SKEWSPIKE: A DICTIONARY-BASED SKEWED-TREE FRAMEWORK FOR CO-FIRING SPIKE COMPRESSION IN SPIKING NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking Neural Networks (SNNs) offer a promising energy-efficient alternative to conventional neural networks due to their event-driven execution. However, modern deep SNNs predominantly utilize rate coding, generating excessive spike volumes that saturate Address Event Representation (AER) communication channels on neuromorphic hardware. In this work, we introduce SkewSpike, a hierarchical dictionary-based AER spike compression framework designed to alleviate this communication bottleneck by exploiting skewed neuronal co-firing activity. SkewSpike maps recurring synchronous spike patterns into compact symbolic AER packets via a right-skewed tree dictionary, while further sparsifying residual communication traffic through stochastic spike pruning. To prevent significant accuracy degradation caused by residual spike pruning, the framework incorporates an adaptive threshold calibration mechanism that preserves rare features within the activity distribution, alongside a deterministic, model-agnostic autotuner that discovers optimal configurations for target AER compression ratios. We evaluate SkewSpike across diverse spiking architectures and datasets, including event-based neuromorphic benchmarks and full-scale ImageNet-1k. Experimental results demonstrate that SkewSpike substantially outperforms standard Huffman coding and existing AER compression baselines, achieving up to 75.41% AER event volume reduction on fully connected models with negligible accuracy loss, and maintaining 50.38% AER compression on ImageNet-1k with only a 2.05% accuracy drop.

1 INTRODUCTION

The rapid growth of artificial intelligence over recent years has led to substantial energy demands, necessitating the development of energy-efficient processing for resource-constrained edge devices. Spiking Neural Networks (SNNs), often categorized as the third generation of neural networks, offer a promising solution by emulating the sparse and event-driven dynamics of biological systems (Abdallah & Dang, 2022; Yan et al., 2026). Unlike conventional artificial neural networks (ANNs) that rely on continuous-valued activations, SNNs process and transmit information using discrete spatiotemporal events known as spikes (Maass, 1996). This shift toward event-driven computation enables low-power asynchronous parallelism on neuromorphic hardware. However, implementing this paradigm requires converting input data into spike trains through a process called spike encoding, which typically utilizes rate or temporal coding (Eshraghian et al., 2023).

Rate coding is widely used in SNNs due to its implementation simplicity and noise robustness (Guo et al., 2021). However, its deployment in deep SNNs on resource-constrained hardware can introduce communication bottlenecks (Plank et al., 2025), particularly during inter-chip transfers (Kim et al., 2023; Qiao & Indiveri, 2018). To maintain accuracy on complex tasks, rate-coded models frequently generate high spike volumes, creating overhead across Address Event Representation (AER) protocols that can limit edge scalability (Linares-Barranco et al., 2006). Although SNNs are fundamentally sparse, activity is typically non-uniform across the network (Roxin et al., 2011); firing rates often follow a skewed distribution where a small subset of active neurons accounts for a disproportionate share of activity while the majority fire infrequently (Mizuseki & Buzsáki, 2013).

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.

Recent research on SNN efficiency has explored several directions, including pruning, quantization, temporal reduction, and spike sparsity. Pruning methods remove redundant synaptic connections to reduce computational and memory costs (Chen et al., 2021; Shi et al., 2024), while quantization methods reduce the numerical precision of weights and neuron states to improve storage and computational efficiency (Lui & Neftci, 2021; Li et al., 2022). Temporal reduction methods seek to alleviate the temporal redundancy of SNNs by reducing the number of simulation timesteps required for inference (Meng et al., 2022; Ding et al., 2024), while spike-sparsity approaches directly reduce neural activity to lower computational and energy costs (Bu et al., 2025). However, these approaches primarily reduce the number of timesteps, generated spike activity, or model parameters, rather than exploiting redundancy within the AER representation of the spike events that are generated. This leaves an important question: whether the non-uniform and correlated firing patterns inherent in rate-coded SNNs can instead be exploited to compress AER spike data while preserving the information necessary for task performance.

