ACT-SNN: ACTIVITY-CONSTRAINED TWO-STAGE CONVERSION FOR EFFICIENT SPIKING LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) incur substantial computational and energy costs during inference, motivating their conversion into energy-efficient spiking neural networks (SNNs). Although existing LLM-to-SNN methods have achieved high conversion fidelity, this does not necessarily guarantee efficient inference: accurate discrete representations can require high spike activity and thus higher energy consumption, while layer-wise temporal accumulation leads to serial inference latency. We propose **A**ctivity-**C**onstrained **T**wo-Stage SNN Conversion (ACT-SNN), a two-stage framework for accurate, sparse, and low-latency LLM-to-SNN conversion. In the first stage, our Basis-Augmented Multi-Threshold Neuron (BA-MTN) jointly learns basis transformations and channel-wise thresholds under an explicit activity constraint, decoupling reconstruction optimization from spike activity control while enlarging the feasible space for a favorable reconstruction–activity trade-off. In the second stage, the Basis-Augmented Symmetric Integrate-and-Fire (BA-SIF) neuron provides an equivalent temporal realization with causal step-by-step spike propagation, enabling pipelined cross-layer execution without layer-wise temporal accumulation; relaxation steps and differential operators further compensate for temporal propagation delays and nonlinear operations. Experiments across LLaMA-2, LLaMA-3, and Qwen2.5 models ranging from 7B to 70B parameters demonstrate that ACT-SNN achieves a strong accuracy–energy trade-off under both full-precision and 4-bit weight settings. In particular, under 4-bit weights, ACT-SNN attains energy ratios of 0.016–0.022, reducing estimated inference energy by 49–64% over representative W4A4 QANN baselines while maintaining competitive zero-shot accuracy.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable performance across a wide range of language understanding and generation tasks (Brown et al., 2020), but their growing scale and extensive dense matrix multiplications incur substantial computational and energy costs during inference. Spiking neural networks (SNNs), in contrast, can replace dense matrix multiplications with sparse spike-driven accumulations on neuromorphic hardware, offering a promising path toward energy-efficient inference (Merolla et al., 2014; Davies et al., 2018; Roy et al., 2019). However, directly training large-scale spiking models remains challenging due to the non-differentiability of spike generation and the additional complexity introduced by temporal dynamics (Neftci et al., 2019). LLM-to-SNN conversion therefore provides a more practical alternative by directly reusing pretrained LLM parameters, offering an attractive pathway to construct large-scale spiking LLMs without retraining from scratch (Rueckauer et al., 2017; Xing et al., 2025).

Existing LLM-to-SNN methods can be broadly viewed as a two-stage process, where continuous activations are first mapped to discrete spike representations and then realized as temporal spike sequences. In the first stage, the main challenge is to preserve activation fidelity under the limited representational capacity of discrete spike states. This becomes particularly difficult in LLMs due to highly non-uniform activation distributions and extreme outliers (Xiao et al., 2023). In the second stage, the main challenge is to faithfully realize the discrete representation through temporal spike dynamics. Finite inference steps and cross-layer propagation can introduce temporal errors, while

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.

nonlinear operations and temporal accumulation are generally non-commutative, making network-level equivalence difficult to preserve (Chen et al., 2026).

At the first stage, existing LLM-to-SNN conversion methods (Xing et al., 2025; Zheng et al., 2026) primarily improve discrete representation fidelity through more expressive discretization, coding, and calibration schemes. However, these methods do not explicitly constrain the spike activity of the learned discrete representation during optimization. Consequently, a high-fidelity representation may still incur unnecessary spike events, leading to additional synaptic operations (SynOps) and higher dynamic energy consumption in event-driven SNNs (Sengupta et al., 2019; Sorbaro et al., 2020). We therefore explicitly incorporate spike activity into the conversion objective, targeting high-fidelity discrete representations under a prescribed activity budget.

At the second stage, high-fidelity temporal realization is essential for preserving the converted representation. Existing high-fidelity methods (Luo et al., 2024; Chen et al., 2026) often rely on full-window reconstruction, where spike responses are accumulated over the complete temporal window. Such reconstruction typically requires layer-by-layer execution, introducing serial dependencies across network depth and consequently increasing inference latency (Park et al., 2020; Huang et al., 2025). In contrast, step-by-step execution allows spikes to propagate to downstream layers immediately after generation, enabling pipelined cross-layer execution and better matching event-driven streaming on neuromorphic hardware (Fang et al., 2023; You et al., 2026). Under a cross-layer pipelined implementation, the critical-path latency can be reduced from $\mathcal{O}(TL)$ for layer-by-layer execution to $\mathcal{O}(T + L)$ for step-by-step execution, where T and L denote the temporal window and network depth, respectively. However, step-by-step propagation violates the full-window reconstruction assumption, as each layer receives only a partial upstream spike sequence at each timestep rather than the complete temporal response. The key challenge is therefore to preserve conversion fidelity under causal step-by-step propagation without relying on layer-wise temporal accumulation.

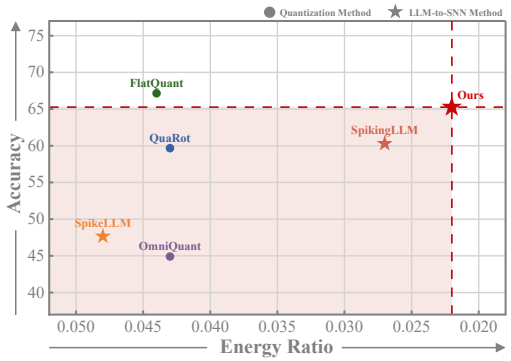


Figure 1: Accuracy–energy trade-off on LLaMA-2-7B under the 4-bit weight setting. Lower energy ratio and higher accuracy are preferred; ACT-SNN achieves a favorable trade-off over existing quantization and LLM-to-SNN methods.

In this paper, we propose **A**ctivity-**C**onstrained **T**wo-Stage SNN Conversion (ACT-SNN), a two-stage framework for efficient LLM-to-SNN conversion. Fig. 1 provides an overview of its empirical efficiency, showing that ACT-SNN maintains competitive accuracy at substantially lower energy. **In the first stage**, we introduce the Basis-Augmented Multi-Threshold Neuron (BA-MTN) to learn high-fidelity and spike-efficient discrete representations. BA-MTN jointly optimizes a learnable basis transformation and channel-wise thresholds under an activity-constrained calibration objective. By augmenting conventional threshold optimization with basis adaptation, BA-MTN enlarges the optimization space of discrete spike representations, providing greater flexibility to preserve reconstruction fidelity while suppressing spike activity and thereby achieving a more favorable reconstruction–activity trade-off. **In the second stage**, the learned BA-MTN representation is temporally realized using the Basis-Augmented Symmetric Integrate-and-Fire (BA-SIF) neuron to enable high-fidelity, low-latency inference with step-by-step spike propagation. BA-SIF provides an equivalent temporal realization of BA-MTN using symmetric unit-magnitude spikes, avoiding layer-wise temporal accumulation and enabling pipelined step-by-step inference. During inference, additional relaxation steps and differential operators are employed to compensate for temporal propagation delays and nonlinear operations thereby preserving network-level conversion fidelity.

Our main contributions are summarized as follows:

- We propose **A**ctivity-**C**onstrained **T**wo-Stage SNN Conversion (ACT-SNN), a two-stage LLM-to-SNN conversion framework that decouples activity-efficient discrete representation learning from low-latency step-by-step temporal realization.

- We introduce the Basis-Augmented Multi-Threshold Neuron (BA-MTN) with activity-constrained calibration. By jointly learning the representation basis and channel-wise thresholds, our method achieves a favorable trade-off between reconstruction fidelity and spike activity.
- We introduce the Basis-Augmented Symmetric Integrate-and-Fire (BA-SIF) neuron and theoretically establish its temporal equivalence to BA-MTN. Together with relaxation steps and differential operators, BA-SIF enables step-by-step inference while preserving the conversion fidelity.
- Extensive experiments demonstrate strong efficiency and fidelity: ACT-SNN scales to models up to 70B parameters with step-by-step inference, and reduces the estimated arithmetic energy by 98.4% on Qwen2.5-32B-Inst. with only a 0.57-point accuracy drop.

2 Related Work

Representation Transformation for Discretization. The first stage of LLM-to-SNN conversion is closely related to activation quantization: both approximate continuous ANN activations with discrete representations while minimizing the resulting reconstruction error. This connection makes representation transformation techniques developed for LLM quantization particularly relevant. SmoothQuant uses equivalent channel-wise scaling to alleviate activation outliers (Xiao et al., 2023), while QuaRot and FlatQuant further exploit orthogonal or learnable transformations to reshape activation distributions into forms that are more amenable to low-bit discretization (Ashkboos et al., 2024; Sun et al., 2025). These methods show that optimizing the representation basis can substantially improve discretization fidelity. However, quantization and SNN conversion differ in both their optimization objectives and realization constraints. Quantization primarily targets numerical reconstruction under low-bit arithmetic, whereas SNNs exploit sparse spike events to replace activation-dependent multiplications with event-driven accumulations, making spike activity directly relevant to computational efficiency. Moreover, high-accuracy activation quantization can employ input-dependent dynamic scales, such as token-wise scaling (Yao et al., 2022; Xiao et al., 2023), while SNN conversion relies on fixed membrane thresholds to define consistent neuron dynamics over time. Consequently, a representation optimized solely for quantization error does not necessarily yield an efficient or physically compatible spiking realization. Our BA-MTN instead performs activity-constrained basis optimization with static channel-wise thresholds, balancing reconstruction fidelity against spike activity under the constraints of temporal spiking inference.

Spiking Language Models. Recent efforts toward spiking language models can be broadly divided into direct construction and LLM-to-SNN conversion. SpikeGPT directly trains a spiking generative language model with event-driven activations (Zhu et al., 2024), while SpikeLM develops elastic bi-spiking mechanisms for fully spike-driven language modeling (Xing et al., 2024). However, directly training spiking models becomes increasingly costly as model scale grows, motivating conversion-based approaches that reuse pretrained LLMs. SpikeLLM scales spiking representations to 7–70B parameter LLMs through generalized integrate-and-fire neurons and saliency-aware spike allocation (Xing et al., 2025). More recent methods further improve conversion fidelity and efficiency: FAS combines conversion-oriented fine-tuning with coarse-to-fine calibration for low-timestep inference (Chen et al., 2025), PhaseSNN introduces distribution-aware multi-granularity phase coding with learnable bases to reduce representation error (Zheng et al., 2026), and LAS develops specialized neurons and spike-equivalent Transformer components toward fully spike-driven LLM conversion (Chen et al., 2026). These methods focus on improving representation fidelity, latency, or spike-compatible operators. In contrast, our framework constrains spike activity during representation learning and targets faithful network-level realization under step-by-step propagation.

