UNISPIKE: BOOSTING THE PERFORMANCE OF SPIKING NEURAL NETWORKS WITH HYBRID TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking neural networks (SNNs) offer an efficient inference paradigm with two desirable properties: *addition-only computation* and *step-by-step inference*. Addition-only computation removes expensive multipliers and reduces arithmetic energy, while step-by-step inference enables temporally progressive outputs for faster response and early exiting. However, achieving high accuracy while preserving these properties remains challenging. Several accuracy-oriented SNN operators, such as SEW-addition, temporal attention, and temporal addition, improve representation learning by introducing multi-valued activations or temporal synchronization, thereby violating addition-only computation or step-by-step inference.

To improve accuracy under these property-preserving constraints, we propose UNISPIKE, a hybrid-training framework that initializes an SNN from a quantized ANN and further fine-tunes it with back-propagation-through-time (BPTT). The key design is **Unified Clip**, a unified ANN-side activation that replaces operators incompatible with spike-based inference, including softmax, layer normalization, and GeLU. Its quantized form is equivalent to the ST-BIF spiking neuron, providing effective ANN-to-SNN initialization for SNN fine-tuning. Built upon Unified Clip, we design **UniFormer**, a spike-compatible vision transformer tailored for accurate, addition-only, and step-by-step SNN inference. On ImageNet-1K, UNISPIKE achieves 80.8% accuracy at 4 time-steps, outperforming the SOTA addition-only and step-by-step direct-training method Spike-Driven Transformer V2.

1 INTRODUCTION

Spiking neural networks (SNNs) leverage spikes to encode, transmit, and learn information from the world. Compared to artificial neural networks (ANNs), SNNs mimic the human brain and exhibit a more efficient computational paradigm, offering notable advantages such as addition-only computation and a step-by-step inference pattern (Roy et al., 2019).

Addition-only computation is a key premise for energy-efficient SNN inference. By removing multipliers, SNNs simplify hardware compute units and reduce arithmetic energy cost Akopyan et al. (2015); Narayanan et al. (2020). As shown in Figure 1(a), under 32-bit precision, an integer addition consumes 32$\times$ less energy than a multiplication (0.1 pJ vs. 3.2 pJ) Horowitz (2014), leading to lower inference energy and carbon footprint. Meanwhile, step-by-step inference enables temporally progressive decision making. As illustrated in Figure 1(b,c), in contrast to layer-by-layer inference, where the output is available only after all time-steps of all layers are completed, step-by-step execution propagates each time-step through the entire network and allows the model to exit once the prediction becomes sufficiently confident Fang et al. (2023). This property is important for latency-critical applications such as autonomous driving Qian et al. (2022).

However, achieving high accuracy while preserving these properties is non-trivial, since several accuracy-oriented SNN operators compromise either addition-only computation or step-by-step inference. As shown in Table 1, SEW-addition Fang et al. (2021) breaks the addition-only property by converting binary/ternary spike activations into multi-valued feature maps through residual addition. Temporal attention Lee et al. (2025) and temporal addition Yao et al. (2025) improve accuracy by aggregating information across time-steps, but introduce temporal synchronization and thus prevent step-by-step inference. Replacing these operators with addition-only and step-by-step friendly alternatives leads to an average 3.58% accuracy drop on ImageNet-1K.

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

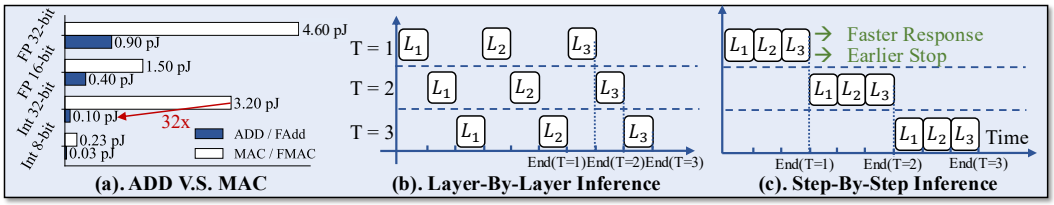


Figure 1: **Advantages of Step-by-step Inference and Addition-Only.** L_1 is the 1st SNN layer. (a) Addition consumes less energy than MAC (Horowitz, 2014), showing the energy-saving potential of addition-only SNN. (b) Layer-by-layer inference produces outputs at the first time-step only after all preceding layers have completed their processing across every time-step. (c) Step-by-step inference processes each time-step separately, providing faster response and earlier stopping.

Table 1: **ImageNet-1K accuracy gaps when enforcing addition-only or step-by-step inference.**

Model	Violate Feature	Original → Replaced Operator	Param(M)	T	Acc. Gap
QKFormer	Addition-only	SEW-shortcut → MS-shortcut	16.47	4	78.80% → 76.48%
STAtten	Step-by-step	Temporal Spike Attention → Spike Attention	66.34	4	78.11% → 76.18%
E-SpikeFormer	Step-by-step	Temporal Addition → Vanilla Addition	10.00	4	77.80% → 71.30%

To address the low-accuracy problem, we propose UNISPIKE, *an SNN framework that achieves high accuracy while preserving addition-only computation and step-by-step inference.* UNISPIKE adopts hybrid training, which first pre-trains an ANN, converts it into an SNN, and then fine-tunes the converted SNN with back-propagation-through-time (BPTT). To preserve desired inference properties, we introduce **Unified Clip**, an ANN-side activation that replaces these unfriendly operators on the ANN side, including softmax, layernorm, and GeLU. Meanwhile, the quantized form of Unified Clip is equivalent to ST-BIF neuron You et al. (2024), providing a better initialization for SNN fine-tuning in hybrid training. Built upon Unified Clip, we further design **UniFormer**, a spike-compatible vision transformer tailored for accurate, addition-only, and step-by-step SNN inference. To measure the energy efficiency of addition-only, we adopt an energy model considering memory access in experiments (Section 5.2), showing a 31.1% energy reduction compared to quantized ANN before conversion. The contributions of UNISPIKE are summarized as follows:

- ▷ We introduce **Unified Clip**, a unified ANN-side activation that replaces operator-level unfriendly components. Its quantized form is equivalent to the ST-BIF neuron You et al. (2024), enabling effective ANN-to-SNN initialization without compromising ANN accuracy.
- ▷ Built upon Unified Clip, we propose **UniFormer**, a transformer architecture tailored for hybrid training. UniFormer is softmax-free and layernorm-free, and bridges ANN pre-training and SNN fine-tuning while supporting addition-only computation and step-by-step inference.
- ▷ UNISPIKE achieves 80.8% ImageNet-1K accuracy at 4 time-steps, outperforming the SOTA addition-only, step-by-step direct learning method Spike-Driven Transformer V2 (80.00%). To our knowledge, it is the first transformer-based hybrid training SNN validated on large-scale datasets.

2 BACKGROUND AND RELATED WORK

2.1 SPIKING NEURON

A spiking neuron is the basic component of an SNN, integrating inputs modulated by synaptic weights and outputting spike trains. To overcome the low accuracy issue in the A2S algorithm, a spiking neuron called *bipolar integrate&fire with spike tracer* (*aka*. ST-BIF) is proposed (Hu et al., 2023). For the i-th pre-synaptic ST-BIF neuron, its output at time-step[1] t is $x^{\text{in}}_{i,t} \in \{-1, 0, 1\}$, and its connection strength (*i.e.*, synaptic weight) with post-synaptic neuron is w_i. Each ST-BIF neuron contains a memory unit called *membrane* V_t, which can be modeled as a capacitor with a capacitance of C. The calculation procedure of the ST-BIF neuron is modeled below:

$$\hat{V}_t = V_{t-1} + (\sum\nolimits_i w_i \cdot x_{i,t})/C; \quad V_t = \hat{V}_t - y_t/C; \quad S_t = S_{t-1} + \Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) \tag{1}$$

[1]time-step: also called step, a suitable time interval whose length is adequate for once update in the system, including all spikes in the synapse to transport, all the neurons to integrate, and fire spikes once.

Table 2: **Summary of Learning Algorithm.** A2S: ANN-to-SNN conversion. DT: directly training. HT: hybrid training. QANN Eq.: if equivalence to QANN. #SOP: the number of synaptic operations.

Work	Type	Structure	Neuron	Surrogate	Time-step	#SOP	QANN Eq.	Unfriendly Op.
SpikformerV2 (Zhou et al., 2024a)	DT	Transformer	LIF	Sigmoid	4	Low	✗	SEW-Shortcut
QKFormer (Zhou et al., 2024b)	DT	Transformer	LIF	Sigmoid	4	Low	✗	SEW-Shortcut
STAtten (Lee et al., 2025)	DT	Transformer	LIF	Sigmoid	4	Low	✗	Temporal Attention
E-SpikeFormer (Yao et al., 2025)	DT	Transformer	IF	n/a	8	Low	✗	Temporal Addition
QCFS (Bu et al., 2023)	A2S	CNN	IF	n/a	256	High	✗	LayerNorm
MST (Wang et al., 2023)	A2S	Transformer	IF	n/a	512	High	✗	LayerNorm
SpikeZIP-TF (You et al., 2024)	A2S	Transformer	ST-BIF	n/a	64	High	✓	LayerNorm
HYB-SNN (Abuhajar et al., 2025)	HT	CNN	IF	BP	5	Low	✗	None
JAS-SNN (Baltes et al., 2023)	HT	CNN	LIF	Sigmoid	250	High	✗	None
UniSpike (Ours)	HT	Transformer	ST-BIF	Sigmoid	4	Low	✓	None

$$\Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) = \begin{cases} 1; & \hat{V}_t \geq V_{\text{thr}} \;\&\; S_{t-1} < S_{\max} \\ 0; & \text{other} \\ -1; & \hat{V}_t < 0 \;\&\; S_{t-1} > S_{\min} \end{cases}, \; Y_t = \Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) \times q_{\text{thr}} \tag{2}$$

where $\hat{V}_t$ is the membrane potential after integration, Y_t is the output spikes at time-step t and V_{thr} is the firing threshold. S_t is an additional memory unit in ST-BIF neuron (*aka*., spike tracer), and $S_{\max}, S_{\min}$ are the maximum and minimum values of the spike tracer. Equations (1) and (2) describe the dynamics of spike integration, neuron firing, and membrane updating, respectively. To facilitate gradient derivation of the decision function, we rewrite Equation (2) as follows:

$$\Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) = \theta(\hat{V}_t - V_{\text{thr}}) \cdot \theta(S_{\max} - S_{t-1} - \varepsilon) - \theta(-\hat{V}_t - \varepsilon) \cdot \theta(S_{t-1} - S_{\min} - \varepsilon) \tag{3}$$

where θ is the step function. We use ε to express strict $>$ and $<$ conditions in Equation (2).

