k_lif`	7.66	9.41	5.88
`blocks.1.attn.v_lif`	7.82	5.96	6.17
`blocks.1.attn.attn_lif`	13.55	16.79	7.37
`blocks.1.attn.proj_lif`	20.76	20.06	13.86
`blocks.1.mlp.fc1_lif`	4.28	4.96	3.95
`blocks.1.mlp.fc2_lif`	27.77	31.19	23.76
Overall	6.86	5.11	4.89

than the other two neurons. In all three models, the highest rates occur in the query neurons (`q_lif`) and the second MLP layer (`fc2_lif`) of the second block, while most other layers stay below 15%.

Because the firing rate determines how many spike-driven operations a network performs on event-driven hardware, the lower rates of *DF*-LIF are favorable for deployment. Measuring the actual energy consumption on neuromorphic hardware is left for future work.

F Algorithm Implementation and Usage Guidance

F.1 Algorithm of the *DF*-LIF Neuron

We now present the practical algorithmic realization of the proposed *DF*-SNN framework. Based on the distributed-order fractional neuron dynamics in (5), we derive an explicit and efficient numerical implementation using a multi-term GL discretization, referred to as `glmulti`. The resulting procedure provides a concrete instantiation of distributed-order fractional spiking neurons and serves as the computational backbone of *DF*-SNNs.

The `glmulti` method approximates the distributed-order fractional operator by a finite weighted combination of GL fractional differences, each corresponding to a distinct fractional order and temporal scale. By aggregating these contributions into a single set of distributed GL coefficients, the neuron dynamics can be expressed as a history-dependent convolution over past membrane potentials. This formulation yields an explicit update rule and avoids implicit solvers, making it well suited for large-scale SNN simulation and training.

Algorithm 1 summarizes the complete update procedure for a *DF*-LIF neuron. At each time step, the neuron first integrates external and synaptic input, then applies the distributed GL convolution over its membrane history, followed by spike generation and reset operations identical to those in conventional SNNs. Importantly, the spike function and reset mechanism are unchanged, ensuring full compatibility with surrogate-gradient training methods.

The fractional orders $\{\alpha_j\}$ are fixed hyperparameters, whereas the distributed coefficients $\{w_j\}$ can be learned jointly with the synaptic weights through standard backpropagation, so that the mixture of temporal scales adapts to each task.

Algorithm 1 *DF*-LIF Neuron (glmulti-*f*-LIF)

Require: Input sequence $\{x(t_k)\}_{k=0}^{N}$; membrane history $\{v_j\}$; spike history $\{s_j\}$; fractional orders $\{\alpha_j\}_{j=1}^{n}$; coefficients $\{w_j\}_{j=1}^{n}$; threshold V_{th}; membrane time constant τ; step size $h > 0$; memory length M; time grid $\{t_k\}_{k=0}^{N}$.

Ensure: Membrane potentials $\{v_k\}_{k=0}^{N}$ and spikes $\{s_k\}_{k=0}^{N}$.

Precompute GL coefficients for each order
for $j = 1$ to n **do**
  $c_0^{(\alpha_j)} \leftarrow 1$
  for $m = 1$ to N **do**
    $c_m^{(\alpha_j)} \leftarrow \left(1 - \frac{1+\alpha_j}{m}\right) c_{m-1}^{(\alpha_j)}$
  end for
end for
Precompute distributed GL weights
for $m = 0$ to N **do**
  $\tilde{c}_m \leftarrow \sum_{j=1}^{n} w_j \, h^{-\alpha_j} \, c_m^{(\alpha_j)}$
end for
Initialization
$v_0 \leftarrow 0, \quad s_0 \leftarrow 0$ (or given initial states)
Store (v_0, s_0) into history
for $k = 0$ to $N - 1$ **do**
  Synaptic drive: $u_k \leftarrow \big(-v_k + I_{\text{in}}\big(x(t_k)\big)\big)/\tau$
  $s \leftarrow \max(0,\, k + 1 - M)$ (short-memory start index)
  GL convolution on membrane history
  $\text{conv} \leftarrow \sum_{j=s}^{k} \tilde{c}_{k+1-j} \, v_j$
  Fractional GL update (subthreshold)
  $v_{k+1} \leftarrow \big(u_k - \text{conv}\big)/\tilde{c}_0$
  Spike generation
  $s_{k+1} \leftarrow H(v_{k+1} - V_{\text{th}})$
  Reset (optional)
  $v_{k+1} \leftarrow \text{Reset}(v_{k+1},\, s_{k+1})$
  Append (v_{k+1}, s_{k+1}) to history
end for

Return: $\{v_k\}_{k=0}^{N}, \{s_k\}_{k=0}^{N}$

F.2 Direct Implementation and Integration

The `LIFNeuronFDE` module implements *DF*-LIF directly within network layers. It combines distributed GL coefficients, historical membrane states, spike generation, and reset, with surrogate gradients for training.