3 Method

3.1 Problem Formulation and Overview

Given a pretrained LLM, our goal is to convert it into a step-by-step SNN that preserves the pretrained functionality while maintaining low spike activity. We formulate this conversion as an activity-constrained reconstruction problem:

$$\min_{\Phi} \ \mathcal{L}_{\mathrm{rec}}(\Phi) \qquad \text{s.t.} \qquad \mathcal{A}(\Phi) \le \tau, \tag{1}$$

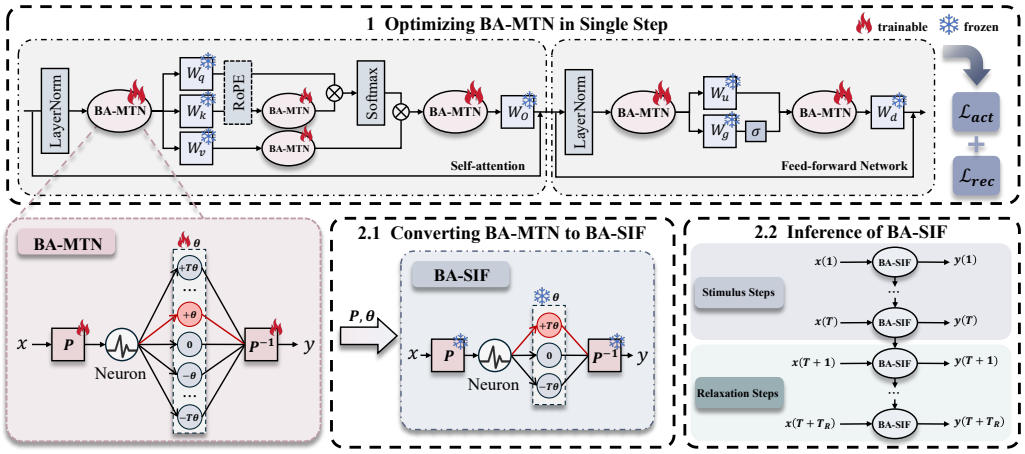


Figure 2: Overview of the ACT-SNN framework. **Stage I** performs activity-constrained single-step conversion using BA-MTNs, where a learnable basis transformation augments channel-wise threshold discretization. **Stage II** inherits the learned basis transformation P and thresholds θ to construct BA-SIF neurons, which realize the multi-level responses with unit-magnitude signed spikes under step-by-step propagation. Flame and snowflake symbols denote trainable and frozen parameters. T and T_R denote the stimulus steps and relaxation steps lengths, respectively

where Φ denotes the learnable conversion parameters, $\mathcal{L}_{\text{rec}}$ measures reconstruction error with respect to the pretrained ANN on a calibration set, $\mathcal{A}$ measures spike activity, and τ is the activity budget, i.e., the target upper bound on spike activity during calibration.

Directly converting an LLM into a step-by-step SNN is challenging because discrete representation learning and temporal spike realization are tightly coupled. As illustrated in Fig. 2, ACT-SNN decouples the conversion into two stages. Stage I constructs a single-step Multi-Threshold Neuron (MTN) representation, where a learnable basis transformation is introduced in addition to channel-wise membrane thresholds to expand the optimization space of spike discretization. The resulting representation is calibrated under the activity-constrained objective to jointly preserve reconstruction fidelity and suppress spike activity. Stage II equivalently unfolds the learned multi-threshold representation into unit-magnitude signed spikes, enabling high-fidelity step-by-step inference without directly optimizing through long-horizon spiking dynamics.

3.2 Activity-Constrained Conversion from LLMs to Multi-Threshold SNNs

Basis-Augmented Multi-Threshold Neuron (BA-MTN). A single-step MTN maps continuous activations onto integer spike levels through threshold-based discretization. Conventional MTNs perform this discretization in the original activation basis. Such a formulation leaves the discretization basis fixed and therefore offers limited flexibility in shaping the representation geometry before mapping activations onto discrete spike levels. For the highly non-uniform activations of pretrained LLMs, preserving reconstruction fidelity in this fixed basis may require higher spike activity.

We therefore propose the Basis-Augmented Multi-Threshold Neuron (BA-MTN) as the core representation module in Stage I, which introduces a learnable invertible basis transformation P in addition to positive channel-wise membrane thresholds θ. This augmented parameterization allows the discretization basis and channel-wise scales to be jointly optimized under the activity-constrained reconstruction objective. The single-step behavior of BA-MTN is formulated as

$$\begin{cases} s = \text{round}\left(\dfrac{xP}{\theta}\right), \\ y = (s \odot \theta)P^{-1}. \end{cases} \tag{2}$$

Here, x denotes the input activation, s is the integer-valued spike response, and y is the reconstructed activation. We preset a temporal horizon T for the subsequent temporal realization. During Stage I

calibration, s is further clipped to $[-T, T]$, matching the response range supported by this temporal budget. This clipping is applied only during Stage I calibration, and is removed during inference so that responses are not permanently bounded. The learnable parameter P adapts the basis in which discretization is performed, while θ controls the channel-wise discretization scale. Since θ is inherited as the membrane threshold in Stage II, it remains static during inference. The joint optimization of P and θ therefore allows the discretization space to adapt toward representations that preserve fidelity with lower spike activity.

To reduce the overhead of basis adaptation, we parameterize $P = P_1 \otimes P_2$ following FlatQuant (Sun et al., 2025). Assuming $P \in \mathbb{R}^{n\times n}$ and $P_1, P_2 \in \mathbb{R}^{\sqrt{n}\times\sqrt{n}}$, the Kronecker vectorization trick reduces the MACs required to compute xP to a fraction $2/\sqrt{n}$ of those required by a dense transformation. Moreover, neither the inverse transformation P^{-1} nor the channel-wise threshold scaling introduces additional online computation. For a downstream linear layer with weight W, we have

$$yW = (s \odot \theta)P^{-1}W = s\widetilde{W}, \qquad \widetilde{W} = \mathrm{diag}(\theta)P^{-1}W. \tag{3}$$

Thus, both the threshold scaling and inverse transformation are folded into the downstream weight offline. Consequently, the only additional online computation introduced by BA-MTN is the lightweight forward transformation xP, while the subsequent linear operation can be directly driven by the integer-valued spike response s.

Activity-Constrained Optimization of BA-MTN. While BA-MTN provides additional degrees of freedom for learning the spike representation, reconstruction-oriented calibration alone does not explicitly control spike activity. To instantiate the activity constraint in equation 1, a natural measure is the average magnitude of the integer-valued spike response s. However, the rounding operator induces a piecewise-constant objective with zero gradients almost everywhere. We therefore use the following continuous spike-activity surrogate:

$$\mathcal{A}(P, \theta) = \frac{1}{M}\left\|\frac{xP}{\theta}\right\|_{L_1}, \tag{4}$$

where M is the number of activation elements. Removing the rounding operation provides informative gradients and induces a smoother optimization landscape. This allows the activity objective to interact more consistently with the reconstruction loss, enabling the optimizer to search for low-activity directions that remain compatible with the local geometry of the reconstruction objective.

Using $\mathcal{A}$ as a differentiable proxy for spike activity, we relax the activity-constrained reconstruction problem in equation 1 with a quadratic exterior penalty. Specifically, we introduce the **activity loss**

$$\mathcal{L}_{\mathrm{act}}(P, \theta) = \frac{1}{2}\mathrm{ReLU}\left(\mathcal{A}(P, \theta) - \tau\right)^2, \tag{5}$$

leading to the optimization objective

$$\mathcal{L} = \mathcal{L}_{\mathrm{rec}} + \lambda\mathcal{L}_{\mathrm{act}}. \tag{6}$$

Here λ controls the activity-constraint strength. The activity penalty is activated only when $\mathcal{A}$ exceeds the budget τ, and the resulting constraint force smoothly vanishes as $\mathcal{A}$ approaches τ. This prevents unnecessary suppression of already feasible representations and allows the optimization to focus on reconstruction fidelity once the activity budget is satisfied. Further discussion of the constrained formulation is provided in Appendix A.

3.3 Conversion to Step-by-Step Unit-Spike SNNs

Basis-Augmented Symmetric Integrate-and-Fire (BA-SIF) Neuron. The BA-MTN introduced in equation 2 provides a single-step discrete representation but does not model temporal spike dynamics. To enable high-fidelity step-by-step inference, we temporally expand each integer-valued BA-MTN response into a sequence of unit-magnitude signed spikes while preserving the learned discrete representation. We realize this temporal expansion using the Basis-Augmented Symmetric

Integrate-and-Fire (BA-SIF) neuron. Under step-by-step propagation, the membrane dynamics of our BA-SIF neuron are defined as

$$\begin{cases} \hat{V}(t) = V(t-1) + x(t)P, \\ s(t) = \begin{cases} +1, & \hat{V}(t) > \frac{T\theta}{2}, \\ 0, & -\frac{T\theta}{2} \le \hat{V}(t) \le \frac{T\theta}{2}, \\ -1, & \hat{V}(t) < -\frac{T\theta}{2}, \end{cases} \\ V(t) = \hat{V}(t) - s(t) \odot T\theta, \\ y(t) = (s(t) \odot T\theta)P^{-1}. \end{cases} \tag{7}$$

where $V(t)$ denotes the membrane potential, $s(t) \in \{-1, 0, +1\}$ is the signed spike emitted at timestep t, and T is the preset temporal horizon introduced in Stage I, corresponding to the number of stimulus steps. Both P and θ are directly inherited from the calibrated BA-MTN and require no additional optimization, allowing the single-step representation learned in Stage I to be directly realized under step-by-step temporal propagation.

Theorem 1 (BA-MTN–BA-SIF Equivalence). *Let $\bar{x} = \frac{1}{T}\sum_{t=1}^{\infty} x(t)$ and $\bar{y} = \frac{1}{T}\sum_{t=1}^{\infty} y(t)$ denote the stimulus-normalized input and output of a BA-SIF neuron over infinite-time. Assuming $V(0) = 0$, then the neuron response satisfies*

$$\bar{y} = \left[\text{round}\left(\frac{\bar{x}P}{\theta}\right) \odot \theta\right] P^{-1}. \tag{8}$$

The proof is provided in Appendix B.1. This result shows that BA-SIF is an equivalent temporal realization of BA-MTN. Consequently, the discrete representation learned during calibration can be preserved when converted into step-by-step spiking dynamics.