Recent work has begun to directly address redundancy in AER spike data. Work in (Ganesh et al., 2025) exploits correlated and frequently occurring firing patterns to compress spike data without modifying the underlying SNN architecture. However, its evaluation is limited to relatively small SNNs. Whether such AER spike-pattern compression can scale effectively to substantially larger and deeper SNNs therefore remains an open question.

To address this scalability challenge, we present SkewSpike, a hierarchical, dictionary-based lossy compression framework that exploits skewed neural activity to efficiently reduce spike data. The framework is specifically designed for rate-coded SNN architectures, where increasing spike-event traffic can stress the finite throughput of AER communication links as networks scale to larger and deeper architectures, potentially making inter-chip communication a bottleneck (Kim et al., 2023; Qiao & Indiveri, 2018).

SkewSpike overcomes this bottleneck by exploiting non-uniform firing in trained SNNs, where a core subset of neurons co-fires frequently while the remainder fires sparsely to encode fine-grained features. Offline, SkewSpike identifies these frequent co-firing patterns into a hierarchical, right-skewed tree dictionary of compact symbols, while calibrating the firing sensitivity of rare neurons to protect residual information. During online inference, co-firing patterns matching the dictionary are compressed into single symbolic AER packets, while uncovered residual spikes are stochastically retained via biologically inspired transmission dynamics (Kappel & Tetzlaff, 2024; Malkin et al., 2024). At the receiving layer, these symbols are efficiently expanded back into constituent neuron spikes, substantially reducing inter-layer communication traffic while preserving model accuracy. Our contributions are as follows:

- We propose SkewSpike, a compression framework that identifies recurring co-firing patterns by constructing a right-skewed tree dictionary via the k-means algorithm. To enhance data reduction, we incorporate a stochastic spike pruning mechanism that further sparsifies activity by removing non-essential residual spikes.
- We introduce a spike threshold calibration mechanism that mitigates accuracy loss by amplifying the firing sensitivity of rare but salient features. This is paired with a deterministic, model- and dataset-agnostic autotuning algorithm that automatically discovers optimal parameter configurations for target compression ratios.
- Through extensive empirical validation, we demonstrate that SkewSpike significantly reduces spike data volume while maintaining accuracy levels close to uncompressed baselines. Our evaluations across diverse datasets and architectures confirm the framework's capability to achieve high compression ratios in deep spiking models.

2 Background

2.1 Related Works

Parameter Compression via Pruning and Quantization Early SNN compression adapted techniques from traditional artificial neural networks to reduce parameter storage and synaptic operations. Weight pruning induces structural or unstructured sparsity through gradient rewiring methods (Chen et al., 2021), unstructured pruning frameworks (Shi et al., 2024), or unsupervised plastic-

ity rules (Guo et al., 2020). Similarly, quantization reduces the numerical precision of weights and membrane potentials through quantization-aware training (Lui & Neftci, 2021) or post-training schemes (Li et al., 2022). While these parameter-centric methods lower model complexity and static memory usage, they do not resolve inference communication bottlenecks. Even when synaptic weights are substantially pruned, rate-coded models can still generate considerable spike activity (Bu et al., 2025), potentially leading to notable packet traffic across neuromorphic interconnects (Plank et al., 2025).

Temporal Latency Reduction and Activity Regularization To lower dynamic compute costs, subsequent research focused on temporal and activation redundancy. Because standard SNNs often require extended simulation windows, latency-reduction techniques train models using few timesteps via direct differentiation on spike representations (Meng et al., 2022) or temporal scaling (Ding et al., 2024). Complementary activity-pruning methods penalize high firing rates to suppress total spike counts (Bu et al., 2025). However, both approaches treat spikes as isolated scalar events. While they reduce overall event counts, they do not exploit spatial co-activation structures or non-uniform population distributions.

AER Communication Bottlenecks and Spike Pattern Compression In neuromorphic hardware, inter-layer spike transmission via Address Event Representation (AER) protocols represents a major communication bottleneck (Linares-Barranco et al., 2006; Plank et al., 2025), yet direct event-stream compression remains largely unexplored. Frequent Spike Pattern Compression (FSPC) (Ganesh et al., 2025) recently merged recurring co-firing patterns into composite symbols. However, FSPC relies on combinatorial pattern mining (such as FP-Max), which exhibits exponential complexity with layer width, restricting its evaluation to shallow networks with at most 100 hidden neurons. Moreover, existing schemes lack mechanisms to prevent accuracy degradation when residual spikes are pruned, and they require manual grid searches to find viable operating parameters.