The direct implementation combines the GL coefficients into an aggregated history kernel. The following code snippets illustrate general solver operations rather than verbatim `LIFNeuronFDE` source. In the following, we walk through the key components aligned with the steps in Algorithm 1.

Precomputation of Distributed GL Coefficients. We first compute the GL coefficients $c_m^{(\alpha_j)}$ for each fractional order α_j. In code, this is performed during solver initialization:

```
c_columns = [torch.ones(n_terms, dtype=dtype, device=device)]
for m in range(1, N + 1):
    c_next = (1 - (1 + alpha) / m) * c_columns[m - 1]
    c_columns.append(c_next)
c = torch.stack(c_columns, dim=1)  # Shape: (n_terms, N+1)
```

Here, `alpha` is a tensor of shape `(n_terms,)` containing $\{\alpha_j\}_{j=1}^{n}$, which are fixed hyperparameters. The recurrence relation matches the standard GL coefficient definition.

Subsequently, we compute the distributed-order weights $\tilde{c}_m = \sum_{j=1}^{n} w_j h^{-\alpha_j} c_m^{(\alpha_j)}$:

```
h_neg_power = torch.pow(h, -alpha)
weighted_h = coefficient * h_neg_power
c_tilde = torch.sum(weighted_h.unsqueeze(1) * c, dim=0)
```

The `coefficient` tensor corresponds to the weights $\{w_j\}$. When `learn_coefficient=True`, `coefficient` is registered as an `nn.Parameter`, enabling gradient-based optimization, while `alpha` always stays fixed.

State Update with Short-Term Memory Truncation. The PyTorch implementation supports an optional memory window of length M, as illustrated below:

```
start_idx, _ = get_memory_bounds(k, memory)
convolution_sum = 0
for j in range(start_idx, k):
    coeff_idx = k - j
    convolution_sum += weights[coeff_idx] * history_i[j]
```

This optional short-memory approximation restricts the convolution to a recent window. All experiments in this paper use the full available history; the CuPy full-history implementation does not apply this truncation.

With $u_k = (-v_k + I_{\text{in}}(x(t_k)))/\tau$, the subthreshold membrane potential is then updated as:

$$v_{k+1} = \frac{u_k - \text{conv}}{\tilde{c}_0}, \tag{S.31}$$

where the corresponding drift is represented by `f_k_i` in the following update:

```
v_next = (f_k_i - convolution_sum) * c_tilde_0_inv
```

Finally, spike generation and reset are handled by the neuron module in `ODEFuncFromFX` using a Heaviside function $H(\cdot)$ in the forward pass and a surrogate gradient during backpropagation—consistent with standard SNN training practices.

Integration into Deep SNNs. The `LIFNeuronFDE` module directly implements *DF*-LIF dynamics and can be integrated into network layers. An alternative `SNNWrapper` interface infers spiking layers from a base network and assigns per-layer $\{\alpha_j\}$ and $\{w_j\}$ configurations. The wrapper uses the `snn_solve` interface for single-term and multi-term fractional dynamics.

This design allows *DF*-SNNs to be deployed in CNNs, Transformers, or any other architecture with minimal code changes, while maintaining full compatibility with PyTorch's autograd system.

F.3 Direct Use of the Distributed-order Neuron

The distributed-order GL neuron is implemented directly as the stand-alone layer `LIFNeuronFDE` with `method="glmulti"`. Following the convention of existing SNN frameworks [18, 21], this layer runs in *multi-step* mode: instead of being called T times with an input of shape $[N, *]$, it is called once with the whole sequence of shape $[T, N, *]$ and returns the spike sequence of the same shape. This section walks through its basic usage.