Stimulus-Relaxation for Step-by-Step Inference. Although Theorem 1 establishes neuron-level equivalence, directly deploying BA-SIF neurons in a deep step-by-step network does not guarantee network-level conversion fidelity. At timestep t, each layer can only observe the causal prefix of the spike sequence generated by the preceding layer, resulting in temporal propagation delays across layers. Moreover, nonlinear operators generally do not commute with temporal accumulation.

Therefore, we adopt a stimulus-relaxation inference scheme. The network first receives the original input for fixed T stimulus steps, followed by T_R relaxation steps during which the external input is removed and bias terms are disabled. For nonlinear operators, we use a differential temporal realization (Huang et al., 2024) that preserves the accumulated nonlinear response. Detailed nonlinear conversion and inference schemes are provided in Appendix C.3 and Appendix C.4, respectively.

Corollary 1. *Consider a BA-MTN network and its step-by-step BA-SIF counterpart with the stimulus-relaxation inference above. As $T_R \to \infty$, the stimulus-normalized output of every BA-SIF layer converges to that of the corresponding BA-MTN layer.*

The proof is provided in Appendix B.2.

4 Experiments

4.1 Settings

Models and Evaluation. We evaluate our ACT-SNN on LLaMA-2-7B/13B/70B, LLaMA-3-8B, and Qwen2.5-7B/32B. For calibration, we randomly sample 128 sequences from WikiText-2 (WT2) (Merity et al., 2017), each with a sequence length of 2048 tokens. We report perplexity (PPL) on WikiText-2 and C4 (Raffel et al., 2020). For commonsense reasoning, we consider ARC-Challenge, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021). Unless otherwise specified, Avg.$_5$ denotes the mean accuracy over all five tasks. For the FP16-weight comparison, we additionally report Avg.$_4$ over ARC-Challenge, ARC-Easy, PIQA, and WinoGrande to match the evaluation protocol available for LAS (Chen et al., 2026) and PhaseSNN (Zheng et al., 2026).

Table 1: Comparison under the **non-quantized FP16-weight setting**. FP16 ANN results are included for reference only. Inference denotes the temporal propagation scheme. "–" denotes unavailable results. **Bold** denotes the best result among SNN methods.

Method	Model Type	Inference	T	LLaMA-2-7B				LLaMA-2-13B				LLaMA-2-70B			
				WT2↓	C4↓	Avg.$_4$ ↑	ER↓	WT2↓	C4↓	Avg.$_4$ ↑	ER↓	WT2↓	C4↓	Avg.$_4$ ↑	ER↓
FP16 (baseline)	ANN	–	–	5.47	6.97	67.25	1.00	4.88	6.47	69.89	1.00	3.32	5.71	74.71	1.00
LAS Chen et al. (2026)	SNN	layer-by-layer	6	45.50	–	65.65	–	–	–	–	–	–	–	–	–
PhaseSNN Zheng et al. (2026)	SNN	layer-by-layer	6	12.19	–	**65.70**	–	–	–	–	–	–	–	–	–
Ours	SNN	step-by-step	8	**6.70**	**9.40**	63.34	**0.569**	**5.43**	**7.51**	**69.75**	**0.556**	**3.71**	**6.04**	**73.12**	**0.482**

Method	Model Type	Inference	T	LLaMA-3-8B				Qwen2.5-7B				Qwen2.5-32B			
				WT2↓	C4↓	Avg.$_4$ ↑	ER↓	WT2↓	C4↓	Avg.$_4$ ↑	ER↓	WT2↓	C4↓	Avg.$_4$ ↑	ER↓
FP16 (baseline)	ANN	–	–	6.14	8.88	71.19	1.00	8.20	14.37	69.18	1.00	5.13	10.44	72.65	1.00
LAS Chen et al. (2026)	SNN	layer-by-layer	6	93.13	>100	**68.97**	–	–	–	–	–	–	–	–	–
PhaseSNN Zheng et al. (2026)	SNN	layer-by-layer	6	8.04	**12.25**	68.10	0.789	–	–	–	–	–	–	–	–
Ours	SNN	step-by-step	8	**7.59**	12.46	68.61	**0.601**	**8.05**	**13.78**	**70.43**	**0.571**	**5.58**	**10.71**	**73.31**	**0.462**

Calibration and Spiking Inference. All pretrained LLM parameters are frozen during calibration, and only the basis transformation P and neuron thresholds θ in BA-MTN are optimized. We set the activity-loss coefficient $\lambda = 1.0$ and activity budget $\tau = 1.0$. Our ACT-SNN performs inference in a step-by-step manner. Unless otherwise specified, we use $T = 8$ stimulus steps followed by $T_R = 8$ relaxation steps. During the stimulus steps, the network continuously receives the external input at each timestep. During the relaxation steps, no external input is provided, and all internal bias terms are set to zero. We evaluate ACT-SNN under both FP16-weight and INT4-weight settings, matching the weight precision adopted by prior full-precision conversion methods and quantized LLM-to-SNN baselines, respectively. Full calibration settings and implementation details are provided in Appendix C.

Arithmetic Energy Estimation. Following the MAC/AC-based energy accounting protocol in prior SNN work (Zheng et al., 2026), we estimate the arithmetic energy as

$$E_{\text{comp}} = N_{\text{MAC}} E_{\text{MAC}} + N_{\text{AC}} E_{\text{AC}}, \qquad \text{ER} = \frac{E_{\text{comp}}}{E_{\text{FP16-ANN}}}. \tag{9}$$

Following the 45-nm arithmetic energy model of Neseem et al. (2024), we use $1.50/0.40$ pJ for an FP16 MAC/AC and $0.065/0.015$ pJ for an INT4 MAC/AC, where the INT4 costs are extrapolated according to the reported precision-scaling rule. Detailed MAC/AC counting rules and operation statistics are provided in Appendix D.

4.2 Main Results

Full-precision conversion. As shown in Table 1, ACT-SNN achieves a favorable fidelity–energy trade-off under FP16 weights, with ER ranging from 0.462 to 0.601 across all evaluated models. Its conversion fidelity remains competitive with existing LLM-to-SNN methods while requiring substantially less arithmetic energy. For example, on LLaMA-3-8B, ACT-SNN achieves comparable performance to PhaseSNN while reducing ER from 0.789 to 0.601.

Beyond arithmetic efficiency, ACT-SNN enables step-by-step spike propagation, avoiding the layer-wise temporal synchronization required by LAS and PhaseSNN. It also scales to substantially larger models, including LLaMA-2-70B and Qwen2.5-32B, beyond the scales evaluated by these prior conversion methods. Notably, ER further decreases with model size within both model families, reaching 0.482 and 0.462 on the two largest models, respectively.

Low-bit conversion. As shown in Table 2, ACT-SNN remains effective under 4-bit weight quantization. Compared with SpikingLLM, ACT-SNN achieves higher zero-shot accuracy and lower ER on all shared models, improving Avg.$_5$ by over 4 points while further reducing arithmetic energy.

ACT-SNN is also competitive with strong QANN baselines. On Qwen2.5-7B and Qwen2.5-32B, its Avg.$_5$ remains within 0.23 points of FlatQuant, while reducing ER from 0.044 to as low as 0.016. Unlike dense low-bit QANNs, which still rely on low-bit MACs and typically require scale application and quantization/dequantization around quantized matrix multiplications, ACT-SNN converts most activation-dependent computation into spike-driven AC operations. These quantization-related

Table 2: Comparison under the **4-bit weight setting**. FP16 ANN results are included for reference only. QANN baselines use 4-bit weights with their corresponding activation settings reported in prior work. Inference denotes the temporal propagation scheme. "–" denotes unavailable results. **Bold** denotes the best result among SNN methods.

Method	Model Type	Inference	T	LLaMA-2-7B				LLaMA-2-13B				LLaMA-2-70B			
				WT2↓	C4↓	Avg.$_5$ ↑	ER↓	WT2↓	C4↓	Avg.$_5$ ↑	ER↓	WT2↓	C4↓	Avg.$_5$ ↑	ER↓
FP16 (baseline)	ANN	–	–	5.47	6.97	69.05	1.00	4.88	6.47	71.74	1.00	3.32	5.71	76.56	1.00
SmoothQuant Xiao et al. (2023)	QANN	–	–	83.12	77.27	–	0.043	35.88	43.19	–	0.043	26.01	34.61	–	0.043
OmniQuant Shao et al. (2024)	QANN	–	–	15.25	19.35	44.91	0.043	12.40	15.87	51.86	0.043	–	–	–	–
QuaRot Ashkboos et al. (2024)	QANN	–	–	8.56	11.86	59.67	0.043	6.10	8.67	66.38	0.043	4.14	6.42	73.24	0.043
FlatQuant Sun et al. (2025)	QANN	–	–	5.79	7.79	67.15	0.044	5.12	7.09	70.38	0.044	3.55	5.91	76.02	0.044
SpikeLLM(O) Xing et al. (2025)	SNN	–	–	11.46	14.45	47.65	0.048	9.56	12.48	50.64	0.049	–	–	–	–
SpikingLLM Xu et al. (2026)	SNN	step-by-step	16	10.99	13.78	60.28	0.027	7.80	10.32	65.36	0.031	–	–	–	–
PhaseSNN Zheng et al. (2026)	SNN	layer-by-layer	8	8.91	–	64.66	–	–	–	–	–	–	–	–	–
Ours	SNN	step-by-step	8	**6.63**	**9.23**	**65.26**	**0.022**	**5.39**	**7.44**	**70.04**	**0.020**	**3.65**	**5.97**	**74.64**	**0.016**

Method	Model Type	Inference	T	LLaMA-3-8B				Qwen2.5-7B-Inst.				Qwen2.5-32B-Inst.			
				WT2↓	C4↓	Avg.$_5$ ↑	ER↓	WT2↓	C4↓	Avg.$_5$ ↑	ER↓	WT2↓	C4↓	Avg.$_5$ ↑	ER↓
FP16 (baseline)	ANN	–	–	6.14	8.88	72.69	1.00	8.20	14.37	71.25	1.00	5.13	10.44	75.16	1.00
SmoothQuant Xiao et al. (2023)	QANN	–	–	210.19	187.93	–	0.043	–	–	–	–	–	–	–	–
OmniQuant Shao et al. (2024)	QANN	–	–	–	–	–	–	–	–	–	–	–	–	–	–
QuaRot Ashkboos et al. (2024)	QANN	–	–	10.60	17.19	62.17	0.043	–	–	–	–	6.95	12.17	70.62	0.043
FlatQuant Sun et al. (2025)	QANN	–	–	6.98	11.13	70.89	0.044	8.46	13.94	70.86	0.044	5.80	10.86	74.82	0.044
SpikeLLM(O) Xing et al. (2025)	SNN	–	–	–	–	–	–	–	–	–	–	–	–	–	–
SpikingLLM Xu et al. (2026)	SNN	step-by-step	16	10.34	15.38	63.70	0.027	–	–	–	–	–	–	–	–
Ours	SNN	step-by-step	8	**7.93**	**13.30**	**67.90**	**0.021**	**8.44**	**14.74**	**70.78**	**0.020**	**5.80**	**10.88**	**74.59**	**0.016**

Table 3: Ablation study on Qwen2.5-7B-Instruct. Activity denotes the average firing rate across all spiking layers (↓). **Bold** and underline denote the best and second-best results, respectively.