2.2 Spiking Neuron Model

We adopt the Leaky Integrate-and-Fire (LIF) model (Lapique, 1907), for its balance between biological plausibility and computational efficiency in deep SNNs. At each timestep t, the input current $I_i[t]$ to postsynaptic neuron i is given by:

$$I_i[t] = \sum_j w_{ji} S_j[t] \tag{1}$$

where w_{ji} is the synaptic weight and $S_j[t]$ is the presynaptic spike. The membrane potential $U_i[t]$ updates with decay factor $\tau \in (0, 1)$ as:

$$U_i[t] = \tau U_i[t-1] + I_i[t] \tag{2}$$

When $U_i[t]$ reaches threshold V_{th}, the neuron emits a spike and undergoes a hard reset to V_{reset}:

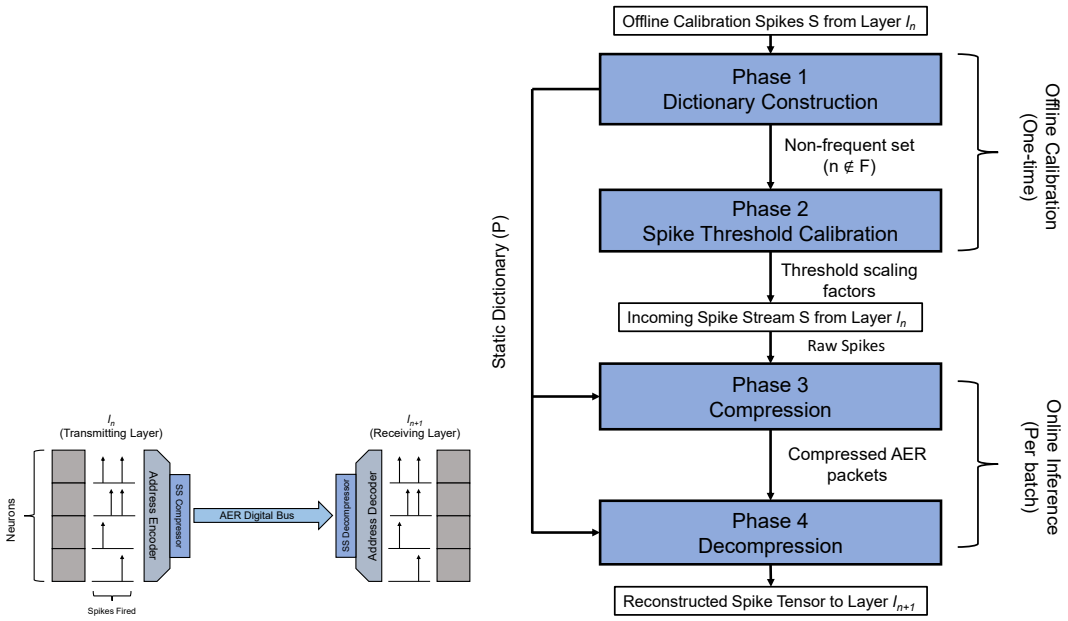
$$S_i[t] = \begin{cases} 1, & \text{if } U_i[t] \geq V_{\text{th}} \\ 0, & \text{otherwise} \end{cases}, \quad U_i[t] \leftarrow V_{\text{reset}} \text{ if } S_i[t] = 1 \tag{3}$$

2.3 Address Event Representation

Address Event Representation (AER) is a neuromorphic communication protocol designed to encode the timing and spatial location of spike events (Sivilotti, 1991; Mahowald, 1992). Rather than transmitting a dense binary spike matrix, AER conveys only the spatiotemporal coordinates of active spikes. Let $\boldsymbol{S} \in \{0, 1\}^{T \times N}$ denote a binary spike tensor, where T is the number of timesteps and N is the total number of neurons. An entry $S_{t,n} = 1$ indicates that neuron n emits a spike at time t, and 0 otherwise. AER represents only the indices where these events occur:

$$A = \{(t, n) \mid S_{t,n} = 1\}. \tag{4}$$

Each tuple $(t, n) \in A$ denotes a spike event for neuron n at timestep t. AER is utilized by leading neuromorphic hardware, including IBM's TrueNorth (Andreou et al., 2016; DeBole et al., 2019) and Intel's Loihi (Davies et al., 2018). In these architectures, each event in A is encapsulated as a digital packet and routed across an asynchronous Network-on-Chip (NoC).