2.2 SNN Learning Algorithm

Direct Learning (*aka*. DT) trains the SNN with the BPTT algorithm at small time-steps (*e.g*. 1 or 4 time-steps (Yao et al., 2024a; Zhou et al., 2024a) in Table 2). In the DT algorithm, leaky IF neuron (*aka.* LIF) is widely adopted (Yao et al., 2024b;a). However, due to the inaccurate gradient estimated by the surrogate function, the accuracy of the SNN is lower than that of the ANN with the same number of parameters (Yao et al., 2024a; Zhou et al., 2024a).

ANN-to-SNN (*aka*. A2S) Conversion transfers the parameters of the pre-trained ANN into its SNN counterpart that yields close-to-ANN accuracy (Malcolm and Casco-Rodriguez, 2023). As shown in Table 2, previous A2S algorithms such as MST (Wang et al., 2023) and QCFS (Bu et al., 2023) use an integrate-and-fire (IF) neuron to replace the quantized ReLU function. In recent studies, to reduce the conversion loss caused by the non-equivalence between IF neuron and the quantized ReLU function, ST-BIF neuron is proposed (Hu et al., 2023). However, compared to the DT algorithm, the high accuracy of converted SNN requires large inference time-steps ($\geq$ 64 time-steps in Table 2), incurring large synaptic operations (*aka*. SOP)[2].

Hybrid Training (*aka*. HT) integrates A2S and DT by using the converted SNN as a pre-trained model and fine-tuning it with DT to achieve high accuracy at small time-steps. As shown in Table 2, only a few studies (Rathi et al., 2020; Rathi and Roy, 2020) adopt the HT algorithm, since the non-equivalence between activation (ReLU) and spiking neuron (IF) causes incorrect weight initialization, resulting in low accuracy.

3 Challenges of Hybrid Training

HT combines the advantages of A2S and DT, making it a promising paradigm for high-accuracy SNNs with ultra-low inference time-steps. However, applying HT to advanced architectures such as Transformers remains challenging. During conversion, ANN-specific SNN architectures struggle to preserve addition-only computation and step-by-step inference without accuracy degradation. In the fine-tuning stage, neuron-activation mismatch further leads to sub-optimal SNN initialization.

[2]One synaptic operation refers to the operation that the spiking neuron integrates one pre-synaptic spike into the membrane or fires one spike to the post-synaptic neurons.

Challenge-1: ANN-Specific SNN architecture violates addition-only and step-by-step properties. In hybrid training, the ANN and SNN share the same architecture (Abuhajar et al., 2025). However, ANN-specific operators, such as softmax, layernorm, and GeLU, involve nonlinear arithmetic, making them incompatible with the addition-only property. Moreover, softmax and layernorm require information aggregation for normalization, which introduces synchronization across time-steps and thus conflicts with step-by-step inference. Therefore, removing these incompatible operators while preserving ANN training accuracy remains a major challenge.

Challenge-2: Sub-optimal SNN initialization caused by "neuron-activation" mismatch. Previous HT algorithms (Rathi and Roy, 2020; Abuhajar et al., 2025; Baltes et al., 2023) replace quantized ReLU with LIF or IF neurons during conversion. These neurons are not mathematically equivalent to quantized ReLU under limited time-steps (Bu et al., 2023). This mismatch turns the ANN-optimal weights into a sub-optimal SNN initialization, increasing the burden on fine-tuning and degrading final performance. Figure 6 further confirms that such neuron mismatch hinders BPTT fine-tuning.

4 METHODOLOGY

To address the challenges in hybrid training, UNISPIKE introduces **Unified Clip**, an activation function that replaces ANN-side operators incompatible with addition-only and step-by-step features (Challenge 1). Moreover, the quantized form of **Unified Clip** is equivalent to ST-BIF neuron, providing effective initialization for SNN fine-tuning (Challenge 2). Built on **Unified Clip**, UNISPIKE propose **UniFormer**, a transformer-based architecture that supports these SNN-specific features.

4.1 UNIFIED-CLIP

Definition. Unified-Clip in UNISPIKE replaces SNN-unfriendly operators (softmax, layernorm, GeLU), making them support step-by-step inference (challenge-1). Therefore, Unified-Clip must satisfy two key constraints: *1)* Neuron updates in Unified-Clip must be independent across time-steps. Therefore, operators involving data collection from future time-steps are not allowed. *2)* Its quantized form must be mathematically equivalent to ST-BIF neurons. To meet these constraints, we design Unified-Clip ($\mathrm{UC}(x)$) as follows:

$$\mathrm{UC}(x) = \mathrm{clip}(\alpha \cdot x + \beta, X_{\min}, X_{\max}) \cdot \gamma \tag{4}$$

where x is an element of the input tensor $\mathbf{X}$, the clip function truncates the input x within $[X_{\min}, X_{\max}]$ and α, β, γ are the parameters of Unified-Clip. Unified-Clip is an element-wise operation, where the neurons update independently.

Equivalent to ST-BIF neuron. We show that quantized Unified-Clip equals the accumulated output of the ST-BIF neuron by leveraging its relation to the quantized function proven in (You et al., 2024):

$$\sum_t^T \mathrm{ST_BIF}(x_t) = \mathrm{Quan}(\sum_t^T x_t) = \mathrm{clip}(R(\tfrac{\sum_t^T x_t}{s}), C_{\min}, C_{\max}) \cdot s \tag{5}$$

ST_BIF is the ST-BIF neuron function and omits the updates of V_t, S_t for similarity. where x_t is the input spikes, T is the total time-steps, R is the rounding function that maps the input to the nearest integer, s is the quantization scale, and $C_{\min}, C_{\max} \in \mathbb{Z}$ are the integer lower and upper bounds. Then, we quantize Unified-Clip, combining Equation (4) and Equation (5) and finally get:

$$\mathrm{Quan}(\mathrm{UC}(\sum_t^T x_t)) = \mathrm{clip}(\mathrm{clip}(R((\alpha \sum_t^T x_t + \beta)\tfrac{\gamma}{s}), R(X_{\min}\tfrac{\gamma}{s}), R(X_{\max}\tfrac{\gamma}{s})), C_{\min}, C_{\max}) \cdot s \tag{6}$$

If we limit $R(X_{\min} \cdot \frac{\gamma}{s}) \leq C_{\min}$ and $R(X_{\max} \cdot \frac{\gamma}{s}) \geq C_{\max}$, the Equation (6) can be simplified:

$$\mathrm{Quan}(\mathrm{UC}(\sum_t^T x_t)) = \mathrm{clip}(R(\tfrac{\sum_t^T (\alpha x_t \cdot \gamma + \beta \cdot \gamma / T)}{s}), C_{\min}, C_{\max}) \cdot s \tag{7}$$

By comparing Equation (5) and Equation (7), we build the equivalence between the output quantized Unified-Clip and the accumulated output of ST-BIF neuron:

$$\sum_t^T \mathrm{ST_BIF}(\alpha x_t \cdot \gamma + \beta \cdot \gamma / T) = \mathrm{Quan}(\mathrm{UC}(\sum_t^T x_t)); \quad \text{s.t.} \quad s \leq \min(\tfrac{|X_{\max}| \cdot \gamma}{|C_{\max}|}, \tfrac{|X_{\min}| \cdot \gamma}{|C_{\min}|}) \tag{8}$$

Equation (8) shows that by limiting the range of quantization scale s, quantized Unified Clip is equivalent to ST-BIF neuron. Therefore, by replacing Unified Clip with ST-BIF neuron, the quantized UniFormer is equivalent to its spike version. The equivalence provides better weight initialization for the SNN converted by the A2S compared to replacing quantized ReLU with IF neuron (Challenge-2).

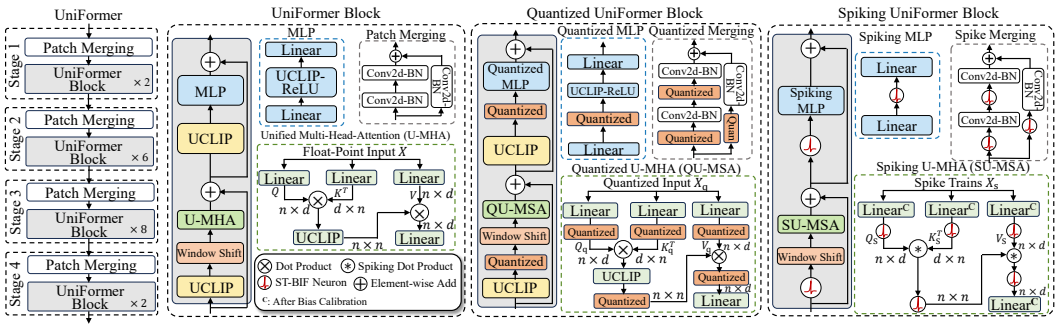


Figure 2: **UniFormer Architecture.** UniFormer consists of 4 stages. Each stage contains a patch merging module and several UniFormer blocks. UniFormer is free of softmax and layer normalization operators and ensures that the inputs of all matrix multiplications are spikes.

4.2 UniFormer

Key Designs for Step-by-step & Addition-only. As shown in Figure 2, we propose UniFormer, a vision transformer with Unified-Clip, featuring two key designs: **1) MS-Shortcut:** Unlike SEW-shortcut (Fang et al., 2021), which inserts neuron layers before residual connections and causes the outputs after residual addition to remain multi-valued rather than spikes, MS-shortcut (Han et al., 2022) places neuron layers before attention and FFN blocks to convert residual outputs into spikes, thereby enabling addition-only matrix multiplications. **2) Softmax/LayerNorm Replacement:** UniFormer introduces Unified-Clip to replace Softmax and LayerNorm, as these operations require aggregating information across all time-steps, preventing step-by-step inference.

UniFormer Block. A UniFormer block consists of a unified multi-head attention layer, an MLP layer, a window shift module (Liu et al., 2021), and two Unified-Clip modules as a replacement of layer normalization. Specifically, we use Unified-Clip with different $\alpha, \beta, \gamma, C_{\min}, C_{\max}$ to substitute the softmax, layer normalization and GeLU modules in traditional transformers.

Softmax Replacement. Softmax normalizes input tensors between 0 and 1, avoiding extremely high activation while maintaining the activation distribution (Wortsman et al., 2023). However, in SNN, spikes from all time steps are necessary for softmax calculation, which is inapplicable to step-by-step processing. Therefore, in UniFormer, we use Unified-Clip to substitute softmax:

$$\mathrm{UC}_{\mathrm{softmax}}(x) = \mathrm{clip}(1/N \cdot x, 0, 1) \tag{9}$$

where N is the number of tokens. We set $C_{\min} = 0, C_{\max} = 1$ to clamp the input to [0, 1] as softmax does. Then, we set $\alpha = 1/N, \beta = 0, \gamma = 1$ to preserve the input distribution as closely as possible.