Configuration	WT2 ↓	C4 ↓	Avg.$_5$ ↑	Activity ↓
Baseline	1276.09	1408.14	44.60	1.53
+ Basis Augmentation	**7.98**	**13.54**	<u>71.67</u>	2.32
+ $\mathcal{L}_{\text{act}}$ (Ours)	<u>8.05</u>	<u>13.78</u>	**71.97**	<u>1.42</u>
w/o Relaxation Steps	13.88	25.43	59.12	**0.86**

overheads are not included in our arithmetic-only ER estimation, making the reported comparison conservative with respect to the event-driven SNN computation.

4.3 Ablation and Sensitivity Analysis

Component Ablation. As shown in Table 3, direct conversion without basis augmentation suffers from severe accuracy degradation. Introducing basis augmentation reduces WT2/C4 PPL from $1276.09/1408.14$ to $7.98/13.54$ and improves Avg.$_5$ from 44.60 to 71.67, but increases activity to 2.32. Adding $\mathcal{L}_{\text{act}}$ reduces activity by 38.8% while largely preserving performance. Removing relaxation significantly degrades performance, confirming its importance for temporal propagation.

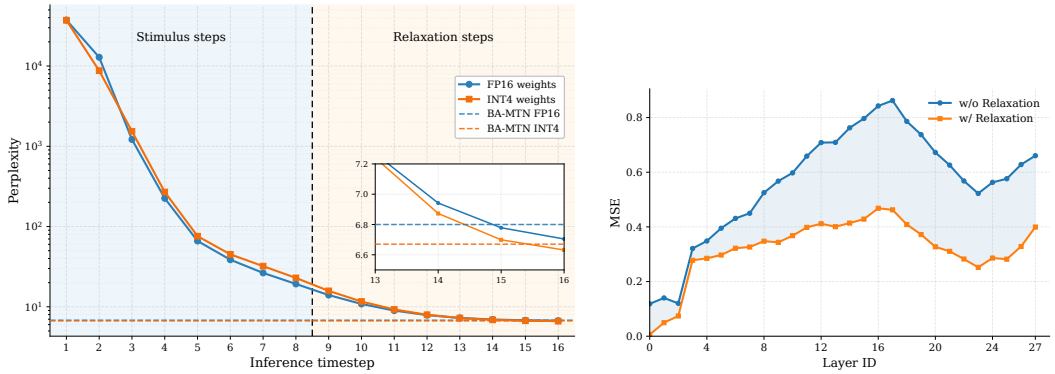
Hyperparameter Sensitivity. Table 4 shows that τ provides a direct control over the fidelity–activity trade-off. With τ increasing from 0.5 to 2.0, the average activity increases from 1.16 to 1.58, while model quality progressively recovers. In particular, Avg.$_5$ improves from 65.86 at $\tau = 0.5$ to 71.97 at $\tau = 1.0$, after which the performance largely saturates. An overly restrictive budget of $\tau = 0$ leads to optimization collapse. The method is relatively insensitive to λ around 1.0, which provides the best overall trade-off between performance and activity and is therefore used by default.

4.4 Analysis of Temporal Realization and Relaxation

Fig. 3 analyzes the temporal realization from BA-MTN to BA-SIF. For each weight-precision setting, the BA-MTN reference is obtained by directly evaluating the calibrated BA-MTN with the same weights, learned basis transformation P, and thresholds θ used to construct the corresponding BA-SIF model, before temporal expansion. As shown in Fig. 3a, perplexity drops rapidly during the $T = 8$ stimulus steps but has not yet fully converged at the end of the stimulus phase. It decreases further during relaxation steps, gradually approaching the corresponding BA-MTN reference under

Table 4: Sensitivity analysis of the activity budget τ and activity-loss coefficient λ on Qwen2.5-7B-Instruct. Activity denotes the average firing rate across all spiking layers (↓). **Bold** and underline denote the best and second-best results, respectively.

Hyperparameter	WT2 ↓	C4 ↓	Avg.$_5$ ↑	Activity ↓
Activity Budget τ				
0.00	137635.80	125786.85	38.74	1.42
0.50	13.10	22.27	65.86	**1.16**
0.75	8.87	14.93	69.67	1.25
1.00	**8.05**	13.78	**71.97**	1.42
2.00	8.21	**13.70**	**71.97**	1.58
Activity Loss Weight λ				
0.50	8.27	13.97	71.94	1.48
1.00	**8.05**	**13.78**	**71.97**	1.42
2.00	8.24	13.95	71.34	**1.35**



(a) Temporal convergence of BA-SIF toward BA-MTN.

(b) Layer-wise temporal realization error.

Figure 3: Effect of relaxation on temporal realization. (a) Perplexity of LLaMA-2-7B on WikiText-2 over inference timesteps, converging toward the BA-MTN reference. (b) Layer-wise MSE between BA-SIF and BA-MTN for Qwen2.5-7B-Instruct on WikiText-2, comparing inference with and without relaxation steps.

both weight settings. Since no additional external input is introduced during relaxation steps, this improvement reflects the continued propagation of residual temporal states through the network.

Consistently, Fig. 3b shows that the temporal realization error accumulates across layers, whereas relaxation steps substantially suppress this mismatch throughout the network, reducing the average layer-wise MSE by about 43%. These results confirm that relaxation steps mitigate incomplete temporal propagation and improve the fidelity of step-by-step realization.

5 Conclusion

In this work, we identify spike activity and serialized temporal execution as two key efficiency bottlenecks in high-fidelity LLM-to-SNN conversion. We therefore propose ACT-SNN, a two-stage conversion framework that separates discrete representation learning from temporal realization. ACT-SNN combines activity-constrained BA-MTN representation learning to reduce spike activity with equivalent BA-SIF temporal realization to enable step-by-step inference. Extensive experiments across LLaMA-2, LLaMA-3, and Qwen2.5 models demonstrate that ACT-SNN maintains high conversion fidelity while substantially reducing estimated arithmetic energy and supporting step-by-step inference. The framework scales consistently across different model families and sizes, from 7B-scale models to LLaMA-2-70B. An important next step is to deploy ACT-SNN on real neuromorphic hardware, where its practical energy savings and latency benefits can be directly validated.

AI USE STATEMENT

We used an LLM solely to aid and polish writing (wording, grammar, and style), including minor edits for clarity and concision, rephrasing author-written passages, harmonizing terminology/notation, and light LaTeX formatting (e.g., captions and tables). The LLM did not contribute to research ideation, model design, experiments, data analysis, or result interpretation. All technical content, equations, and claims were authored and verified by the authors.

ETHICS STATEMENT

This work aims to improve the inference efficiency of large language models through ANN-to-SNN conversion. The converted models may inherit the biases, harmful behaviors, and other limitations of their pretrained ANN counterparts. Our experiments use publicly available models and standard benchmarks without collecting personal or sensitive data. The reported energy-efficiency gains may also vary across hardware platforms and deployment settings.

REPRODUCIBILITY STATEMENT

Detailed calibration, optimization, and inference settings are provided in Appendix C, while energy estimation and operation statistics are documented in Appendix D. Theoretical derivations and proofs are given in Appendices A and B to support reproducibility of the reported results.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Long Chen, Xiaotian Song, Andy Song, Badong Chen, Jiancheng Lv, and Yanan Sun. FAS: fast ANN-SNN conversion for spiking large language models. *CoRR*, abs/2502.04405, 2025.

Long Chen, Xiaotian Song, and Yanan Sun. LAS: loss-less ANN-SNN conversion for fully spike-driven large language models. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 1730–1738. AAAI Press, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham N. Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios D. Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1): 82–99, 2018.

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.

Zihan Huang, Xinyu Shi, Zecheng Hao, Tong Bu, Jianhao Ding, Zhaofei Yu, and Tiejun Huang. Towards high-performance spiking transformers from ANN to SNN conversion. In Jianfei Cai, Mohan S. Kankanhalli, Balakrishnan Prabhakaran, Susanne Boll, Ramanathan Subramanian, Liang Zheng, Vivek K. Singh, Pablo César, Lexing Xie, and Dong Xu (eds.), *Proceedings of the 32nd ACM International Conference on Multimedia, MM 2024, Melbourne, VIC, Australia, 28 October 2024 - 1 November 2024*, pp. 10688–10697. ACM, 2024.

Zihan Huang, Wei Fang, Tong Bu, Peng Xue, Zecheng Hao, Wenxuan Liu, Yuanhong Tang, Zhaofei Yu, and Tiejun Huang. Differential coding for training-free ann-to-snn conversion. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

Xinhao Luo, Man Yao, Yuhong Chou, Bo Xu, and Guoqi Li. Integer-valued training and spike-driven inference spiking neural network for high-performance and energy-efficient object detection. In Ales Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (eds.), *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XXXII*, volume 15090 of *Lecture Notes in Computer Science*, pp. 253–272. Springer, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345 (6197):668–673, 2014.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7197–7206. PMLR, 2020.

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 Process. Mag.*, 36(6):51–63, 2019.

Marina Neseem, Conor McCullough, Randy Hsin, Chas Leichner, Shan Li, In Suk Chong, Andrew G. Howard, Lukasz Lew, Sherief Reda, Ville-Mikko Rautio, and Daniele Moro. Pikelpn: Mitigating overlooked inefficiencies of low-precision neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pp. 15996–16005. IEEE, 2024.

Seongsik Park, Sei Joon Kim, Byunggook Na, and Sungroh Yoon. T2FSNN: deep spiking neural networks with time-to-first-spike coding. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pp. 1–6. IEEE, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

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

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, 2021.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik. Optimizing the energy consumption of spiking neural networks for neuromorphic applications. *Frontiers in neuroscience*, 14:662, 2020.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. Flatquant: Flatness matters for LLM quantization. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 2023.