(a) Inter-layer AER hardware deployment (b) Four-phase SkewSpike operational pipeline

Figure 1: Overview of SkewSpike. (a) Inter-layer hardware deployment: compressing spike events across the AER bus between layers l_n and l_{n+1}. (b) Operational pipeline: offline dictionary construction (m) and threshold calibration (ω), followed by online symbolic compression (p, σ) and spike reconstruction.

3 METHOD

In this section, we detail the SkewSpike framework. SkewSpike consists of four phases: two offline calibration phases (dictionary construction and threshold calibration) and two online inference-time phases (compression and decompression), inserted between layers l_n and l_{n+1}. We also introduce a deterministic autotuning algorithm to automate parameter configuration. Compression and decompression are applied per batch over the full dataset.

3.1 PHASE 1: DICTIONARY CONSTRUCTION

This is a one-time offline calibration step that extracts recurring co-activation patterns from layer l_n across the dataset. Taking the uncompressed spike tensor $\mathbf{S} \in \{0,1\}^{T\times B\times N}$ over $M = T \times B$ total transactions as input, we first isolate the core set of consistently active neurons. A neuron is designated as frequent if its firing support meets the Minimum Spike Rate (MSR) threshold $m \in [0,1]$:

$$F = \left\{ n \in \{1,\dots,N\} \;\middle|\; \frac{1}{M}\sum_{t=1}^{T}\sum_{b=1}^{B} S_{t,b,n} \geq m \right\} \tag{5}$$

The firing activity of F across transactions is gathered into a matrix $\mathbf{X} \in \{0,1\}^{|F|\times M}$ and recursively clustered into a hierarchical, right-skewed tree via bisected k-means ($k = 2$). At each partitioning step, each candidate neuron cluster c is scored by its combined spike volume V_c and temporal cohesion H_c:

$$W_c = V_c \cdot H_c, \quad V_c = \sum_{r=1}^{M}\sum_{n\in c} X_{n,r}, \quad H_c = \frac{1}{|c|}\,\text{mean}\left(\sum_{n\in c} X_{n,r} \;\middle|\; \text{active } r \right) \tag{6}$$

To enforce the skewed tree structure, the cluster with the higher score W_c (the heavy branch) is immediately finalized as a leaf symbol node, while recursion continues exclusively down the lighter branch up to depth $L = \max(1, \lfloor \log_2(|F|/\mu_{\text{sync}}) \rfloor)$, where μ_{sync} is the average number of co-firing neurons per active transaction. This recursive process outputs a binary dictionary matrix

$\mathbf{P} \in \{0,1\}^{N \times |V|}$, where parent nodes preserve the union of their children ($P_{\text{parent}} = P_{\text{left}} \cup P_{\text{right}}$) to facilitate multi-level symbol substitution during compression.

3.2 Phase 2: Spike Threshold Calibration

This offline calibration phase offsets information loss caused by Phase 3, where uncompressed residual spikes are stochastically discarded to save bandwidth. Because infrequent neurons ($n \notin F$) fire sparsely, random dropout risks extinguishing their fine-grained features entirely. To protect these signals, we lower their effective threshold to $\omega < V_{\text{th}}$ by scaling incoming currents I_n by a factor ϕ_n:

$$\phi_n = \begin{cases} V_{\text{th}} & \text{if } n \in F \\ \frac{V_{\text{th}}}{\omega} & \text{if } n \notin F \end{cases}, \quad I'_n = \phi_n \cdot I_n \tag{7}$$

Since $\phi_n > 1$ for rare neurons, this scaling elevates their spike counts during active periods. Consequently, enough residual spikes survive Phase 3 stochastic pruning to preserve feature fidelity and accuracy at layer l_{n+1}.