Layer Normalization Replacement. Layer normalization normalizes across the feature dimension of each individual sample. It reduces fluctuations in the input distribution to each layer during training, making training more stable (Ba et al., 2016). However, similar to softmax, layer normalization also uses spikes from all time steps for normalization in SNN, violating the step-by-step inference. (Zhu et al., 2025) finds that the input-output mapping of layer normalization can be approximated with dynamic tanh. However, dynamic tanh is not equivalent to ST-BIF neuron. Therefore, UniFormer uses Unified-Clip to approximate layer normalization by using hard-tanh:

$$\mathrm{UC}_{\mathrm{LayerNorm}}(x) = \mathrm{clip}(\alpha \cdot x + \beta, -1, 1) \cdot \gamma \tag{10}$$

where $C_{\min} = -1$, $C_{\max} = 1$, and α, β, γ are learnable parameters. Unified-Clip is similar to dynamic hard-tanh (Zhu et al., 2025), but differs in the position of the affine transformation. Unified-Clip applies the affine transform before clipping, rather than after, making the equivalence constraint in Equation (8) easier to satisfy.

GeLU Replacement. GeLU activation function (Lee, 2023) performs well in vision transformer. However, GeLU function is not equivalent to ST-BIF neuron. Therefore, we replace GeLU with the ReLU function and use Unified-Clip to express ReLU:

$$\mathrm{UC}_{\mathrm{ReLU}}(x) = \mathrm{clip}(x, 0, \infty) \tag{11}$$

where we set $\alpha = 1, \beta = 0, \gamma = 1, C_{\min} = 0, C_{\max} = \infty$. By replacing GeLU with Unified-Clip, we ensure the equivalence between ANN and SNN while maintaining the ANN performance.

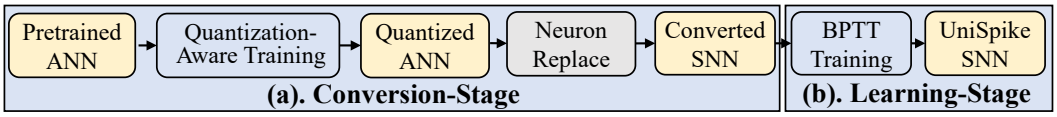


Figure 3: **UNISPIKE Pipeline.** UNISPIKE trains an SNN in two stages: conversion and learning.

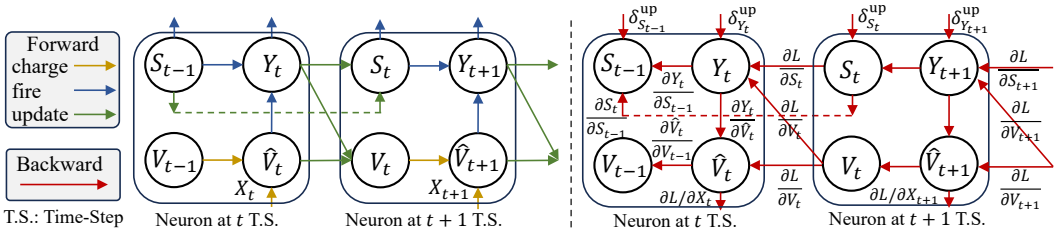


Figure 4: **Computation Graph of ST-BIF Neuron.**

4.3 UNISPIKE PIPELINE

The pipeline of UNISPIKE is shown in Figure 3, which contains four stages: (1) Build a vanilla ANN and train the ANN from scratch. (2) Insert the quantization function into the trained ANN and train the quantized ANN (QANN) by quantization-aware training. (3) Replace the quantization function with a learnable ST-BIF neuron to convert the QANN to an SNN. (4) Train the converted SNN by the BPTT algorithm and finally obtain a high-performance SNN.

4.4 BACKWARD PROPAGATION THROUGH TIME (BPTT) FOR ST-BIF NEURON

Backward Gradient of ST-BIF neuron. Figure 4 shows the computation graph of ST-BIF neuron. After forwarding, we calculate the gradients in the opposite direction of the forward computation graph. As displayed in the backward pass in Figure 4 (right), ST-BIF neuron at t time-step receives partial derivatives $\delta^{\text{up}}_{S_{t-1}}, \delta^{\text{up}}_{Y_t}$ from neurons in the next layer. Then, it receives partial derivatives $\partial\mathcal{L}/\partial V_t, \partial\mathcal{L}/\partial S_t$ from the same neuron at next time-step $t+1$. After receiving these partial derivatives, the neuron calculates the partial derivatives to the same neuron at the last time-step and neurons in the last layers, including $\partial\mathcal{L}/\partial V_{t-1}, \partial\mathcal{L}/\partial S_{t-1}$ and $\partial\mathcal{L}/\partial X_t$:

$$\begin{aligned}
\partial\mathcal{L}/\partial V_{t-1} &= (\partial\mathcal{L}/\partial Y_t + \partial\mathcal{L}/\partial S_t - V_{\text{thr}} \times \partial\mathcal{L}/\partial V_t)(\partial\Theta(\hat{V}_t, V_{\text{thr}}, S_t)/\partial\hat{V}_t) + \partial\mathcal{L}/\partial V_t \\
\partial\mathcal{L}/\partial S_{t-1} &= (\partial\mathcal{L}/\partial Y_t - V_{\text{thr}} \times \partial\mathcal{L}/\partial V_t)(\partial\Theta(\hat{V}_t, V_{\text{thr}}, S_t)/\partial S_{t-1}) + \partial\mathcal{L}/\partial S_t \\
\partial\mathcal{L}/\partial X_t &= (\partial\mathcal{L}/\partial Y_t + \partial\mathcal{L}/\partial S_t - V_{\text{thr}} \times \partial\mathcal{L}/\partial V_t)(\partial\Theta(\hat{V}_t, V_{\text{thr}}, S_t)/\partial\hat{V}_t) + \partial\mathcal{L}/\partial V_t
\end{aligned} \tag{12}$$

The detailed derivation is in the appendix (Section A1). $\partial\Theta(\hat{V}_t, V_{\text{thr}}, S_t)/\partial\hat{V}_t, \partial\Theta(\hat{V}_t, V_{\text{thr}}, S_t)/\partial S_{t-1}$ are partial derivatives of fire decision function, which is non-differentiable. Therefore, we use a surrogate gradient to estimate the gradients.

Surrogate Gradient of ST-BIF neuron. In ST-BIF neuron, the fire decision function Θ is expressed by several step functions in Equation (3). Therefore, we expand the partial derivatives of Θ as follows:

$$\begin{aligned}
\partial\Theta(\cdot)/\partial\hat{V}_t &= \theta'(\hat{V}_t - V_{\text{thr}}) \cdot \theta(S_{\max} - S_{t-1} - \varepsilon) + \theta'(-\hat{V}_t - \varepsilon) \cdot \theta(S_{t-1} - S_{\min} - \varepsilon) \\
\partial\Theta(\cdot)/\partial S_{t-1} &= -\theta(\hat{V}_t - V_{\text{thr}}) \cdot \theta'(S_{\max} - S_{t-1} - \varepsilon) - \theta(-\hat{V}_t - \varepsilon) \cdot \theta'(S_{t-1} - S_{\min} - \varepsilon)
\end{aligned} \tag{13}$$

where θ' is Dirac Delta function. Since Dirac Delta function is hard to implement on the GPU platform, we use the sigmoid function to estimate θ:

$$\sigma(x) = 1/(1 + e^{-\mu x}) \tag{14}$$

where μ is a constant, controlling the steepness of the sigmoid function.

5 EXPERIMENT

Experiment Setup. We evaluate UNISPIKE on both static and neuromorphic vision datasets. The static datasets include CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009),

Table 3: **Comparison between SOTA algorithms on ImageNet.** $^{\triangle}$ denotes non-spike operations estimated as MAC operations, while † denotes QANN without weight quantization. The best and second-best results are highlighted in **bold** and gray, respectively, among methods supporting addition-only and step-by-step inference at comparable energy. ‡ reports the compute energy of MAC/AC operations only, excluding memory access; full energy analysis is provided in Table 5.

Methods	Cate.	Architecture	Addition Only	Step-by-Step	Param (M)	Energy‡ (mJ)	Step	Acc.(%)
ANN	BP	RegNetY-16G	✗	✗	84.0	-	1	82.90
		ViT-B	✗	✗	86.0	-	1	82.30
		Swin-B	✗	✗	88.0	-	1	**83.50**
		T2T-ViT	✗	✗	64.3	-	1	82.30
		UniFormer-128	✗	✗	65.5	-	1	83.30
QANN†	BP	UniFormer-128	✗	✗	66.5	63.7	1	81.63
DieT-SNN	HT	VGG16	✓	✓	138.4	-	5	69.00
Hybrid-STDP	HT	VGG16	✓	✓	138.4	-	250	65.19
Fast-SNN	A2S	VGG16	✓	✓	138.4	-	7	72.95
MST	A2S	Swin-T(BN)	✓	✓	28.5	-	512	78.51
SpikeZIP-TF	A2S	SViT-B-32Level	✓	✓	86.6	403.2	64	82.71
SpikingFormer	DT	Spikingformer-8-512	✓	✓	29.7	7.46	4	74.79
		Spikingformer-8-768	✓	✓	66.3	13.7	4	75.85
E-SpikeFormer	DT	E-SpikeFormer	✓	✗	19.0	5.90	4	79.80
			✓	✗	83.0	19.1	4	83.20
QKFormer	DT	HST-10-384	✗	✓	16.5	22.0/105.9$^{\triangle}$	4	78.80
		HST-10-768	✗	✓	65.0	38.9/236.2$^{\triangle}$	4	84.22
MS-QKFormer	DT	HST-10-384	✓	✓	16.5	5.52	4	76.48
		HST-10-768	✓	✓	65.0	6.79	1	77.78
SDFormerV2	DT	Meta-SpikeFormer	✓	✓	15.1	16.7	4	74.10
			✓	✓	31.3	32.8	4	77.20
			✓	✓	55.4	52.4	4	80.00
UniSpike	HT	UniFormer-64	✓	✓	16.2	4.68	4	76.90
	HT	UniFormer-128	✓	✓	66.5	18.0	4	**80.81**

Table 4: **Experimental results on CIFAR-10, CIFAR-100 and CIFAR10-DVS.** *CF* is the abbreviation of CIFAR. The best results are in **bold**, the runner-up results are in gray.