Xingrun Xing, Zheng Zhang, Ziyi Ni, Shitao Xiao, Yiming Ju, Siqi Fan, Yequan Wang, Jiajun Zhang, and Guoqi Li. Spikelm: Towards general spike-driven language modeling via elastic bi-spiking mechanisms. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54698–54714. PMLR / OpenReview.net, 2024.

Xingrun Xing, Boyan Gao, Zheng Liu, David A. Clifton, Shitao Xiao, Wanpeng Zhang, Li Du, Zheng Zhang, Guoqi Li, and Jiajun Zhang. Spikellm: Scaling up spiking neural network to large language models via saliency-based spiking. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

Zekai Xu, Hanqi Li, Kang You, Chen Nie, Qinghai Guo, Xiang Wang, and Zhezhi He. Spikingllm: A conversion-based method with window inhibition mechanism for spiking large language models. 2026.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Kang You, Chen Nie, Lee Jun Yan, Ziling Wei, Cheng Zou, Zekai Xu, Yu Feng, Honglan Jiang, and Zhezhi He. ELSA: an elastic snn inference architecture for efficient neuromorphic computing. In *53rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2026, Raleigh, NC, USA, June 27 - July 1, 2026*, pp. 2550–2566. IEEE, 2026.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4791–4800. Association for Computational Linguistics, 2019.

Hanyuan Zheng, Haozhen Zhang, Tianshuo Chen, Zhaogeng Liu, Yi Chang, and Bin Gu. Distribution-aware multi-granularity phase coding: Towards lower conversion error for spike-driven large language models. In *International Conference on Learning Representations*, volume 2026, pp. 80051–80076, 2026.

Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks. *Trans. Mach. Learn. Res.*, 2024, 2024.

A Constrained Optimization Perspective and Optimality Conditions

In this section, we provide a constrained-optimization interpretation of the activity objective introduced in the main text. Specifically, we show that the proposed activity loss naturally follows from a quadratic exterior penalty formulation and further relate the resulting objective to the Karush–Kuhn–Tucker (KKT) optimality conditions.

A.1 From Constrained Optimization to the Activity Loss

As described in the main text, activity-constrained calibration is formulated as

$$\min_{P,\theta} \mathcal{L}_{\mathrm{rec}}(P,\theta) \qquad \text{s.t.} \qquad \mathcal{A}(P,\theta) \le \tau, \tag{10}$$

where P and θ denote the learnable calibration parameters, $\mathcal{L}_{\mathrm{rec}}$ measures the reconstruction error, $\mathcal{A}$ measures the activity of the resulting spiking representation, and τ denotes the prescribed activity budget. This formulation treats reconstruction fidelity as the primary objective while explicitly constraining spike activity.

To convert equation 10 into an unconstrained optimization problem, we define

$$g(P,\theta) = \mathcal{A}(P,\theta) - \tau, \tag{11}$$

such that the activity constraint is equivalently written as

$$g(P,\theta) \le 0. \tag{12}$$

For an inequality constraint of this form, the quadratic exterior penalty method constructs the unconstrained objective

$$\mathcal{L}(P,\theta) = \mathcal{L}_{\mathrm{rec}}(P,\theta) + \frac{\lambda}{2}\left[\mathrm{ReLU}\left(g(P,\theta)\right)\right]^2, \tag{13}$$

where $\lambda > 0$ controls the strength of the penalty. Substituting equation 11 into equation 13 gives

$$\mathcal{L}(P,\theta) = \mathcal{L}_{\mathrm{rec}}(P,\theta) + \lambda \mathcal{L}_{\mathrm{act}}(P,\theta), \tag{14}$$

where

$$\mathcal{L}_{\mathrm{act}}(P,\theta) = \frac{1}{2}\mathrm{ReLU}\left(\mathcal{A}(P,\theta) - \tau\right)^2. \tag{15}$$

Therefore, the activity loss used in our calibration objective is exactly the quadratic exterior penalty associated with the activity constraint in equation 10.

This formulation differs fundamentally from applying an unconditional activity regularizer. When

$$\mathcal{A}(P,\theta) \le \tau, \tag{16}$$

the activity constraint is already satisfied and

$$\mathcal{L}_{\mathrm{act}}(P,\theta) = 0. \tag{17}$$

The optimization is therefore governed solely by the reconstruction objective. In contrast, when

$$\mathcal{A}(P,\theta) > \tau, \tag{18}$$

the constraint is violated and the penalty increases quadratically with the amount of violation. The activity objective therefore does not continuously force the activity toward zero, but instead suppresses excessive activity only when the prescribed budget is violated. This behavior is consistent with our goal of reducing spike activity without introducing unnecessary degradation in reconstruction fidelity.

For an infeasible point satisfying $\mathcal{A}(P,\theta) > \tau$, differentiating equation 14 yields

$$\nabla_{P,\theta}\mathcal{L} = \nabla_{P,\theta}\mathcal{L}_{\mathrm{rec}} + \lambda\left(\mathcal{A}(P,\theta) - \tau\right)\nabla_{P,\theta}\mathcal{A}. \tag{19}$$

Hence, the contribution of the activity objective is automatically scaled by the magnitude of the current constraint violation. A larger violation produces a stronger activity-reduction gradient, whereas this contribution smoothly vanishes as the activity approaches the feasible region.

A.2 Connection to the KKT Conditions

The relation between the constrained formulation and the proposed activity loss can be further understood through the KKT conditions. Since λ is already used as the penalty weight in equation 14, we use μ to denote the Lagrange multiplier associated with the activity constraint.

The Lagrangian of equation 10 is

$$\mathcal{J}(P, \theta, \mu) = \mathcal{L}_{\text{rec}}(P, \theta) + \mu \left(\mathcal{A}(P, \theta) - \tau \right), \tag{20}$$

where $\mu \geq 0$. Under a suitable constraint qualification, a local optimum $(P^\star, \theta^\star)$ satisfies the KKT conditions

$$\nabla_{P,\theta} \mathcal{L}_{\text{rec}}(P^\star, \theta^\star) + \mu^\star \nabla_{P,\theta} \mathcal{A}(P^\star, \theta^\star) = 0, \tag{21}$$

$$\mathcal{A}(P^\star, \theta^\star) - \tau \leq 0, \tag{22}$$

$$\mu^\star \geq 0, \tag{23}$$

$$\mu^\star \left(\mathcal{A}(P^\star, \theta^\star) - \tau \right) = 0. \tag{24}$$

The complementary-slackness condition in equation 24 characterizes two cases. If the activity constraint is inactive,

$$\mathcal{A}(P^\star, \theta^\star) < \tau, \tag{25}$$

then $\mu^\star = 0$, and the stationarity condition reduces to

$$\nabla_{P,\theta} \mathcal{L}_{\text{rec}}(P^\star, \theta^\star) = 0. \tag{26}$$

Thus, when the activity budget is satisfied with a strict margin, the constrained optimum requires no additional activity-reduction force.

If the activity constraint is active,

$$\mathcal{A}(P^\star, \theta^\star) = \tau, \tag{27}$$

then $\mu^\star$ may be positive, and the stationarity condition becomes

$$\nabla_{P,\theta} \mathcal{L}_{\text{rec}}(P^\star, \theta^\star) = -\mu^\star \nabla_{P,\theta} \mathcal{A}(P^\star, \theta^\star). \tag{28}$$

At such a solution, the reconstruction and activity gradients are locally balanced at the boundary of the feasible region. This characterizes the desired trade-off: further reduction in activity cannot be achieved locally without affecting the reconstruction objective.

The quadratic exterior penalty in equation 14 provides an unconstrained approximation to this KKT system. We define the penalty-induced multiplier as

$$\mu_\lambda(P, \theta) = \lambda \, \text{ReLU} \left(\mathcal{A}(P, \theta) - \tau \right). \tag{29}$$

The gradient of the penalized objective can then be expressed as

$$\nabla_{P,\theta} \mathcal{L} = \nabla_{P,\theta} \mathcal{L}_{\text{rec}} + \mu_\lambda(P, \theta) \nabla_{P,\theta} \mathcal{A}. \tag{30}$$

Equation equation 30 has the same form as the KKT stationarity condition in equation 21, with μ_λ acting as an implicit multiplier determined by the current constraint violation.

When $\mathcal{A}(P, \theta) \leq \tau$, we have

$$\mu_\lambda(P, \theta) = 0, \tag{31}$$

which is consistent with the inactive-constraint case in the KKT conditions. When $\mathcal{A}(P, \theta) > \tau$, the induced multiplier becomes

$$\mu_\lambda(P, \theta) = \lambda \left(\mathcal{A}(P, \theta) - \tau \right), \tag{32}$$

and increases with both the penalty weight and the amount of constraint violation.

For a finite λ, the minimizer of the quadratic exterior-penalty problem may remain slightly outside the feasible region when the corresponding constraint is active. This is a standard property of quadratic exterior penalties: a small positive constraint violation is required to generate a nonzero penalty gradient capable of balancing the reconstruction gradient. As λ increases, the same effective multiplier can be produced by a progressively smaller violation.

More formally, consider a sequence of minimizers $(P_\lambda, \theta_\lambda)$ of the penalized problem as λ increases. Under the standard regularity assumptions of the quadratic exterior penalty method, if this sequence converges to a regular solution of the original constrained problem, then

$$\mathrm{ReLU}\left(\mathcal{A}(P_\lambda, \theta_\lambda) - \tau\right) \to 0 \qquad \text{as } \lambda \to \infty. \tag{33}$$

Meanwhile, the induced multiplier

$$\mu_\lambda = \lambda\, \mathrm{ReLU}\left(\mathcal{A}(P_\lambda, \theta_\lambda) - \tau\right) \tag{34}$$

approaches the corresponding KKT multiplier under the usual regularity conditions. Therefore, the penalized problem progressively approaches the optimality conditions of the original activity-constrained formulation.

Overall, the constrained formulation explicitly expresses the desired optimization objective: minimizing reconstruction error subject to a prescribed activity budget. The proposed activity loss

$$\mathcal{L}_{\mathrm{act}}(P, \theta) = \frac{1}{2}\mathrm{ReLU}\left(\mathcal{A}(P, \theta) - \tau\right)^2 \tag{35}$$

is the corresponding quadratic exterior penalty, while its gradient induces an implicit multiplier that reflects the degree of constraint violation. This provides a principled justification for the final calibration objective

$$\mathcal{L} = \mathcal{L}_{\mathrm{rec}} + \lambda \mathcal{L}_{\mathrm{act}}, \tag{36}$$

which penalizes excessive spike activity while preserving reconstruction fidelity as the primary optimization objective.