3.3 Phase 3: Compression

During online inference, Phase 3 compresses incoming spike trains by matching co-firing neuron activity against the calibrated dictionary. Taking incoming spikes represented in Address-Event Representation (AER) format, $E = \{(t,b,n) \mid S_{t,b,n} = 1\}$, candidate dictionary patterns are triggered at transaction $r = (t-1)B + b$ if their co-active spike count meets the Pattern Matching Threshold (PMT) ratio $p \in [0,1]$:

$$\Gamma_{r,v} = \mathbb{I}\left[|E_r \cap P_v| \geq \lceil |P_v| \cdot p \rceil\right] \tag{8}$$

where $E_r = \{n \mid (t,b,n) \in E\}$ is the active spike set at transaction r, and $|P_v|$ is the size of pattern v. To maximize compression and avoid redundancy, candidate triggers are resolved across the tree hierarchy: whenever a parent node and its constituent children are co-triggered, the parent symbol Γ^*_{r,v_p} is activated, and its child activations are suppressed.

Spikes not covered by any finalized active pattern form the residual set $E_{\text{leftover}} = \{(t,b,n) \in E \mid \sum_{v=1}^{|V|} \Gamma^*_{r,v} P_{n,v} = 0\}$. Inspired by stochastic biological synaptic transmission (Kappel & Tetzlaff, 2024; Malkin et al., 2024), these leftover spikes undergo probabilistic survival with probability $\sigma \in [0,1]$. The resulting compressed payload E_{comp} is constructed by aggregating the active dictionary symbols with the surviving residual spikes:

$$E_{\text{comp}} = \left\{(t,b,s_v) \mid \Gamma^*_{r,v} = 1\right\} \cup \left\{e \in E_{\text{leftover}} \mid \text{Bernoulli}(\sigma) = 1\right\} \tag{9}$$

where s_v is the unique symbol ID representing pattern v.

3.4 Phase 4: Decompression

During online inference, this phase reconstructs the full binary spike tensor $\hat{\mathbf{S}} \in \{0,1\}^{T \times B \times N}$ from the compressed AER payload E_{comp} at the receiving layer l_{n+1}. The incoming payload contains two types of events: raw residual spikes and composite dictionary symbols. Reconstruction is performed in parallel by mapping residual spikes directly to their coordinates while expanding symbolic events across their constituent neuron patterns:

$$\hat{S}_{t,b,n} = 1 \quad \Longleftrightarrow \quad (t,b,n) \in E_{\text{comp}} \quad \text{or} \quad \exists (t,b,s_v) \in E_{\text{comp}} \text{ with } P_{n,v} = 1 \tag{10}$$

where $P_{n,v} = 1$ indicates that neuron n belongs to symbol pattern v. The reconstructed spike tensor $\hat{\mathbf{S}}$ is then passed directly as the input to layer l_{n+1}.

3.5 Autotuner Logic

To determine a suitable hyperparameter configuration $(m^*, p^*, \sigma^*, \omega^*)$ for a specified compression target C_{target}, we develop a deterministic autotuner. Rather than performing an intractable exhaustive grid search, the autotuner employs a four-level hierarchical beam search with backtracking over a calibration subset D (set to 50% of the evaluation data). At each stage, the search space is pruned

to the top-K candidates ($K = 4$): levels 1 and 2 rank the Minimum Spike Rate (m) and Pattern Matching Threshold (p) by task accuracy, level 3 selects the Survival Probability (σ) to minimize the deviation from the target compression $|C - C_{\text{target}}|$, and level 4 tunes the Spiking Threshold (ω) for accuracy recovery. An immediate early exit is triggered once a configuration satisfies $C \geq C_{\text{target}}$ while maintaining an accuracy drop $(A_{\text{base}} - A) \leq 2\%$. If no configuration satisfies both criteria, the search returns the best-found configuration according to the lexicographic objective $(|C - C_{\text{target}}|, A_{\text{base}} - A)$. The complete search logic and pseudocode are detailed in Appendix A.1 (Algorithm 1).