Methods	**Category**	**Arch.**	**Param.**	**T**	**CF10**	**CF100**	**Arch.**	**Param.**	**T**	**CF10-DVS**
ResNet-19	ANN	Conv.	12.6	1	94.97	75.35	n/a	n/a	n/a	n/a
Transformer	ANN	4-384	9.32	1	96.73	81.02	n/a	n/a	n/a	n/a
Spikformer	SNN	4-384	9.32	4	95.19	77.86	2-256	2.57	16	80.9
Spikingformer	SNN	4-384	9.32	4	95.81	78.21	2-256	2.57	16	81.3
CML	SNN	4-384	9.32	4	96.04	80.02	2-256	2.57	16	80.9
S-Transformer	SNN	4-384	10.3	4	95.60	78.40	2-256	2.57	16	80.0
QKFormer	SNN	4-384	6.74	4	96.18	81.15	2-256	1.50	16	84.0
UniSpike	SNN	UniFormer-T1	4.50	4	**96.44**	**81.83**	UniFormer-T2	2.15	8	**85.0**

and ImageNet (Deng et al., 2009), while CIFAR10-DVS (Hongmin et al., 2017) is used as the neuromorphic dataset. On ImageNet-1K, we adopt two UniFormer variants: 1) UniFormer-128, with an embedding dimension of 128 and $[2, 6, 8, 2]$ blocks across four stages; and 2) UniFormer-64, with an embedding dimension of 64 and $[2, 2, 8, 2]$ blocks. For CIFAR-10/100 and CIFAR10-DVS, we use smaller-embedding variants, denoted UniFormer-T1 and UniFormer-T2, for fair comparison with existing SNNs. Detailed architectures are provided in Appendix (Table A5). During quantization-aware training, activations are quantized to 10 levels. To satisfy the equivalence condition in Equation (8), we constrain the quantization scale s as $s < \min(\frac{|X_{\max}|\cdot\gamma}{|C_{\max}|}, \frac{|X_{\min}|\cdot\gamma}{|C_{\min}|})$. For the sigmoid function in Equation (14), we set $\mu = 4$. The total training time of UniFormer-128 on ImageNet-1K is 65 hours on 8 RTX-4090 GPUs, which is lower than Spike-Driven Transformer V2 (81.62 h) and QKFormer (137.21 h), the detailed comparison is shown in the Appendix (Section A3).

5.1 Main Results

Comparison with ANN Architectures. As shown in Table 3, we compare UniFormer with classic ANN architecture including ViT-B (Dosovitskiy, 2020), Swin-B (Liu et al., 2021), RegNet-Y

Table 5: **Operation counts and energy breakdown.** Initial patch embeddings are excluded. The energy model in Ottati et al. (2023) is used to estimate the energy including weight, activation, and neuron-state memory access. Computation consumption includes AC/MAC operations in SNNs. In (b), QKV combines query, key, and value projections; M and A denote the two attention multiplication stages, and Y is the output projection. "SDT-V2" is spike-driven transformer V2.

(a) Energy breakdown (mJ)

Energy Breakdown	SDT-V2	QKFormer	UniFormer	
			QANN	SNN
Weight Memory	282.2 (75.9%)	254.1 (35.0%)	66.69 (33.2%)	90.90 (65.8%)
Activation Memory	37.99 (10.2%)	247.7 (34.1%)	70.22 (35.0%)	26.99 (19.5%)
Neuron State Memory	1.030 (0.28%)	0.971 (0.13%)	0.000 (0.00%)	3.194 (2.31%)
Computation	50.68 (13.6%)	223.5 (30.8%)	63.70 (31.8%)	17.11 (12.4%)
Total	**371.9 (100%)**	**726.3 (100%)**	**200.6 (100%)**	**138.2 (100%)**

(b) Operation counts

Transformer Layers		SDT-V2	QKFormer		UniFormer
		GACs	GACs	GMACs	GACs
Self-Attention	QKV	13.743	0.000	12.485	5.953
	M	0.012	0.013	0.000	0.573
	A	0.033	0.018	0.000	0.388
	Y	6.288	0.419	0.000	0.875
MLP	FC1	2.529	0.000	18.497	7.490
	FC2	0.128	0.640	0.000	0.587
Conv Blocks		32.005	–	–	–
Patch Merging		1.575	1.433	17.110	3.144
Total Operation		**56.314**	2.522	48.092	19.009

(Radosavovic et al., 2020) and T2T-ViT (Yuan et al., 2021). UniFormer achieves the second-best accuracy (83.3%), less than the Swin-B (83.5%) with fewer parameters (67M versus 88M). Notably, UniFormer replaces softmax, layer normalization, and GeLU activation with unified-clip, indicating that Unified-Clip serves as an effective substitute during ANN training.

Comparison with SOTA SNN Algorithms. We compare UNISPIKE against prior DT SNNs and A2S SNNs in Table 3. E-Spikformer is omitted as it does not support step-by-step inference and its energy depends on specific neuromorphic hardware (Man et al., 2024). **Compared to QKFormer**, replacing SEW with the addition-only MS shortcut drops the HST-10-384 backbone by 2.32% (Table 3). UNISPIKE's accuracy surpasses MS-QKFormer with the HST-10-384 architecture. **Among other addition-only DT SNNs**, UNISPIKE surpasses Spike-Driven Transformer V2 at comparable parameters, benefiting from the conversion-stage initialization. **Against SpikeZIP-TF**, the SOTA A2S baseline, BPTT compresses the time-step, yielding 16× fewer time-steps at comparable accuracy.

Results on CIFAR and Neuromorphic Datasets On CIFAR and neuromorphic datasets, UNISPIKE achieves SOTA performance. UNISPIKE achieves 0.26% and 0.68% higher task accuracy on CIFAR10 and CIFAR100 compared to QKFormer with smaller parameters (4.50M versus 6.74M in QKFormer) at 4 time-step. On the temporal neuromorphic dataset CIFAR10-DVS, UNISPIKE SNN has 1% higher accuracy compared to QKFormer with fewer time-steps.

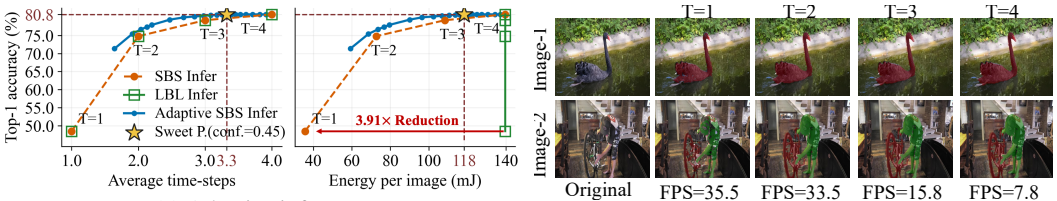
5.2 Operation & Energy Analysis with Addition-only

To demonstrate the benefits of addition-only SNNs, we adopt and extend the energy model from (Ottati et al., 2023) that incorporates memory-access energy, which is often neglected in prior SNN studies. The energy results are shown in Table 5. The model estimates inference energy by summing the 45-nm CMOS costs of arithmetic operations (4.6 pJ/MAC and 0.9 pJ/AC) and on-chip SRAM accesses (10 pJ per 64-bit access) for activations/spikes, weights, neuron states, and outputs, while considering spike sparsity and time-steps. The detailed energy model is provided in the Appendix (Section A4).

Table 5 shows three observations. 1) Memory access dominates SNN energy consumption in Table 5a, accounting for 86.4%, 68.0%, and 87.1% of total energy in STD-V2, QKFormer, and UniFormer, respectively, due to the higher cost of SRAM access compared with arithmetic operations. 2) UniFormer achieves 2.7% lower energy than STD-V2, the SOTA addition-only and step-by-step SNN, with higher accuracy in Table 3. UniFormer also has lower energy than the original ANN before A2S conversion, benefiting from 2.6× lower activation memory access (ternary spikes vs. FP32 activations in fake quantization) and 3.5× lower computation energy enabled by addition-only operations. 3) In QKFormer, as shown in Table 5b, MAC operations consume more energy than AC operations due to the SEW-shortcuts, which limits the energy benefits of addition-only computation.

5.3 Step-by-Step Inference on UniFormer

Step-by-step inference provides an advantage over layer-by-layer inference by enabling early outputs and reducing response latency. As illustrated in Figure 5(a), we compare both strategies using the same UniFormer on ImageNet-1K classification. Although they achieve identical time-step–



(a) Adaptive inference

(b) Outputs at different time-steps

Figure 5: (a) Accuracy versus classification time steps and energy for different inference strategies on ImageNet-1K. "SBS" and "LBL" are short for step-by-step and layer-by-layer. "Sweet P." denotes sweet point. (b) UniFormer outputs at different time steps on DAVIS2017.

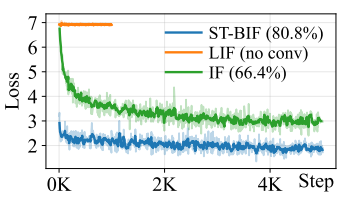
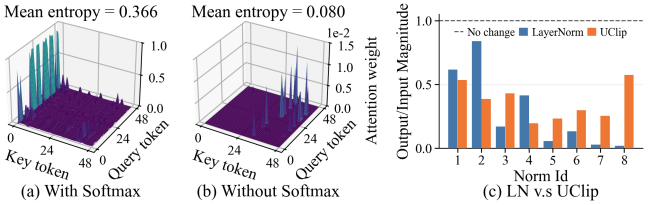


Figure 6: Training Loss of UniFormer with neuron models.



(a) With Softmax (b) Without Softmax (c) LN v.s UClip

Figure 7: Visualization of (a) the attention maps from the last block and (b) the output/input 99.9^{th} percentile magnitudes ratio $\frac{P_{99.9}(X_{out})}{P_{99.9}(X_{\text{in}})}$ for normalizations in the last blocks of four stages.

accuracy curves, layer-by-layer inference cannot expose intermediate predictions until all time-steps are processed, resulting in a fixed energy cost (vertical green line). In contrast, step-by-step inference produces outputs after each time-step and achieves a 3.91× energy reduction at the first time-step with the same accuracy. Furthermore, combined with SEENN's confidence-based early-exiting method (Li et al., 2023), step-by-step inference achieves a better energy–accuracy trade-off (blue curve), reducing time-steps by 17.5% and energy by 15.7% with only a 0.03% accuracy drop (80.83% to 80.80%).

We further evaluate UniFormer as the backbone of UniTrack (Wang et al., 2021) on DAVIS2017 (Pont-Tuset et al., 2017) for video object segmentation. As shown in Figure 5(b), step-by-step inference generates outputs after each time-step, enabling adaptive computation with scene complexity. For simple tracking cases (Image-1), T=1 provides sufficient mask quality and achieves 35.5 FPS, which is 4.5× faster than T=4 (7.8 FPS). For challenging cases (Image-2), additional time-steps improve mask quality, allowing UniFormer to trade-off accuracy and latency for real-time applications.

5.4 Ablation Study

Neuron Type. The equivalence between Unified-Clip and ST-BIF provides a suitable initialization for SNN fine-tuning. Figure 6 compares the training loss of UniFormer with different neurons on ImageNet-1K. ST-BIF achieves the lowest loss due to its equivalence with Unified-Clip. IF converges slowly since the equivalence only holds when time-steps approach infinity (Bu et al., 2023), while LIF fails to converge due to its leaky membrane dynamics.