Step Mode and Data Format. In multi-step mode, the time dimension is always the first dimension of the input. A minimal example with vector inputs is

```
import torch
from spikeDE_dis.neuron import LIFNeuronFDE

T, N, D = 16, 32, 128
neuron = LIFNeuronFDE(
    tau=2.0, threshold=1.0,
    method="glmulti", step_mode="m", backend="cupy",
    T=T, batchsize=N, step_size=0.1,
    multi_beta=[0.3, 0.4, 0.5],
```

```
    multi_coefficient=[0.5, 0.5, 0.7],
).cuda()

x_seq = torch.rand([T, N, D]).cuda()
y_seq = neuron(x_seq)
print(y_seq.shape)   # torch.Size([16, 32, 128])
```

The same layer accepts convolutional features of shape $[T, N, C, H, W]$. Data loaders for event datasets usually return tensors of shape $[N, T, C, H, W]$, which must be permuted before entering the network:

```
x_seq = frames.permute(1, 0, 2, 3, 4).contiguous()   # [T, N, C, H, W]
```

Note that multi-step mode only changes how the sequence is passed to the layer: the neuron still updates its state in temporal order, and the GL history at step k contains all previous steps of the current sequence. The sequence length must equal the `T` given at construction.

Distributed-order Coefficients. The fractional orders $\{\alpha_j\}$ and their weights $\{w_j\}$ are given by `multi_beta` and `multi_coefficient`, two lists of the same length in which the j-th weight belongs to the j-th order. Whether and how the weights are learned is set by the arguments below:

Argument	Effect
`learn_coefficient`	`False`: fixed weights; `True`: trainable weights.
`coefficient_constraint`	`"softplus"`: $w_j > 0$; `"softmax"`: $w_j > 0$ and $\sum_j w_j = 1$.
`glmulti_history_grad`	`"full"`: back-propagate through the full history.

For example, a neuron with learnable Softplus weights is created by

```
neuron = LIFNeuronFDE(
    tau=2.0, threshold=1.0,
    method="glmulti", step_mode="m", backend="cupy",
    T=T, batchsize=N, step_size=0.1,
    multi_beta=[0.3, 0.4, 0.5],
    multi_coefficient=[0.5, 0.5, 0.7],
    learn_coefficient=True,
    coefficient_constraint="softplus",
    glmulti_history_grad="full",
).cuda()
```

The weights that are actually used in the forward pass, i.e., after the constraint, can be inspected at any time:

```
w = neuron.get_multi_coefficient()
print(w.detach().cpu().tolist())   # one weight per order
```

Using the Neuron in a Network. `LIFNeuronFDE` is an ordinary `nn.Module` and can replace a LIF layer in any multi-step SNN. When `learn_coefficient=True`, the weights are registered as parameters and are optimized together with the network weights:

```
import torch.nn.functional as F
from spikingjelly.activation_based import functional

optimizer = torch.optim.Adam(net.parameters())
for frames, label in train_loader:              # frames: [N, T, C, H, W]
    x_seq = frames.permute(1, 0, 2, 3, 4).contiguous().cuda()
    out = net(x_seq).mean(0)                    # firing rate over T
    loss = F.cross_entropy(out, label.cuda())
    optimizer.zero_grad()
```

```
    loss.backward()
    optimizer.step()
    functional.reset_net(net)                    # reset neuron states
```

As in previous methods [18, 21], the neuron keeps its membrane state and GL history inside the module, so the states of all neurons are reset by `functional.reset_net(net)` after every batch. Each neuron instance owns its own weights by default; to share one set of weights across the whole network, pass the same `shared_multi_coefficient` parameter to all instances.

F.4 Alternative SNNWrapper Usage Example

To facilitate the practical application of our proposed *DF*-SNN, we provide a comprehensive example that illustrates how to configure and deploy different fractional-order dynamics using the `SNNWrapper` class within a simple two-layer SNN architecture. This section demonstrates various configurations, including per-layer single-term or multi-term fractional orders and learnable distributed weights.

Basic Setup and Configuration. The `SNNWrapper` interface provides an alternative way to configure fractional-order dynamics on a per-layer basis. Below, we define a minimal 2-layer SNN (`SimpleSNN`) for demonstration purposes:

```
class SimpleSNN(nn.Module):
    def __init__(self, input_dim=10, hidden_dim=20, output_dim=10):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.bn1 = nn.BatchNorm1d(hidden_dim)
        self.lif1 = LIFNeuron(tau=2.0, threshold=1.0)

        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.bn2 = nn.BatchNorm1d(output_dim)
        self.lif2 = LIFNeuron(tau=2.0, threshold=1.0)

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.lif1(x)

        x = self.fc2(x)
        x = self.bn2(x)
        x = self.lif2(x)

        return x
```

Configuring Fractional Dynamics. The behavior of the fractional derivative is controlled via the `alpha` argument (the fixed fractional orders) and related flags in `SNNWrapper`. The framework supports three canonical configurations through intuitive parameter patterns—no explicit mode selection is required in most cases, as the system automatically detects the intended configuration from the structure of `alpha` and `multi_coefficient`.