4 EXPERIMENTS

In this section, we evaluate the performance, scalability, and efficiency of SkewSpike across diverse spiking architectures and datasets. Our empirical analysis is organized into four main parts: (1) a comparative benchmark against FSPC and Huffman coding on MNIST and Fashion-MNIST, (2) a scalability evaluation across deeper spiking architectures on complex datasets, (3) comprehensive ablation studies, including component analyses and one-factor-at-a-time (OFAT) parameter sweeps, to examine how individual mechanisms govern the trade-off between compression efficiency and task accuracy, and (4) hardware-aware energy evaluation. Data reduction is quantified as the percentage reduction in transmitted AER events: Compression $= (1 - |E_{\text{SkewSpike}}|/|E|) \times 100\%$, evaluated across three target compression regimes: 30% (speed-oriented), 50% (balanced), and 70% (aggressive). Complete hardware specifications, software implementation details, and autotuner search grids are provided in Appendix A.2.

4.1 COMPARISON

We benchmark SkewSpike against Frequent Spike Pattern Compression (FSPC) (Ganesh et al., 2025) and lossless Huffman coding. Following the experimental protocol of Ganesh et al. (2025), evaluations are conducted on MNIST and Fashion-MNIST across two fully connected architectures (784-100-10 and 784-100-60-10) using 3,000 test images per dataset. We evaluate SkewSpike under the 30% target compression regime (SS_{30}) to establish a comparable data reduction level against the baselines. Complete baseline configurations, algorithmic parameters, and layer insertion details are provided in Appendix A.3. As shown in Table 1, SkewSpike achieves 30.04% to 37.39% compression while maintaining or improving accuracy, outperforming Huffman coding and avoiding FSPC's severe accuracy loss on Fashion-MNIST.

Table 1: Comparison of SkewSpike (SS_{30}, 30% target) against FSPC and Huffman coding on MNIST and F-MNIST. Parameters denote (m, p, σ, ω) for SkewSpike and (fsp, pmt) for $\text{FSPC}_{\text{fpmax}}$. Baseline accuracy is reported in parentheses. Actual compression may moderately exceed the specified target.

Dataset	Model (Base %)	Method	Params.	Post Acc. (%)	Comp. (%)
MNIST	784-100-10 (93.17)	Huffman	–	93.17	8.26
		$\text{FSPC}_{\text{fpmax}}$	$(9, 0.3)$	94.30	16.08
		SS_{30}	$(0.0, 0.7, 0.8, 1.0)$	94.50	32.16
	784-100-60-10 (95.07)	Huffman	–	95.07	10.81
		$\text{FSPC}_{\text{fpmax}}$	$(7.0, 0.4)$	95.13	28.81
		SS_{30}	$(0.0, 0.8, 0.7, 1.0)$	94.17	37.39
F-MNIST	784-100-10 (86.27)	Huffman	–	86.27	10.30
		$\text{FSPC}_{\text{fpmax}}$	$(10.0, 0.3)$	80.63	34.95
		SS_{30}	$(0.0, 1.0, 0.7, 1.0)$	86.93	30.04
	784-100-60-10 (84.60)	Huffman	–	84.60	11.38
		$\text{FSPC}_{\text{fpmax}}$	$(10.0, 0.5)$	82.43	39.18
		SS_{30}	$(0.0, 0.9, 0.8, 1.0)$	84.30	37.14

4.2 EVALUATION ON DIVERSE MODELS AND DATASETS

We evaluate the scalability of SkewSpike across a broad range of spiking architectures and datasets. For digit classification, we evaluate 4-layer fully connected (FC) SNNs on MNIST, Fashion-MNIST,

and N-MNIST, reporting mean $\pm$ standard deviation over 4 random seeds, and compression as the mean across seeds. To benchmark scalability on deeper networks, we evaluate SpikingVGG11-BN, SpikingVGG16-BN, SEW-ResNet18, and SpikingResNet18 across CIFAR-10/100, DVS-CIFAR-10, DVS128-Gesture, ImageNet-100, and ImageNet-1k. Model pretraining protocols, simulation timesteps ($T = 20$ for FC, $T = 12$ for Conv), and training configurations are detailed in Appendix A.4. Results across the three target compression levels are reported in Table 2, with autotuned parameter tuples in Appendix Table 6.