Softmax&LayerNorm Replacement We train an ANN baseline with Softmax and LayerNorm for 300 epochs, achieving 83.6% Top-1 accuracy on ImageNet-1K, and evaluate the replacement effectiveness of UClip. As shown in Figure 7(a)(b), both attention weights exhibit spike-like distributions, indicating that UClip preserves token discrimination capability. Furthermore, UClip produces lower attention magnitudes than Softmax since it removes the sum-to-1 normalization, which also eliminates the attention sink phenomenon (Xiao et al., 2024). For normalization replacement, Figure 7(c) shows that both LayerNorm and UClip achieve input/output magnitude ratios below 1.0, demonstrating their outlier suppression capability for training stability.

6 Conclusion

We propose UniSpike, a hybrid learning framework to boost the performance of SNNs. UniSpike SNN incorporates an addition-only feature with step-by-step inference, constructing a more energy-efficient architecture than ANN, and is suitable for real-world applications on neuromorphic hardware.

AI USE STATEMENT

Large Language Models (LLMs) were used solely for language polishing, including sentence rephrasing, grammar checking, and improving readability. All research ideas, methodological design, code implementation, experiments, and analyses were independently developed and conducted by the authors. The authors take full responsibility for the content of the manuscript.

REFERENCES

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon. Spinalflow: An architecture and dataflow tailored for spiking neural networks. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 349–362. IEEE, 2020.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014. doi: 10.1109/ISSCC.2014.6757323.

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.

Rui Qian, Xin Lai, and Xirong Li. 3d object detection for autonomous driving: A survey. *Pattern Recognition*, 130:108796, 2022.

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.

Donghyun Lee, Yuhang Li, Youngeun Kim, Shiting Xiao, and Priyadarshini Panda. Spiking transformer with spatial-temporal attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13948–13958, June 2025.

Man Yao, Xuerui Qiu, Tianxiang Hu, Jiakui Hu, Yuhong Chou, Keyu Tian, Jianxing Liao, Luziwei Leng, Bo Xu, and Guoqi Li. Scaling spike-driven transformer with efficient spike firing approximation training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(4):2973–2990, 2025. doi: 10.1109/TPAMI.2025.3530246.

Kang You, Zekai Xu, Chen Nie, Zhijie Deng, Qinghai Guo, Xiang Wang, and Zhezhi He. Spikezip-tf: Conversion is all you need for transformer-based snn. *arXiv preprint arXiv:2406.03470*, 2024.

Yangfan Hu, Qian Zheng, Xudong Jiang, and Gang Pan. Fast-snn: fast spiking neural network by converting quantized ann. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

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

Chenlin Zhou, Han Zhang, Zhaokun Zhou, Liutao Yu, Liwei Huang, Xiaopeng Fan, Li Yuan, Zhengyu Ma, Huihui Zhou, and Yonghong Tian. Qkformer: Hierarchical spiking transformer using q-k attention, 2024b. URL https://arxiv.org/abs/2403.16552.

Tong Bu, Wei Fang, Jianhao Ding, PengLin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. *arXiv preprint arXiv:2303.04347*, 2023.

Ziqing Wang, Yuetong Fang, Jiahang Cao, Qiang Zhang, Zhongrui Wang, and Renjing Xu. Masked spiking transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1761–1771, 2023.

Nidal Abuhajar, Zhewei Wang, Marc Baltes, Ye Yue, Li Xu, Avinash Karanth, Charles D. Smith, and Jundong Liu. Three-stage hybrid spiking neural networks fine-tuning for speech enhancement. *Frontiers in Neuroscience*, Volume 19 - 2025, 2025. ISSN 1662-453X. doi: 10.3389/fnins.2025.1567347. URL https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2025.1567347.

Marc Baltes, Nidal Abujahar, Ye Yue, Charles D. Smith, and Jundong Liu. Joint ann-snn co-training for object localization and image segmentation, 2023. URL https://arxiv.org/abs/2303.12738.

Man Yao, Jiakui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. *arXiv preprint arXiv:2404.03663*, 2024a.

Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer. *Advances in neural information processing systems*, 36, 2024b.

Kai Malcolm and Josue Casco-Rodriguez. A comprehensive review of spiking neural networks: Interpretation, optimization, efficiency, and best practices, 2023. URL https://arxiv.org/abs/2303.10780.

Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*, 2020.

Nitin Rathi and Kaushik Roy. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, 2020.

Jiahao Han, Ning Xiao, Wanting Yang, Shichao Luo, Jun Zhao, Yan Qiang, Suman Chaudhary, and Juanjuan Zhao. Ms-resnet: disease-specific survival prediction using longitudinal ct images and clinical data. *International Journal of Computer Assisted Radiology and Surgery*, 17(6):1049–1057, 2022.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers, 2023. URL https://arxiv.org/abs/2309.08586.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization, 2025. URL https://arxiv.org/abs/2503.10622.

Minhyeok Lee. Gelu activation function in deep learning: A comprehensive mathematical analysis and performance, 2023. URL https://arxiv.org/abs/2305.12073.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *technical report*, 2009.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Li Hongmin, Liu Hanchao, Ji Xiangyang, Li Guoqi, and Shi Luping. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11, 2017.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020. doi: 10.1109/CVPR42600.2020.01044.

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet, 2021. URL https://arxiv.org/abs/2101.11986.

Yao Man, Ole Richter, Guangshe Zhao, Ning Qiao, Yannan Xing, Wang Dingheng, Tianxiang Hu, Wei Fang, Tugba Demirci, Michele Marchi, Tianyi Yan, Carsten Nielsen, Sadique Sheik, Chenxi Wu, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip. *Nature Communications*, 15, 05 2024. doi: 10.1038/s41467-024-47811-6.

Fabrizio Ottati, Chang Gao, Qinyu Chen, Giovanni Brignone, Mario R Casu, Jason K Eshraghian, and Luciano Lavagno. To spike or not to spike: A digital hardware perspective on deep learning acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(4): 1015–1025, 2023.

Yuhang Li, Tamar Geller, Youngeun Kim, and Priyadarshini Panda. Seenn: Towards temporal spiking early exit neural networks. *Advances in Neural Information Processing Systems*, 36:63327–63342, 2023.

Zhongdao Wang, Hengshuang Zhao, Ya-Li Li, Shengjin Wang, Philip H. S. Torr, and Luca Bertinetto. Do different tracking tasks require different appearance models?, 2021. URL https://arxiv.org/abs/2107.02156.

Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 DAVIS Challenge on Video Object Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895, 2024.

Eric W Weisstein. Heaviside step function. *https://mathworld. wolfram. com/*, 2002.

Sadri Hassani and Sadri Hassani. Dirac delta function. *Mathematical Methods: For Students of Physics and Related Fields*, pages 139–170, 2009.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020. URL https://arxiv.org/abs/2003.10555.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers, 2022. URL https://arxiv.org/abs/2106.08254.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019. URL https://arxiv.org/abs/1909.13719.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018. URL https://arxiv.org/abs/1710.09412.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6022–6031, 2019. doi: 10.1109/ICCV.2019.00612.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016. URL https://arxiv.org/abs/1603.09382.

Samuel G. Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation, 2021. URL https://arxiv.org/abs/2103.10158.

Table A1: The Notations in Backward Gradient Formula Derivation

Symbol	Description
$\mathcal{L}$	Loss function
w_i	synaptic weights connected to i-th neuron
X_t	Input at t time-step
Y_t	Output spikes at t time-step
V_t	Membrane potential at t time-step
V_{thr}	Membrane threshold for neuron firing
S_t	Spike tracer at t time-step
$\hat{V}_t$	Membrane potential after charging before firing at t time-step
$S_{\min}$	The minimum value of spike tracer
$S_{\max}$	The maximum value of spike tracer
Θ	Fire decision function of ST-BIF Neuron You et al. (2024)
θ	Step function Weisstein (2002)
ε	A small value

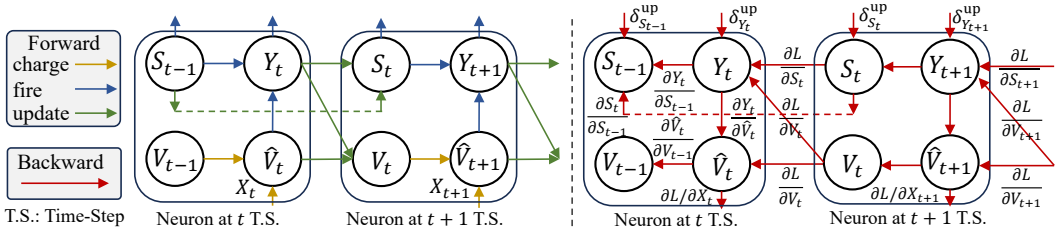


Figure A1: **Computation Graph of ST-BIF Neuron.**

Appendix

A1 BPTT FOR SNN WITH ST-BIF NEURON

The forward procedure of ST-BIF neuron is different from LIF neuron. Consequently, deriving the backward gradient of ST-BIF neuron is essential for implementing the BPTT algorithm on the SNN comprised of ST-BIF neurons. For the convenience of derivation, we summarize the meaning of notations appearing in Table A1.

A1.1 FEED-FORWARD OF ST-BIF NEURON

Before the derivation of backward gradient, we display the forward path of ST-BIF neuron in Figure A1(left). ST-BIF neuron charges the membrane potential V_{t-1} with the input X_t to get $\hat{V}_t$ (yellow arrows in Figure A1), fires spikes according to the firing decision function Θ (blue arrows in Figure A1) and update the $S_{t-1}, \hat{V}_t$ to the states at the next time-step (green arrows in Figure A1). To simplify the derivation, we set $C = 1$. The feed-forward definition of ST-BIF neuron is below:

$$\hat{V}_t = V_{t-1} + (\sum_i W_i \cdot X_{i,t}) \tag{A1}$$

$$\begin{aligned} Y_t &= \Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) \\ \Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) &= \theta(\hat{V}_t - V_{\text{thr}}) \cdot \theta(S_{\max} - S_{t-1} - \varepsilon) - \theta(-\hat{V}_t - \varepsilon) \cdot \theta(S_{t-1} - S_{\min} - \varepsilon) \end{aligned} \tag{A2}$$

$$V_t = \hat{V}_t - V_{\text{thr}} \times Y_t; \quad S_t = S_{t-1} + \Theta(\hat{V}_t, V_{\text{thr}}, S_{t-1}) \tag{A3}$$

where Equation (A1),Equation (A2),Equation (A3) are the charging, firing and updating procedures in ST-BIF neuron, respectively.