B Proofs of Theoretical Results

B.1 Proof of Theorem 1

We prove the equivalence by considering the accumulated membrane dynamics of the BA-SIF neuron. Recall that its dynamics are given by

$$\hat{V}(t) = V(t-1) + x(t)P, \tag{37}$$

$$s(t) = \begin{cases} +1, & \hat{V}(t) > \dfrac{T\theta}{2}, \\ 0, & -\dfrac{T\theta}{2} \le \hat{V}(t) \le \dfrac{T\theta}{2}, \\ -1, & \hat{V}(t) < -\dfrac{T\theta}{2}, \end{cases} \tag{38}$$

$$V(t) = \hat{V}(t) - s(t) \odot T\theta, \tag{39}$$

$$y(t) = \left(s(t) \odot T\theta\right) P^{-1}. \tag{40}$$

Combining Eqs. equation 37 and equation 39 gives

$$V(t) = V(t-1) + x(t)P - s(t) \odot T\theta. \tag{41}$$

Let $\mathcal{T}$ denote a finite temporal realization horizon. Summing equation 41 from $t = 1$ to $\mathcal{T}$ yields

$$V(\mathcal{T}) - V(0) = \sum_{t=1}^{\mathcal{T}} x(t)P - \left(\sum_{t=1}^{\mathcal{T}} s(t)\right) \odot T\theta. \tag{42}$$

With the standard initialization $V(0) = 0$, define the accumulated signed spike count as

$$N(\mathcal{T}) = \sum_{t=1}^{\mathcal{T}} s(t). \tag{43}$$

Then

$$V(\mathcal{T}) = \sum_{t=1}^{\mathcal{T}} x(t)P - N(\mathcal{T}) \odot T\theta. \tag{44}$$

The BA-SIF neuron continues its temporal realization until all remaining membrane potential that exceeds a firing threshold has been emitted. Therefore, in the infinite-time limit, the residual membrane potential satisfies element-wise

$$-\frac{T\theta}{2} \leq V(\infty) \leq \frac{T\theta}{2}. \tag{45}$$

Let

$$N = \sum_{t=1}^{\infty} s(t) \tag{46}$$

denote the resulting total signed spike count. Taking $\mathcal{T} \to \infty$ in equation 44 gives

$$V(\infty) = \left(\sum_{t=1}^{\infty} x(t)\right) P - N \odot T\theta. \tag{47}$$

Dividing equation 47 element-wise by $T\theta$ and using equation 45, we obtain

$$-\frac{1}{2} \leq \frac{\left(\sum_{t=1}^{\infty} x(t)\right) P}{T\theta} - N \leq \frac{1}{2}. \tag{48}$$

Since each entry of N is an integer, equation 48 implies that N is the nearest-integer representation of the normalized accumulated input, i.e.,

$$N = \mathrm{round}\left(\frac{\left(\sum_{t=1}^{\infty} x(t)\right) P}{T\theta}\right). \tag{49}$$

Using the stimulus-normalized input

$$\bar{x} = \frac{1}{T}\sum_{t=1}^{\infty} x(t), \tag{50}$$

equation 49 can be rewritten as

$$N = \mathrm{round}\left(\frac{\bar{x}P}{\theta}\right). \tag{51}$$

We next consider the stimulus-normalized output. From equation 40,

$$\begin{aligned}
\bar{y} &= \frac{1}{T}\sum_{t=1}^{\infty} y(t) \\
&= \frac{1}{T}\sum_{t=1}^{\infty} \left(s(t) \odot T\theta\right) P^{-1} \\
&= \left[\left(\sum_{t=1}^{\infty} s(t)\right) \odot \theta\right] P^{-1} \\
&= \left[N \odot \theta\right] P^{-1}.
\end{aligned} \tag{52}$$

Substituting equation 51 into equation 52 gives

$$\boxed{\bar{y} = \left[\mathrm{round}\left(\frac{\bar{x}P}{\theta}\right) \odot \theta\right] P^{-1}}. \tag{53}$$

This is exactly the BA-MTN response to the same stimulus-normalized input $\bar{x}$. Therefore, BA-SIF provides an equivalent temporal realization of BA-MTN in the infinite-time limit.

Remark on threshold ties. The argument above assumes that no element of $\bar{x}P/\theta$ lies exactly on a half-integer quantization boundary. This is the generic case for continuous-valued activations. If an exact tie occurs, equivalence is preserved by using the same tie-breaking convention for $\mathrm{round}(\cdot)$ as that induced by the BA-SIF firing rule.

Remark on finite termination. Although Theorem 1 is stated using an infinite temporal summation, the corresponding BA-SIF realization terminates after a finite number of steps. After the first T stimulus steps, the network no longer receives external input or bias terms. Therefore, proceeding layer by layer from the input side, each BA-SIF neuron receives only a finite amount of residual incoming activity from the preceding layer. Since every emitted spike changes the membrane potential by a finite amount $T\theta$, the residual membrane potential can trigger only finitely many additional spikes before entering the resting interval

$$-\frac{T\theta}{2} \le V(t) \le \frac{T\theta}{2}. \tag{54}$$

Once this interval is reached and no further input arrives, the neuron remains inactive thereafter. By induction over the network depth, all BA-SIF neurons eventually become quiescent after a finite number of temporal steps. Hence, the infinite sums in Theorem 1 are in fact finite sums with only finitely many nonzero terms.

B.2 Proof of Corollary 1

We prove the result by induction over the layers of the network. The proof relies on two properties: the BA-SIF equivalence established in Theorem 1 and the telescoping property of the differential formulation used for nonlinear operators.

Consider first a nonlinear operator $f(\cdot)$. As defined in the main text, let

$$\bar{x}(t) = \frac{1}{m_t}\sum_{k=1}^{t} x(k), \qquad m_t = \min\{t, T\}, \tag{55}$$

and define its temporal output as

$$y(t) = m_t f(\bar{x}(t)) - m_{t-1} f(\bar{x}(t-1)), \tag{56}$$

where $m_0 = 0$. Summing equation 56 from $t = 1$ to a finite realization horizon $\mathcal{T} = T + T_R$ where T denotes the fixed stimulus steps length and T_R is an extendable relaxation steps horizon. We have

$$\begin{aligned}\sum_{t=1}^{\mathcal{T}} y(t) &= \sum_{t=1}^{\mathcal{T}} \left[m_t f(\bar{x}(t)) - m_{t-1} f(\bar{x}(t-1))\right] \\ &= m_{\mathcal{T}} f(\bar{x}(\mathcal{T})),\end{aligned} \tag{57}$$

since all intermediate terms cancel.

For $\mathcal{T} = T + T_R \ge T$, we have $m_{\mathcal{T}} = T$. Hence,

$$\frac{1}{T}\sum_{t=1}^{\mathcal{T}} y(t) = f(\bar{x}(\mathcal{T})). \tag{58}$$

Taking $\mathcal{T} \to \infty$ ($T_R \to \infty$, since T is fixed) and assuming the stimulus-normalized input converges,

$$\bar{x}(\mathcal{T}) \to \bar{x}, \tag{59}$$

continuity of f at $\bar{x}$ gives

$$\lim_{\mathcal{T}\to\infty} \frac{1}{T}\sum_{t=1}^{\mathcal{T}} y(t) = f(\bar{x}). \tag{60}$$

Thus, the differential temporal formulation exactly recovers the output of the corresponding nonlinear operator in the stimulus-normalized infinite-time limit.

We now proceed layer by layer. Suppose that the stimulus-normalized input to layer ℓ in the step-by-step SNN converges to the activation supplied to the corresponding layer of the BA-MTN network. For a BA-MTN activation replaced by BA-SIF, Theorem 1 guarantees that

$$\bar{y}_\ell^{\mathrm{SNN}} \to \left[\mathrm{round}\left(\frac{\bar{x}_\ell^{\mathrm{SNN}} P_\ell}{\theta_\ell}\right) \odot \theta_\ell\right] P_\ell^{-1}. \tag{61}$$

By the induction hypothesis, $\bar{x}_\ell^{\mathrm{SNN}}$ converges to the input of the corresponding BA-MTN layer. Therefore, the right-hand side of equation 61 is exactly the output of that BA-MTN layer.

Likewise, if the layer contains a nonlinear operator, the telescoping result in equation 60 shows that its stimulus-normalized temporal output converges to the nonlinear transformation of the corresponding normalized input. Linear transformations require no additional approximation, but the stimulus and relaxation phases need to be distinguished for the bias term. Consider an affine transformation

$$f(x) = xW + b. \tag{62}$$

The temporal realization of our stimulus-relaxation inference scheme is defined as

$$y(t) = \begin{cases} x(t)W + b, & 1 \le t \le T, \\ x(t)W, & T < t \le \mathcal{T}. \end{cases} \tag{63}$$

That is, the bias is injected only during the T stimulus steps and is disabled during relaxation, while residual temporal inputs are still propagated through the linear transformation.

Summing equation 63 over the entire realization horizon gives

$$\begin{aligned} \frac{1}{T}\sum_{t=1}^{\mathcal{T}} y(t) &= \frac{1}{T}\left(\sum_{t=1}^{\mathcal{T}} x(t)W + \sum_{t=1}^{T} b\right) \\ &= \left(\frac{1}{T}\sum_{t=1}^{\mathcal{T}} x(t)\right) W + b. \end{aligned} \tag{64}$$

Therefore, the stimulus-normalized temporal output is exactly the affine transformation of the stimulus-normalized input. In particular, setting the bias to zero during the relaxation steps is essential: it allows residual activity generated after the stimulus phase to propagate through the network without introducing additional bias beyond that of the corresponding BA-MTN layer.

Consequently, every operation in a layer preserves the stimulus-normalized equivalence between the step-by-step SNN and its BA-MTN counterpart. Since the input layer trivially satisfies this equivalence, induction over the network depth yields

$$\lim_{T_R \to \infty} \bar{x}_\ell^{\mathrm{SNN}}(T + T_R) = x_\ell^{\text{BA-MTN}} \qquad \text{for every layer } \ell, \tag{65}$$

where $x_\ell^{\text{BA-MTN}}$ denotes the corresponding BA-MTN layer output.

Therefore, replacing each BA-MTN neuron with its BA-SIF temporal realization and implementing nonlinear operators using the differential formulation preserves the learned discrete representation throughout the network in the infinite-time limit.