Table 15: Configurations of four fractional dynamics.

Configuration	Format of `alpha`	`multi_coefficient`	Notes
Scalar α	`float`	Not needed.	Same α for all layers.
Per-layer α	`list`	Not needed.	One α per layer.
Multi-term Broadcast	`list`	Optional flat list.	Same multi-term FDE applied to all layers.

Additional control is provided via:

- `learn_coefficient`: whether multi-term weights are trainable;

- `alpha_mode`: can be `'auto'` (default), `'per_layer'`, or `'multiterm'`, usually omitted unless disambiguation is needed.

To configure a 2-layer network where Layer 0 uses a single-term $\alpha = 0.4$ and Layer 1 uses a 3-term distributed-order model with orders `[0.3, 0.6, 0.9]` and learnable coefficients initialized to `[1.0, 0.3, 0.1]`, simply pass:

```
net = SNNWrapper(
    SimpleSNN(),
    integrator='fdeint',
    alpha=[[0.4], [0.3, 0.6, 0.9]],
    multi_coefficient=[[1.0], [1.0, 0.3, 0.1]],
    learn_coefficient=[False, True]
)
```

The nested structure automatically triggers per-layer multi-term mode.

Forward Pass and Output Specification. Upon initialization, the wrapped network processes input time series $\mathbf{X} \in \mathbb{R}^{T\times B\times N}$, where T denotes the number of time steps, B the batch size, and N the dimensionality of input features per time step. A monotonically increasing time vector $\boldsymbol{t} \in \mathbb{R}^T$ must also be provided to define the evaluation points for the fractional differential equation solver; this is typically generated via `torch.linspace(0, 1, T)`.

The network outputs a binary spike tensor $\mathbf{O} \in \{0,1\}^{T\times B\times C}$, where C is the number of output neurons. Each element $o_{t,b,c}$ indicates whether the c-th output neuron emitted a spike at time step t for the b-th sample. This spike train serves as a direct representation for downstream tasks. For classification, common decoding strategies include temporal voting or rate coding. Specifically, the average firing rate per neuron and sample over the simulation window is computed as:

```
firing_rate = output.mean(dim=0)  # Shape: [B, C]
```

This rate vector provides a differentiable, low-dimensional feature representation suitable for integration with standard classification or regression heads.

End-to-End Training with Learnable Dynamics. When `learn_coefficient` is enabled, the distributed weights `multi_coefficient` become trainable, while the fractional orders `alpha` remain fixed. Gradients propagate seamlessly through the fractional integrator module, enabling joint optimization of the distributed weights and the underlying neural network weights via standard gradient-based methods. This facilitates fully differentiable, end-to-end training of the hybrid fractional-spiking architecture.

By following these examples, users can easily configure and experiment with various fractional-order dynamics in their SNN models. The flexibility provided by `SNNWrapper` allows for seamless integration into existing PyTorch workflows, supporting advanced research and applications in spiking neural networks.

G Broader Impact

This paper presents work whose goal is to advance the field of spiking neural networks and temporal modeling. While the proposed method may contribute to future applications of neuromorphic computing, we do not foresee any immediate ethical concerns or negative societal impacts that require specific discussion at this stage.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The main claims are stated in the Introduction and supported by the method development.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The section "Limitation and Future Work" discusses the shared fixed-order configuration, the cost of full-history computation, and directions for future extensions.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: The theorems proposed in this paper are supported by detailed proofs.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We describe the main experimental results in Section experiments. More detailed dataset descriptions and implementation settings are provided in Appendix.

   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 neuron implementations and network definitions are provided in the supplementary material; complete preprocessing, configurations, training/evaluation scripts, and seeds will be released at `https://github.com/cjge-ustc/DF-SNN-official`.

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 report the settings in the experiments settings.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: Error bars are reported as mean $\pm$ sample standard deviation for the neuromorphic classification results (three seeds), the key comparisons (matched single-order baseline, non-negative weights, model scales, richer neurons, HarDVS efficiency; five seeds), and the graph experiments (20 runs). The static image, object detection, and a few supplementary results are single runs and are labeled as such in their captions.

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 implementation framework and hardware environment in Appendix, and the forward-efficiency measurement protocol with peak GPU memory usage in 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 conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We discuss broader impacts in Section G.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not pose such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We cite the original paper that produced the code package or dataset.

    Guidelines:

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

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

13. **New assets**

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

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

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

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

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

    Answer: [N/A]

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

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve human-subject research and therefore does not require IRB approval.

    Guidelines:

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

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were not used as an important, original, or non-standard component of the core methodology.

    Guidelines:

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