A1.2 BACKWARD OF ST-BIF NEURON

After forwarding, we calculate the gradients through the opposite direction of the forward computing graph. As displayed in the backward pass in Figure A1 (right), ST-BIF neuron at t time-step receives

partial derivatives of the loss function $\mathcal{L}$ to membrane potential V_t and spike tracer S_t from the same neuron at $t+1$ time-step, *i.e.*, $\partial\mathcal{L}/\partial V_t, \partial\mathcal{L}/\partial S_t$, as well as the partial derivatives of $\partial\mathcal{L}/\partial Y_t$ from neurons in next layer (red arrows outside the neuron at t time-step in Figure A1).

After receiving these partial derivatives, the neuron calculates the partial derivatives of the loss function $\mathcal{L}$ to membrane potential V_{t-1}, spike tracer S_{t-1} and input X_t, *i.e.*, $\partial\mathcal{L}/\partial V_{t-1}, \partial\mathcal{L}/\partial S_{t-1}$ and $\partial\mathcal{L}/\partial X_t$ for the $t-1$ time-step. Then, the neuron transmits $\partial\mathcal{L}/\partial V_{t-1}$ and $\partial\mathcal{L}/\partial S_{t-1}$ to the same neuron at $t-1$ time-step, and sends $\partial\mathcal{L}/\partial X_t$ to the neurons in the last layer. In summary, we need to derive three partial derivatives: $\partial\mathcal{L}/\partial V_{t-1}, \partial\mathcal{L}/\partial S_{t-1}$ and $\partial\mathcal{L}/\partial X_t$.

According to the computation graph in Figure A1 and chain rule, we write the calculation formulation for the three partial derivatives:

$$\begin{aligned}
&\frac{\partial\mathcal{L}}{\partial V_{t-1}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Y_t}\frac{\partial Y_t}{\partial \hat{V}_t}\frac{\partial \hat{V}_t}{\partial V_{t-1}} + \frac{\partial\mathcal{L}}{\partial V_t}\frac{\partial \hat{V}_t}{\partial V_{t-1}},\ \frac{\partial\mathcal{L}}{\partial S_{t-1}} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Y_t}\frac{\partial Y_t}{\partial S_{t-1}} + \frac{\partial\mathcal{L}}{\partial S_t}\frac{\partial S_t}{\partial S_{t-1}} \\
&\frac{\partial\mathcal{L}}{\partial X_t} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Y_t}\frac{\partial Y_t}{\partial \hat{V}_t}\frac{\partial \hat{V}_t}{\partial X_t} + \frac{\partial\mathcal{L}}{\partial V_t}\frac{\partial \hat{V}_t}{\partial X_t},\ \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Y_t} = \frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t}\frac{\partial S_t}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial V_t}\frac{\partial V_t}{\partial Y_t}
\end{aligned} \tag{A4}$$

where $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}Y_t}$ contains all the partial derivatives transmitted to Y_t, including $\frac{\partial\mathcal{L}}{\partial Y_t}$, $\frac{\partial\mathcal{L}}{\partial S_t}$ and $\frac{\partial\mathcal{L}}{\partial V_t}$ (the red arrows to Y_t in Figure A1). Then, combining with the firing decision function of ST-BIF neuron defined in Equation (A2), we calculate the partial derivatives in Equation (A4) and have:

$$\frac{\partial \hat{V}_t}{\partial V_{t-1}} = \frac{\partial S_t}{\partial Y_t} = \frac{\partial \hat{V}_t}{\partial X_t} = 1, \frac{\partial V_t}{\partial Y_t} = -V_{\mathrm{thr}} \tag{A5}$$

$$\frac{\partial S_t}{\partial S_{t-1}} = 1 - \frac{\partial Y_t}{\partial S_{t-1}} \tag{A6}$$

$$\frac{\partial Y_t}{\partial \hat{V}_t} = \frac{\partial\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial \hat{V}_t},\ \frac{\partial Y_t}{\partial S_{t-1}} = \frac{\partial\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial S_{t-1}} \tag{A7}$$

$$\frac{\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial \hat{V}_t} = \theta'(\hat{V}_t - V_{\mathrm{thr}}) \cdot \theta(S_{\max} - S_{t-1} - \varepsilon) + \theta'(-\hat{V}_t - \varepsilon) \cdot \theta(S_{t-1} - S_{\min} - \varepsilon) \tag{A8}$$

$$\frac{\partial\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial S_{t-1}} = -\theta(\hat{V}_t - V_{\mathrm{thr}}) \cdot \theta'(S_{\max} - S_{t-1} - \varepsilon) - \theta(-\hat{V}_t - \varepsilon) \cdot \theta'(S_{t-1} - S_{\min} - \varepsilon) \tag{A9}$$

where θ' is Dirac delta function Hassani and Hassani (2009), which is the derivative of θ. Since Dirac delta function is hard to implement in hardware, we use the sigmoid function as the surrogate function of θ'. Then, we substitute Equation (A5)-Equation (A9) to Equation (A4) and finally get:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial V_{t-1}} &= (\frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\partial Y_t}{\partial \hat{V}_t} + \frac{\partial\mathcal{L}}{\partial V_t} \\
\frac{\partial\mathcal{L}}{\partial S_{t-1}} &= (\frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\partial Y_t}{\partial S_{t-1}} + \frac{\partial\mathcal{L}}{\partial S_t}(1 - \frac{\partial Y_t}{\partial S_{t-1}}) \\
\frac{\partial\mathcal{L}}{\partial X_t} &= (\frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\partial Y_t}{\partial \hat{V}_t} + \frac{\partial\mathcal{L}}{\partial V_t}
\end{aligned} \tag{A10}$$

where we do not substitute $\frac{\partial Y_t}{\partial \hat{V}_t}$ and $\frac{\partial Y_t}{\partial S_{t-1}}$ for the convenient of reading. After simply transforming, we finally get:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial V_{t-1}} &= (\frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial \hat{V}_t} + \frac{\partial\mathcal{L}}{\partial V_t} \\
\frac{\partial\mathcal{L}}{\partial S_{t-1}} &= (\frac{\partial\mathcal{L}}{\partial Y_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\partial\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial S_{t-1}} + \frac{\partial\mathcal{L}}{\partial S_t} \\
\frac{\partial\mathcal{L}}{\partial X_t} &= (\frac{\partial\mathcal{L}}{\partial Y_t} + \frac{\partial\mathcal{L}}{\partial S_t} - V_{\mathrm{thr}} \times \frac{\partial\mathcal{L}}{\partial V_t})\frac{\Theta(\hat{V}_t, V_{\mathrm{thr}}, S_t)}{\partial \hat{V}_t} + \frac{\partial\mathcal{L}}{\partial V_t}
\end{aligned} \tag{A11}$$

With Equation (A11), we correctly estimate the gradient for ST-BIF neuron and make the SNN in UNISPIKE support the BPTT training algorithm.

A2 EXPERIMENT SETUPS

A2.1 TRAINING SETTINGS

For the reproduction of our experiment results, we show detailed training settings in Tables A2 to A4, including training hyperparameters, data augmentations, and other training technologies.

Table A2: **ANN Pre-training, Quantization-Aware Training and SNN Training Setting on ImageNet (UniFormer).**

config	ANN	QANN	SNN
optimizer	AdamW	AdamW	AdamW
learning rate	1e-3	1e-4	2.5e-5
weight decay	0.05	0.05	0.0005
optimizer momentum	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999
layer-wise lr decay Clark et al. (2020); Bao et al. (2022)	None	None	None
batch size	1024	1024	128
learning rate schedule	cosine decay	cosine decay	cosine decay
warmup epochs	20	5	0
training epochs	300	40	10
augmentation	RandAug (9, 0.5) Cubuk et al. (2019)	RandAug (9, 0.5) Cubuk et al. (2019)	RandAug (9, 0.5) Cubuk et al. (2019)
label smoothing Szegedy et al. (2016)	0.1	0.1	0.1
mixup Zhang et al. (2018)	0.8	0.5	None
cutmix Yun et al. (2019)	1.0	1.0	None
drop path Huang et al. (2016)	0.1	0.1	0.1

Table A3: **ANN pretraining, Quantization-Aware Training and SNN Training Setting on CIFAR100 (UniFormer-T1).**

config	ANN	QANN	SNN
optimizer	AdamW	AdamW	AdamW
learning rate	4e-3	4e-4	5e-4
weight decay	0.05	0.05	0.05
optimizer momentum	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999
layer-wise lr decay Clark et al. (2020); Bao et al. (2022)	None	None	None
batch size	1024	512	128
learning rate schedule	cosine decay	cosine decay	cosine decay
warmup epochs	10	5	5
training epochs	300	100	10
augmentation	RandAug (9, 0.5) Cubuk et al. (2019)	RandAug (9, 0.5) Cubuk et al. (2019)	RandAug (9, 0.5) Cubuk et al. (2019)
label smoothing Szegedy et al. (2016)	0.1	0.1	0.1
mixup Zhang et al. (2018)	0.5	0.5	None
cutmix Yun et al. (2019)	1.0	1.0	None
drop path Huang et al. (2016)	0.1	0.1	0.1

A2.2 NETWORK ARCHITECTURE DETAILS

Table A5 displays the architecture details of UNISPIKE, including UniFormer, UniFormer-T1 and UniFormer-T2. All the patch embedding and patch merging blocks in UNISPIKE SNN are convolution

Table A4: **ANN pretraining, Quantization-Aware Training and SNN Training Setting on CIFAR10-DVS (UniFormer-T2).**

config	ANN	QANN	SNN
optimizer	AdamW	AdamW	AdamW
learning rate	2e-3	5e-3	5e-5
weight decay	0.5	0.5	0.0005
optimizer momentum	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999
layer-wise lr decay Clark et al. (2020); Bao et al. (2022)	None	None	None
batch size	512	256	128
learning rate schedule	cosine decay	cosine decay	cosine decay
warmup epochs	10	0	0
training epochs	300	100	10
augmentation	SNNAugmentWide Müller and Hutter (2021)	SNNAugmentWide Müller and Hutter (2021)	SNNAugmentWide Müller and Hutter (2021)
label smoothing Szegedy et al. (2016)	0.1	0.1	0.1
mixup Zhang et al. (2018)	0.5	0.5	0.5
cutmix Yun et al. (2019)	1.0	1.0	1.0
drop path Huang et al. (2016)	0.1	0.1	0.1

Table A7: Training Cost comparison with SD-FormerV2 and QKFormer.

		SD-FormerV2	QKFormer	UniFormer
Stage1	Catag.	SNN(T=1)	SNN(T=4)	ANN
	Ep./Time	200/58.31h	300/137.21h	300/38.54h
Stage2	Catag.	SNN(T=4)	-	QANN
	Ep./Time	20/23.32h	-	40/12.80h
Stage3	Catag.	-	-	SNN(T=1)
	Ep./Time	-	-	10/13.66h
	Total	81.62h	137.21h	**65.00h**

Table A8: Training time breakdown on RTX 4090. "LN.", "UClip", and "SM." are LayerNorm, Softmax, and Unified Clip.