C Calibration and Inference Details

C.1 Layer-Wise Calibration

We calibrate the model in a layer-wise manner. For each Transformer layer, BA-MTN modules are inserted according to Fig. 2, while all previously calibrated layers are kept fixed. The calibration of each layer consists of two stages.

In the first stage, which lasts for E_1 epochs, we update the channel-wise threshold θ from the activation distribution while optimizing only the basis transformation P. Specifically, at the beginning of each epoch, we perform one forward pass over the calibration set and collect the activation statistics of the current layer. A threshold estimate $\hat{\theta}$ is obtained from the collected activation distribution and incorporated into θ through an exponential moving average (EMA). The resulting θ is then fixed for the remainder of the epoch, and only P is optimized.

In the second stage, which lasts for E_2 epochs, we jointly refine the basis transformation and the threshold. To guarantee positivity of the threshold during optimization, we reparameterize the final threshold from the first stage as

$$\theta = \exp(\alpha), \qquad \alpha \leftarrow \log \theta, \tag{66}$$

Table 5: Calibration hyperparameters used across all models and model scales.

Setting	**Stage I**	**Stage II**
τ	1.0	1.0
λ	1.0	1.0
Epochs	15	10
Optimizer	AdamW	AdamW
Batch size	4	4
Initial learning rate	5×10^{-3}	5×10^{-5}
Learning-rate schedule	Cosine	Exponential
Final learning rate	5×10^{-6}	N/A
decay factor	N/A	0.98
Optimized parameters	P	P, α
EMA decay coefficient β	0.8	N/A

and jointly optimize P and α.

Both stages optimize the reconstruction and activity objectives defined in the main text,

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda \mathcal{L}_{\text{act}}. \tag{67}$$

Because these two objectives may produce conflicting optimization directions, we employ Projecting Conflicting Gradients (PC-Grad) (Yu et al., 2020) during calibration. Let

$$g_{\text{rec}} = \nabla_\phi \mathcal{L}_{\text{rec}}, \qquad g_{\text{act}} = \nabla_\phi \left(\lambda \mathcal{L}_{\text{act}}\right), \tag{68}$$

where $\phi = P$ in the first stage and $\phi = \{P, \alpha\}$ in the second stage. When two gradients conflict, i.e.,

$$g_i^\top g_j < 0, \tag{69}$$

PC-Grad projects g_i onto the normal plane of g_j as

$$g_i \leftarrow g_i - \frac{g_i^\top g_j}{\|g_j\|_2^2} g_j. \tag{70}$$

The projected gradients are then combined for parameter updates. This gradient surgery mitigates destructive interference between reconstruction fidelity and activity reduction during calibration.

After all layers have been calibrated, we reparameterize the model by folding the learned basis transformations into the corresponding weights according to equation 3. For experiments requiring low-bit weights, we subsequently quantize the reparameterized weights using round-to-nearest (RTN) Nagel et al. (2020).

The complete calibration procedure is summarized in Algorithm 1.

C.2 Calibration Hyperparameters

We use a unified calibration configuration across all model families and model scales. The activity-budget and loss-weight hyperparameters are fixed to

$$\tau = 1.0, \qquad \lambda = 1.0 \tag{71}$$

throughout all experiments. We sample 128 calibration sequences from WikiText-2, each containing 2048 tokens.

For the first calibration stage, we use $E_1 = 15$ epochs with AdamW and a batch size of 4. The learning rate is initialized to 5×10^{-3} and cosine-decayed to 5×10^{-6}. For the second stage, we use $E_2 = 10$ epochs with the same optimizer and batch size. The learning rate starts from 5×10^{-5} and follows an exponential decay schedule with a multiplicative decay factor of 0.98.

The calibration settings are summarized in Table 5.

Algorithm 1 Layer-Wise Calibration of ACT-SNN

Require: Pretrained model $\mathcal{M}$, calibration set $\mathcal{D}$, E_1, E_2, τ, λ, β
Ensure: Calibrated and reparameterized model $\mathcal{M}$

for each Transformer layer $\ell = 1, \ldots, L$ **do**
  Insert BA-MTN modules according to Fig. 2
  Initialize P_ℓ and θ_ℓ
  for $e = 1, \ldots, E_1$ **do**
    Run one forward pass on $\mathcal{D}$ and collect activation statistics
    Estimate $\hat{\theta}_\ell^{(e)}$ from the activation distribution
    Update $\theta_\ell \leftarrow \beta\theta_\ell + (1-\beta)\hat{\theta}_\ell^{(e)}$
    Freeze θ_ℓ
    for each mini-batch $\mathcal{B} \subset \mathcal{D}$ **do**
      Compute $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{act}}$
      Compute PC-Grad using $\mathcal{L}_{\text{rec}}$ and $\lambda\mathcal{L}_{\text{act}}$
      Update P_ℓ
    end for
  end for
  $\alpha_\ell \leftarrow \log\theta_\ell$
  Reparameterize $\theta_\ell = \exp(\alpha_\ell)$
  for $e = 1, \ldots, E_2$ **do**
    for each mini-batch $\mathcal{B} \subset \mathcal{D}$ **do**
      Compute $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{act}}$
      Compute PC-Grad using $\mathcal{L}_{\text{rec}}$ and $\lambda\mathcal{L}_{\text{act}}$
      Jointly update P_ℓ and α_ℓ
    end for
  end for
  Freeze the calibrated layer
end for
Reparameterize all learned basis transformations according to equation 3
if low-bit weight quantization is enabled **then**
  Quantize the reparameterized weights using RTN
end if
return $\mathcal{M}$

C.3 Conversion from BA-MTN to BA-SIF

After calibration, we convert the BA-MTN network into its step-by-step BA-SIF counterpart. Each BA-MTN is first replaced by the corresponding BA-SIF neuron while directly inheriting its calibrated parameters. The remaining nonlinear operators are then replaced by their differential temporal realizations.

We consider two classes of operators. The unary operators include RMSNorm, SiLU, and SoftMax, while the binary operators include matrix multiplication (MatMul) and the Hadamard product. Their differential forms are constructed such that temporal accumulation recovers the corresponding operation on the stimulus-normalized representation.

Unary operators. For a unary operator $f(\cdot)$, define the accumulated input as

$$S_x(t) = \sum_{k=1}^{t} x(k), \tag{72}$$

and

$$m_t = \min\{t, T\}, \qquad \bar{x}(t) = \frac{S_x(t)}{m_t}. \tag{73}$$

The temporal differential output is

$$y(t) = m_t f(\bar{x}(t)) - m_{t-1} f(\bar{x}(t-1)), \tag{74}$$

where the second term is defined as zero for $t = 1$. By telescoping over time,

$$\frac{1}{m_t}\sum_{k=1}^{t} y(k) = f(\bar{x}(t))\,, \tag{75}$$

which guarantees consistency with the corresponding unary operator under stimulus normalization.

Binary operators. For a binary operator, let $\#$ denote either matrix multiplication or the Hadamard product. Given two temporal inputs $x_1(t)$ and $x_2(t)$, we define

$$S_{x_1}(t) = \sum_{k=1}^{t} x_1(k), \qquad S_{x_2}(t) = \sum_{k=1}^{t} x_2(k), \tag{76}$$

and

$$M(t) = S_{x_1}(t) \,\#\, S_{x_2}(t). \tag{77}$$

Since both MatMul and the Hadamard product are bilinear, $M(t)$ can be updated incrementally as

$$\begin{aligned} M(t+1) = M(t) &+ S_{x_1}(t) \,\#\, x_2(t+1) \\ &+ x_1(t+1) \,\#\, S_{x_2}(t) + x_1(t+1) \,\#\, x_2(t+1). \end{aligned} \tag{78}$$

Define

$$Q(t) = \frac{M(t)}{m_t}, \qquad Q(0) = 0. \tag{79}$$

The temporal output is then given by

$$y(t) = Q(t) - Q(t-1), \tag{80}$$

or equivalently, for $t > 1$,

$$y(t) = \frac{M(t)}{m_t} - \frac{M(t-1)}{m_{t-1}}. \tag{81}$$

The temporal sum therefore satisfies

$$\sum_{k=1}^{t} y(k) = \frac{S_{x_1}(t) \,\#\, S_{x_2}(t)}{m_t}. \tag{82}$$

Dividing by the stimulus normalization factor m_t gives

$$\frac{1}{m_t}\sum_{k=1}^{t} y(k) = \left(\frac{S_{x_1}(t)}{m_t}\right) \# \left(\frac{S_{x_2}(t)}{m_t}\right), \tag{83}$$

showing that the differential module exactly realizes the binary operator on the corresponding stimulus-normalized inputs.

C.4 Stimulus–Relaxation Inference

We simulate step-by-step SNN inference on GPUs using a stimulus–relaxation schedule. Let T denote the fixed number of stimulus steps and T_R denote the extendable number of relaxation steps, giving a total realization horizon

$$\mathcal{T} = T + T_R. \tag{84}$$

For an input embedding e, the external input supplied to the SNN is

$$e(t) = \begin{cases} e, & 1 \le t \le T, \\ 0, & T < t \le T + T_R. \end{cases} \tag{85}$$

Thus, the same input is replicated for the first T stimulus steps and propagated through the model step by step. After the stimulus phase, external input is removed and a zero embedding (`zero_embed`) is used to advance the network during the relaxation phase, allowing residual spike activity to propagate through subsequent layers.

Bias terms are injected only during the stimulus phase. For an affine transformation $xW + b$, its temporal realization is therefore

$$y(t) = \begin{cases} x(t)W + b, & 1 \le t \le T, \\ x(t)W, & T < t \le T + T_R. \end{cases} \tag{86}$$

This ensures that the bias contributes exactly T times before stimulus normalization, while no additional bias is introduced during relaxation.

The differential nonlinear modules follow the same principle. During the relaxation phase, $m_{t \ge T} = T$. Hence, if a unary module receives a zero temporal input, its accumulated state remains unchanged and equation 74 produces exactly zero output. Likewise, a binary differential module produces zero output whenever both temporal inputs are zero. We exploit this property to skip redundant nonlinear computation during inactive relaxation steps.