Stage	**LN.+SM.**	**LN.+UClip**	**UClip**
ANN	41.99h	41.23h	38.54h
QANN	13.57h	13.49h	12.80h
SNN	14.80h	14.46h	13.66h
Total	70.36h	69.18h	65.00h

residual blocks in ResNet. The input resolution of all the datasets is 224x224 to fully utilize the pretrained ANN on ImageNet.

Table A5: The Network Architecture Details of UNISPIKE.

Name	**Resolution**	**Patch Size**	**Window Size**	**Embedding Dim**	**Depths**	**Num Heads**
UniFormer-64	224×224	4	7	64	[2, 2, 8, 2]	[4, 8, 16, 32]
UniFormer-128	224×224	4	7	128	[2, 6, 8, 2]	[4, 8, 16, 32]
UniFormer-T1	224×224	4	7	36	[2, 2, 6, 2]	[1, 3, 6, 12]
UniFormer-T2	224×224	4	7	24	[2, 2, 6, 2]	[1, 3, 6, 12]

A3 TRAINING COST ANALYSIS

We compare the training cost of Spike-Driven Transformer V2, QKFormer, and UniFormer on 8 RTX 4090 GPUs in Table A7. UniFormer achieves the lowest training time, requiring only 65.00 hours despite using the largest number of training epochs (350 epochs). This efficiency mainly comes from its hybrid-training pipeline, where most epochs are spent on ANN pre-training, which is much faster than multi-step SNN training. In addition, Unified Clip replaces softmax, LayerNorm, and GeLU with a simple element-wise operation, further reducing the training overhead. As shown in Table A8, Unified Clip reduces the total training time from 70.36 to 65.00 hours compared with the UniFormer variant using LayerNorm and softmax, yielding a 7.6% reduction on RTX 4090 GPUs.

Table A6: Notations Used in Energy Estimation

Symbol	Description
$E_{\text{add}}, E_{\text{sub}}, E_{\text{mult}}, E_{\text{comp}}$	Energy per addition, subtraction, multiplication, and comparison
$E_{\text{rd}}, E_{\text{wr}}$	Energy per FP32 read and write
$E_{\text{rdtot}}, E_{\text{wrtot}}$	Total operand-read and output-write energy
$E_{\text{MAC}}, E_{\text{AC}}, E_{\text{act}}$	MAC, spike-accumulation, and activation-function energy
$E_{\text{state}}, E_{\text{neuron}}$	State-update energy and full neuron-processing energy
$E_{\text{ANN}}, E_{\text{SNN}}$	ANN and SNN energy within the specified operator scope
$E^V_{\text{rd}}, E^V_{\text{wr}}, E^S_{\text{rd}}, E^S_{\text{wr}}$	Membrane-potential and spike-tracer read/write energy
b, b_v, b_s	Generic, numerical-value, and spike storage bit widths
N_{rd}, L	Fan-in per output and number of processed token rows
$d_{\text{in}}, d_{\text{out}}, d, d_f, d_h$	Input, output, model, FFN hidden, and attention-head dimensions
G, N	Number of independent products (# head) and tokens per attention window
m, k, n	Dimensions of operands (m, k) and (k, n)
t, T	Current time-step index and total executed steps
$X, W, Y;\ A, B$	Linear input/weight/output matrices; matrix-product operands
$Q, K, V;\ A$	Query/key/value matrices; attention weights in AV
$x_t, y_t;\ \bar{X}_{t-1}, \bar{Y}_{t-1}$	Current spike increments and their preceding cumulative sums
ΔC_t	Increment of the cumulative attention product
$\gamma, \gamma_{x,t}, \gamma_{y,t}, \gamma_t$	Operand, incremental-input, and output-spike nonzero fractions
$\text{nnz}(\cdot)$	Number of nonzero elements
X_t, Y_t	Input-current and output-spike feature maps
$V_t, \hat{V}_t, V_{\text{thr}}$	Membrane potential after reset, before reset, and firing threshold
S_t, Θ	Cumulative signed spike tracer and ST-BIF firing function

A4 Energy Estimation

We follow the read–compute–state–write decomposition in Sec. IV-A of Ottati et al. (2023), extending it to Transformer matrix products and signed neurons. Notations are summarized in Table A6. Weights, numerical activations, and states are FP32. We use the following costs in pJ:

$$E_{\text{add}} = E_{\text{sub}} = E_{\text{comp}} = 0.9\text{pJ}, \quad E_{\text{mult}} = 3.7\text{pJ}, \quad E_{\text{rd}} = E_{\text{wr}} = 5\text{pJ}/32\text{bit}. \tag{A12}$$

Here, E_{add}, E_{sub}, E_{mult}, and E_{comp} denote the energy costs of one scalar addition, subtraction, multiplication, and comparison, respectively; E_{rd} and E_{wr} refer to energy consumption accessing to one FP32 value. Each 64-bit access costs 10 pJ and packs two FP32 values. A b-bit value therefore costs $(b/32)E_{\text{rd}}$ or $(b/32)E_{\text{wr}}$. We use $b = 1$ for binary spikes and $b = 2$ for ternary spikes; fake-quantized QANN activations remain 32-bit. Access counts are amortized, with no operand-cache reuse.

Basic energy model For one output neuron with fan-in N_{rd}, Ottati et al. (2023) decomposes ANN and SNN energy as

$$E_{\text{ANN}} = E_{\text{rdtot}} + E_{\text{MAC}} + E_{\text{act}} + E_{\text{wrtot}}, \tag{A13}$$

$$E_{\text{SNN}} = E_{\text{rdtot}} + E_{\text{AC}} + E_{\text{state}} + E_{\text{wrtot}}. \tag{A14}$$

Here, E_{ANN} and E_{SNN} denote the energy for one ANN output neuron and one SNN output neuron per time step, respectively. The terms are defined as follows under dense execution.

E_{rdtot} accounts only for reading the inputs and synaptic weights:

$$E^{\text{ANN}}_{\text{rdtot}} = N_{\text{rd}} \underbrace{(E_{\text{rd}} + E_{\text{rd}})}_{\text{one activation read + one weight read}} = 2N_{\text{rd}}E_{\text{rd}},$$

$$E^{\text{SNN}}_{\text{rdtot}} = N_{\text{rd}} \underbrace{\left(\frac{b_s}{b_v}E_{\text{rd}} + E_{\text{rd}}\right)}_{\text{one spike read + one weight read}}.$$

Here, E_{rd} is the energy of reading one b_v-bit numerical value, and b_s is the spike bit width.

E_{MAC} and E_{AC} account for the synaptic computation:

$$E_{\mathrm{MAC}} = N_{\mathrm{rd}}(E_{\mathrm{mult}} + E_{\mathrm{add}}), \qquad E_{\mathrm{AC}} = N_{\mathrm{rd}} E_{\mathrm{add}}.$$

Each ANN connection requires a multiplication and an accumulation; each active SNN connection requires only an accumulation. E_{act} accounts for the ANN activation function; for ReLU, $E_{\mathrm{act}} = E_{\mathrm{comp}}$, corresponding to one comparison with zero. E_{state} includes both computation and memory accesses for updating the SNN membrane state. For the soft-reset LIF neuron in the worst case,

$$E_{\mathrm{state}} = \underbrace{E_{\mathrm{rd}}^{\mathrm{state}}}_{\text{state read}} + \underbrace{E_{\mathrm{mult}} + E_{\mathrm{add}}}_{\text{leakage and integration}} + \underbrace{E_{\mathrm{comp}} + E_{\mathrm{sub}}}_{\text{threshold and reset}} + \underbrace{E_{\mathrm{wr}}^{\mathrm{state}}}_{\text{state write}}.$$

Here, the superscript state identifies membrane-state accesses; their costs equal E_{rd} and E_{wr} when the state uses the same b_v-bit representation.

E_{wrtot} accounts only for writing the final output activation or spike:

$$E_{\mathrm{wrtot}}^{\mathrm{ANN}} = E_{\mathrm{wr}}, \qquad E_{\mathrm{wrtot}}^{\mathrm{SNN}} = \frac{b_s}{b_v} E_{\mathrm{wr}}.$$

Here, E_{wr} is the energy of writing one b_v-bit value. State writes are already included in E_{state} and are not counted again. We inherit this decomposition, set $b_v = 32$ and $b_s \in \{1, 2\}$, and replace each term using measured activity and the implemented operator/neuron. Then, we provide the detailed energy model for each layer in Transformer blocks.

Energy model for Linear layers and FFN. For Linear layer $Y = XW$, let $X \in \mathbb{R}^{L \times d_{\mathrm{in}}}$ and $W \in \mathbb{R}^{d_{\mathrm{in}} \times d_{\mathrm{out}}}$. Here, L is the number of processed token rows, including executed time-steps for SNNs, and d_{in} and d_{out} are the input and output dimensions. Each of the Ld_{out} outputs has fan-in $N_{\mathrm{rd}} = d_{\mathrm{in}}$.

Following the preceding read–compute–write decomposition, a dense ANN Linear layer consumes

$$\begin{aligned} E_{\mathrm{ANN}}^{\mathrm{Linear}} &= E_{\mathrm{rdtot}}^{\mathrm{ANN}} + E_{\mathrm{MAC}} + E_{\mathrm{wrtot}}^{\mathrm{ANN}} \\ &= Ld_{\mathrm{out}} \left[d_{\mathrm{in}} \left(2E_{\mathrm{rd}} + E_{\mathrm{mult}} + E_{\mathrm{add}} \right) + E_{\mathrm{wr}} \right]. \end{aligned} \tag{A15}$$

Here, $2E_{\mathrm{rd}}$ accounts for one input-activation read and one weight read per product, and E_{wr} accounts for writing one output value.

For a spike-driven SNN Linear layer with zero skipping,

$$\begin{aligned} E_{\mathrm{SNN}}^{\mathrm{Linear}} &= E_{\mathrm{rdtot}}^{\mathrm{SNN}} + E_{\mathrm{AC}} + E_{\mathrm{wrtot}}^{\mathrm{SNN}} \\ &= Ld_{\mathrm{out}} \left[d_{\mathrm{in}} \left(\frac{b_s}{b_v} E_{\mathrm{rd}} + \gamma E_{\mathrm{rd}} + \gamma E_{\mathrm{add}} \right) + E_{\mathrm{wr}} \right]. \end{aligned} \tag{A16}$$

Here, $\gamma = \mathrm{nnz}(X)/(Ld_{\mathrm{in}})$ is the fraction of nonzero input spikes. The term $(b_s/b_v)E_{\mathrm{rd}}$ accounts for scanning each input spike, while γE_{rd} and γE_{add} account for weight reads and accumulations triggered by nonzero spikes. Without zero skipping, set $\gamma = 1$. Unlike the output of a spiking neuron, the Linear output is a b_v-bit synaptic current; its write therefore costs E_{wr}, not $(b_s/b_v)E_{\mathrm{wr}}$.