After all $T + T_R$ temporal steps are completed, we collect the temporal outputs $\{y(t)\}_{t=1}^{T+T_R}$ and compute the final stimulus-normalized output as

$$\bar{y} = \frac{1}{T} \sum_{t=1}^{T+T_R} y(t). \tag{87}$$

Unless otherwise specified, all standard experiments use

$$T = 8, \qquad T_R = 8. \tag{88}$$

Remark on neuromorphic inference latency. The GPU implementation above explicitly simulates all temporal steps and is used for evaluating numerical accuracy. On a neuromorphic substrate, the step-by-step formulation permits temporal pipelining across layers. Suppose the model contains L Transformer blocks, and each block contains five causally ordered BA-SIF stages, as illustrated in Fig. 2. Under a fully pipelined execution model, the pipeline depth is approximately $5L$, while the temporal stream contains $T + T_R$ steps. The resulting inference latency therefore scales as

$$\mathcal{O}\left(T + T_R + 5L\right). \tag{89}$$

In contrast, a layer-by-layer execution schedule that completes all T simulation steps of one BA-SIF stage before advancing to the next requires a latency of

$$\mathcal{O}(5LT). \tag{90}$$

Thus, step-by-step execution removes the multiplicative dependence between network depth and simulation length, which is the principal latency advantage of the proposed temporal realization on neuromorphic hardware.

D Energy Estimation and Operation Statistics

This section provides the detailed arithmetic-energy model and operation-counting rules used in the main text. Following prior SNN energy analyses (Zheng et al., 2026), we consider only the arithmetic cost of multiply–accumulate (MAC) and accumulate (AC) operations. Memory access, data movement, normalization, nonlinear functions, and control logic are not included. Therefore, the reported energy ratios should be interpreted as arithmetic-energy estimates rather than end-to-end hardware power measurements.

D.1 Arithmetic Energy Model

We adopt the 45-nm CMOS arithmetic-energy measurements summarized by Neseem et al. (2024). Their measurements report an FP16 multiplication cost of $1.10\,\text{pJ}$ and an FP16 addition cost of $0.40\,\text{pJ}$. Since one MAC consists of one multiplication followed by one accumulation, we use

$$E_{\text{MAC}}^{\text{FP16}} = 1.10 + 0.40 = 1.50\ \text{pJ}, \qquad E_{\text{AC}}^{\text{FP16}} = 0.40\ \text{pJ}. \tag{91}$$

For low-precision integer arithmetic, Neseem et al. (2024) directly report $0.20\,\text{pJ}$ and $0.03\,\text{pJ}$ for an INT8 multiplication and addition, respectively, while the corresponding INT4 energy values are

not directly reported. We extrapolate the INT4 costs following the conventional bit-width scaling of digital arithmetic, where multiplication cost scales approximately quadratically and addition cost approximately linearly with operand precision. This gives

$$E_{\mathrm{MUL}}^{\mathrm{INT4}} = 0.20 \left(\frac{4}{8}\right)^2 = 0.050 \text{ pJ}, \tag{92}$$

and

$$E_{\mathrm{ADD}}^{\mathrm{INT4}} = 0.03 \left(\frac{4}{8}\right) = 0.015 \text{ pJ}. \tag{93}$$

Accordingly, we use

$$E_{\mathrm{MAC}}^{\mathrm{INT4}} = 0.050 + 0.015 = 0.065 \text{ pJ}, \qquad E_{\mathrm{AC}}^{\mathrm{INT4}} = 0.015 \text{ pJ}. \tag{94}$$

For a model containing N_{MAC} MACs and N_{AC} ACs, its arithmetic energy is estimated as

$$E_{\mathrm{comp}} = N_{\mathrm{MAC}} E_{\mathrm{MAC}} + N_{\mathrm{AC}} E_{\mathrm{AC}}. \tag{95}$$

The energy ratio (ER) is normalized by the corresponding FP16 ANN baseline,

$$\mathrm{ER} = \frac{E_{\mathrm{comp}}}{E_{\text{FP16-ANN}}}. \tag{96}$$

D.2 Operation Counting for ANN and Quantized ANN

For ANNs and QANNs, we count MACs from standard linear layers, including the query, key, value, and output projections in self-attention, the gate, up, and down projections in the feed-forward network, and the final language-modeling head. Other operations are excluded from the arithmetic accounting defined above. Since dense linear layers do not exploit spike sparsity, no AC operations are counted separately, and their energy is

$$E_{\mathrm{ANN/QANN}} = N_{\mathrm{MAC}}^{\mathrm{ANN/QANN}} E_{\mathrm{MAC}}^{\mathrm{FP16/INT4}}. \tag{97}$$

Consider a model with hidden dimension h, intermediate dimension h_i, L Transformer blocks, vocabulary size V, and sequence length S. For LLaMA-2, which uses standard multi-head attention, the number of MACs is

$$N_{\mathrm{MAC}}^{\mathrm{LLaMA2}} = S\left[L\left(4h^2 + 3hh_i\right) + hV\right]. \tag{98}$$

For LLaMA-2-7B, using $h = 4096$, $h_i = 11008$, $L = 32$, and $V = 32000$, this becomes

$$N_{\mathrm{MAC}}^{\text{LLaMA2-7B}} = 6.607077376 \times 10^9 S. \tag{99}$$

For an illustrative sequence length of $S = 2048$,

$$N_{\mathrm{MAC}}^{\text{LLaMA2-7B}} = 1.3531294466048 \times 10^{13} \approx 13.531 \text{ TMAC}. \tag{100}$$

The corresponding arithmetic energy is

$$\begin{aligned} E_{\text{FP16-ANN}}^{\text{LLaMA2-7B}} &= 13.531 \times 10^{12} \times 1.50 \times 10^{-12} \\ &\approx 20.297 \text{ J}, \end{aligned} \tag{101}$$

and

$$\begin{aligned} E_{\text{INT4-QANN}}^{\text{LLaMA2-7B}} &= 13.531 \times 10^{12} \times 0.065 \times 10^{-12} \\ &\approx 0.880 \text{ J}. \end{aligned} \tag{102}$$

D.3 Operation Counting for SNNs

For SNNs, we separately count MAC and AC operations. Dense or non-spike-driven computations that still require multiplication are counted as MACs, whereas synaptic propagation driven by unit-magnitude spikes is counted as ACs. For a signed unit spike $s \in \{-1, 0, +1\}$, multiplication by the spike value can be replaced by an addition, subtraction, or no operation, and therefore does not require a full MAC.

For FP16-weight SNNs, the arithmetic-energy estimate is

$$E_{\mathrm{SNN}}^{\mathrm{FP16}} = 1.50N_{\mathrm{MAC}} + 0.40N_{\mathrm{AC}} \quad \mathrm{pJ}, \tag{103}$$

whereas for the 4-bit setting we use

$$E_{\mathrm{SNN}}^{\mathrm{INT4}} = 0.065N_{\mathrm{MAC}} + 0.015N_{\mathrm{AC}} \quad \mathrm{pJ}. \tag{104}$$

The AC count is directly related to spike activity. Let $N_{\mathrm{MAC},\ell}^{\mathrm{dense}}$ denote the dense-equivalent MAC count of a spike-driven linear layer ℓ, and let Act_ℓ denote the average number of nonzero unit-spike events per input activation over the complete temporal realization. Each nonzero input spike activates the corresponding synaptic accumulations, giving

$$N_{\mathrm{AC},\ell} = N_{\mathrm{MAC},\ell}^{\mathrm{dense}}\,\mathrm{Act}_\ell. \tag{105}$$

Summing over all spike-driven layers,

$$N_{\mathrm{AC}} = \sum_\ell N_{\mathrm{MAC},\ell}^{\mathrm{dense}}\,\mathrm{Act}_\ell. \tag{106}$$

For reporting a single network-level activity value, we use the dense-operation-weighted activity

$$\mathrm{Activity} = \frac{\sum_\ell N_{\mathrm{MAC},\ell}^{\mathrm{dense}}\mathrm{Act}_\ell}{\sum_\ell N_{\mathrm{MAC},\ell}^{\mathrm{dense}}}. \tag{107}$$

Defining

$$N_{\mathrm{MAC}}^{\mathrm{dense}} = \sum_\ell N_{\mathrm{MAC},\ell}^{\mathrm{dense}}, \tag{108}$$

we obtain the compact relation

$$\boxed{N_{\mathrm{AC}} = N_{\mathrm{MAC}}^{\mathrm{dense}} \times \mathrm{Activity}}. \tag{109}$$

Therefore, reducing spike activity directly reduces the dominant AC cost of spike-driven linear transformations. Remaining MAC operations are counted separately according to the actual implementation of each method.

D.4 Operation Statistics

Tables 6 and 7 summarize the MAC/AC statistics used for energy estimation. ER is computed using equation 96 and is always normalized to the FP16 ANN baseline of the corresponding model.

Table 6: Arithmetic operation statistics and energy ratios of the FP16-weight models. ER is normalized to the corresponding FP16 ANN baseline.

Model	Method	TMACs	TACs	Activity	ER
LLaMA-2-7B	FP16 Baseline	13.53	0.00	–	1.000
	Ours	2.25	20.43	1.51	0.569
LLaMA-2-13B	FP16 Baseline	26.32	0.00	–	1.000
	Ours	3.96	40.01	1.52	0.556
LLaMA-2-70B	FP16 Baseline	140.72	0.00	–	1.000
	Ours	15.67	195.60	1.39	0.482
LLaMA-3-8B	FP16 Baseline	15.37	0.00	–	1.000
	Ours	2.88	23.82	1.55	0.601
Qwen2.5-7B	FP16 Baseline	14.48	0.00	–	1.000
	Ours	2.78	20.56	1.42	0.571
Qwen2.5-32B	FP16 Baseline	65.50	0.00	–	1.000
	Ours	7.03	87.11	1.33	0.462

Table 7: Arithmetic operation statistics and energy ratios of the INT4-weight models. The FP16 ANN is retained as the reference baseline for ER.

Model	Method	TMACs	TACs	Activity	ER
LLaMA-2-7B	FP16 Baseline	13.53	0.00	–	1.000
	Ours (INT4)	2.03	20.97	1.55	0.022
LLaMA-2-13B	FP16 Baseline	26.32	0.00	–	1.000
	Ours (INT4)	3.28	38.43	1.46	0.020
LLaMA-2-70B	FP16 Baseline	140.72	0.00	–	1.000
	Ours (INT4)	11.04	177.31	1.26	0.016
LLaMA-3-8B	FP16 Baseline	15.37	0.00	–	1.000
	Ours (INT4)	2.30	22.29	1.45	0.021
Qwen2.5-7B	FP16 Baseline	14.48	0.00	–	1.000
	Ours (INT4)	1.94	20.56	1.42	0.020
Qwen2.5-32B	FP16 Baseline	65.50	0.00	–	1.000
	Ours (INT4)	5.59	80.56	1.23	0.016