These expressions exclude bias; each enabled output-bias addition incurs $E_{\mathrm{rd}} + E_{\mathrm{add}}$. An FFN applies the corresponding expression to FC1 and FC2, with dimensions (d, d_f) and (d_f, d), where d is the model dimension and d_f the hidden dimension. Activation and neuron-state costs are added separately. QKV and output projections follow the same rule.

Energy model for attention matrix products. For G independent products AB, let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. Here, G counts the processed attention heads and windows, including executed time-steps when applicable; m and n are the output dimensions, and $k = N_{\mathrm{rd}}$ is the accumulation length per output. Unlike Linear layers, both operands are activations, so their reads are charged to activation memory.

For dense ANN attention, the energy is

$$\begin{aligned} E_{\mathrm{ANN}}^{\mathrm{Attention}} &= E_{\mathrm{rdtot}}^{\mathrm{ANN}} + E_{\mathrm{MAC}} + E_{\mathrm{wrtot}}^{\mathrm{ANN}} \\ &= Gmn \left[k(2E_{\mathrm{rd}} + E_{\mathrm{mult}} + E_{\mathrm{add}}) + E_{\mathrm{wr}} \right]. \end{aligned} \tag{A17}$$

Here, each scalar product reads two b_v-bit operands and performs one MAC, while each output is written once.

For SNN attention with one spike operand and one numerical operand, zero-skipping execution gives

$$E_{\mathrm{SNN}}^{\mathrm{Attention}} = E_{\mathrm{rdtot}}^{\mathrm{SNN}} + E_{\mathrm{AC}} + E_{\mathrm{wrtot}}^{\mathrm{SNN}} = Gmn\left[k\left(\frac{b_s}{b_v}E_{\mathrm{rd}} + \gamma E_{\mathrm{rd}} + \gamma E_{\mathrm{add}}\right) + E_{\mathrm{wr}}\right]. \tag{A18}$$

Here, γ is the nonzero fraction of the spike operand: $\mathrm{nnz}(A)/(Gmk)$ for left gating or $\mathrm{nnz}(B)/(Gkn)$ for right gating, with nonzeros counted across all G products. The spike operand is scanned densely, whereas numerical operand reads and accumulations occur only for nonzero spikes. If both operands are spikes, replace γE_{rd} by $\gamma(b_s/b_v)E_{\mathrm{rd}}$. The output remains a b_v-bit accumulated value. Only one operand controls zero skipping; no product of the two operands' sparsity fractions is assumed. Fixed spike scales are assumed absorbed into the numerical representation, with explicit scaling charged separately.

For QANN, QK^{T} uses $(m, k, n) = (N, d_h, N)$ and AV uses (N, N, d_h), where N is the token count per window and d_h is the channel dimension per head. SDT-V2 and QKFormer self-attention instead compute $K^{\mathsf{T}}V$ with (d_h, N, d_h), followed by $Q(K^{\mathsf{T}}V)$ with (N, d_h, d_h). The first product has two spike operands; the second has a spike operand and a numerical intermediate. QKFormer's token-attention reductions and elementwise gates are counted separately.

Energy model for incremental attention in UniFormer. UniFormer inherits the spike-equivalent self-attention of SpikeZIP-TF You et al. (2024), which computes

$$\Delta C_t = (x_t + \bar{X}_{t-1})y_t + x_t\bar{Y}_{t-1}. \tag{A19}$$

Here, x_t and y_t are the spike increments at time step t, and $\bar{X}_{t-1}$ and $\bar{Y}_{t-1}$ are their cumulative sums before that step. The equation represents G independent matrix products with operand shapes (m, k) and (k, n). Thus, each product has mn outputs with fan-in k. For the QK^{T} stage, the right operands are transposed before applying these dimensions.

The nonzero fractions of the spike increments are

$$\gamma_{x,t} = \frac{\mathrm{nnz}(x_t)}{Gmk}, \qquad \gamma_{y,t} = \frac{\mathrm{nnz}(y_t)}{Gkn}. \tag{A20}$$

Here, nonzeros are counted across all G products. The first product accumulates numerical values only when y_t is nonzero, while the second does so only when x_t is nonzero. Their computation energy is

$$E_{\mathrm{AC}} = Gmkn(\gamma_{x,t} + \gamma_{y,t})E_{\mathrm{add}}. \tag{A21}$$

Following the same read–compute–write decomposition, the read energy is

$$E_{\mathrm{rdtot}}^{\mathrm{SNN}} = 2Gmkn\frac{b_s}{b_v}E_{\mathrm{rd}} + Gmkn(\gamma_{x,t} + \gamma_{y,t})E_{\mathrm{rd}} + Gmk\left(\frac{b_s}{b_v} + 1\right)E_{\mathrm{rd}}. \tag{A22}$$

Here, the first term scans one spike operand for each of the two products; the factor 2 counts these two scans. The second term reads the numerical partners of nonzero spikes. The last term reads x_t and $\bar{X}_{t-1}$ to form their sum. As before, b_s and b_v denote spike and numerical-value bit widths, respectively.

The output-write energy is

$$E_{\mathrm{wrtot}}^{\mathrm{SNN}} = (Gmk + Gmn)E_{\mathrm{wr}}. \tag{A23}$$

Here, Gmk numerical values are written for $x_t + \bar{X}_{t-1}$, and Gmn for ΔC_t. Dot-product accumulators and the two product results are combined locally, without additional buffer writes.

The total incremental-attention energy is therefore

$$E_{\mathrm{SNN}}^{\mathrm{Incremental}} = E_{\mathrm{rdtot}}^{\mathrm{SNN}} + E_{\mathrm{AC}} + E_{\mathrm{wrtot}}^{\mathrm{SNN}} + (Gmk + Gmn)E_{\mathrm{add}}. \tag{A24}$$

Here, the additional additions form the intermediate sum (Gmk) and combine the two products (Gmn). All reads and writes above belong to activation memory. Neuron-state updates, bias, scaling, clipping, and operand preparation are charged separately.

Energy model for LIF and ST-BIF neurons. For an input feature map $X_t \in \mathbb{R}^{m\times n}$ at time-step t, each element updates one neuron. Thus, mn neuron updates are performed at each step. Here, X_t contains the numerical synaptic currents produced by the preceding Linear or attention operator.

For LIF neurons, the membrane update is

$$\hat{V}_t = V_{t-1} + \frac{X_t - V_{t-1}}{2}, \tag{A25}$$

followed by threshold comparison and hard reset to zero. Here, V_{t-1} and $\hat{V}_t$ are the membrane feature maps before and after integration. The state-update energy at step t is

$$E^{\mathrm{LIF}}_{\mathrm{state},t} = mn\left(E^V_{\mathrm{rd}} + E_{\mathrm{sub}} + E_{\mathrm{mult}} + E_{\mathrm{add}} + E_{\mathrm{comp}} + E^V_{\mathrm{wr}}\right). \tag{A26}$$

Here, each neuron reads and writes one membrane value, with one subtraction, multiplication, addition, and comparison. Hard reset selects zero without an additional arithmetic operation.

For ST-BIF neurons, the updates are

$$\hat{V}_t = V_{t-1} + X_t, \qquad Y_t = \Theta(\hat{V}_t, V_{\mathrm{thr}}, S_{t-1}), \tag{A27}$$

$$V_t = \hat{V}_t - V_{\mathrm{thr}}Y_t, \qquad S_t = S_{t-1} + Y_t. \tag{A28}$$

Here, Y_t is the signed output-spike feature map, V_{thr} is the firing threshold, and S_t is the cumulative signed spike tracer. The firing function Θ tests the membrane potential and tracer bounds. Defining the output firing fraction as $\gamma_t = \mathrm{nnz}(Y_t)/(mn)$, the state-update energy is

$$\begin{aligned} E^{\mathrm{STBIF}}_{\mathrm{state},t} = mn\big[&E^V_{\mathrm{rd}} + E^S_{\mathrm{rd}} + E^V_{\mathrm{wr}} + E^S_{\mathrm{wr}} \\ &+ E_{\mathrm{add}} + 4E_{\mathrm{comp}} + \gamma_t(E_{\mathrm{sub}} + E_{\mathrm{add}})\big]. \end{aligned} \tag{A29}$$

Here, the superscripts V and S distinguish membrane and spike-tracer accesses. Both are read and written for every feature-map element. Each update requires one integration addition and four comparisons; each nonzero spike additionally triggers a membrane reset and a tracer update. When a prefire correction is applied at step t, another mnE_{add} is charged. Both states use b_v bits, so $E^V_{\mathrm{rd}} = E^S_{\mathrm{rd}} = E_{\mathrm{rd}}$ and $E^V_{\mathrm{wr}} = E^S_{\mathrm{wr}} = E_{\mathrm{wr}}$.

Following the read–state-update–write decomposition, the per-step neuron energies are

$$E^{\mathrm{LIF}}_{\mathrm{neuron},t} = mnE_{\mathrm{rd}} + E^{\mathrm{LIF}}_{\mathrm{state},t} + \frac{mn}{b_v}E_{\mathrm{wr}}, \tag{A30}$$

$$E^{\mathrm{STBIF}}_{\mathrm{neuron},t} = mnE_{\mathrm{rd}} + E^{\mathrm{STBIF}}_{\mathrm{state},t} + \frac{2mn}{b_v}E_{\mathrm{wr}}. \tag{A31}$$

Here, the first term reads the input-current feature map; the last writes one-bit LIF or two-bit ST-BIF output spikes. The tracer write is already included in E^S_{wr} and is distinct from the spike write. The preceding operator already accounts for writing X_t.

For T executed steps, initialization is included once:

$$E^{\mathrm{LIF}}_{\mathrm{neuron}} = \sum_{t=1}^{T} E^{\mathrm{LIF}}_{\mathrm{neuron},t} + mnE^V_{\mathrm{wr}}, \tag{A32}$$

$$E^{\mathrm{STBIF}}_{\mathrm{neuron}} = \sum_{t=1}^{T} E^{\mathrm{STBIF}}_{\mathrm{neuron},t} + mn(E^V_{\mathrm{wr}} + E^S_{\mathrm{wr}}). \tag{A33}$$

State accesses remain dense even without firing; only event-dependent arithmetic scales with γ_t.

Reporting. We sum the terms above with separately traced activation, quantization, normalization, and residual costs, then average over 1,000 fixed images. The breakdown separates computation, weight access, activation access, and neuron-state access; E_{state} is split into its arithmetic and memory components. The main table excludes initial patch embedding, but includes later-stage downsampling. These are modeled deployment energies, not measured GPU energies.