Table 2: Scalability of SkewSpike across architectures and datasets. SS_x denotes $x\%$ target compression, reporting **Post Acc. (Comp.)** in %. Autotuned configurations are listed in Appendix Table 6. Abbreviations: F-MNIST (Fashion-MNIST), D-CIFAR10 (DVS-CIFAR-10), DVS-Gesture (DVS128-Gesture), IN-100/1k (ImageNet-100/1k), S-VGG (Spiking VGG), S-ResNet (Spiking ResNet).

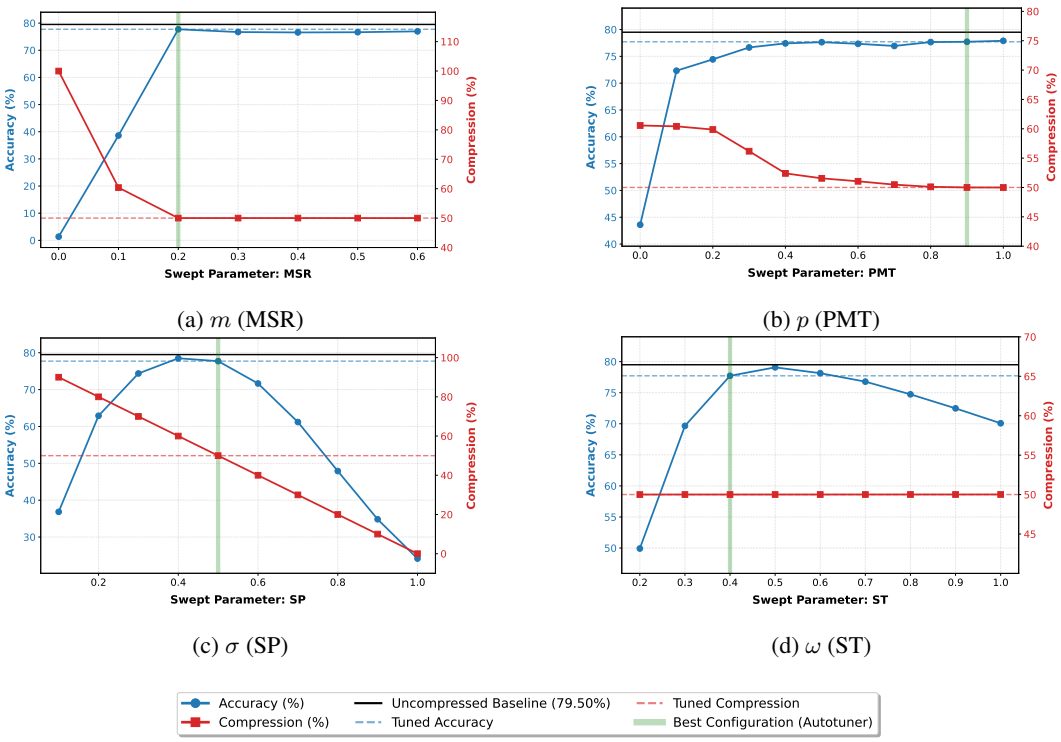
Dataset	Base (%)	SS_{30} [Acc. (Comp.)]	SS_{50} [Acc. (Comp.)]	SS_{70} [Acc. (Comp.)]
4-Layer FC (784-10k-5k-10)				
MNIST	98.43 ± 0.02	98.41 ± 0.04 (33.75)	97.85 ± 0.22 (53.64)	97.84 ± 0.55 (73.17)
F-MNIST	86.79 ± 0.17	87.25 ± 0.07 (30.94)	87.27 ± 0.24 (52.95)	87.13 ± 0.53 (71.76)
4-Layer FC (2312-10k-5k-10)				
N-MNIST	98.97 ± 0.05	98.91 ± 0.05 (33.52)	98.89 ± 0.07 (53.22)	98.48 ± 0.22 (75.41)
S-VGG11-BN				
CIFAR-10	90.04	87.88 (39.99)	88.00 (50.12)	79.00 (70.00)
D-CIFAR10	74.50	73.90 (30.09)	55.10 (50.00)	11.00 (70.00)
DVS-Gesture	92.71	92.28 (30.53)	57.98 (50.01)	30.21 (70.00)
S-VGG16-BN				
CIFAR-10	93.18	92.08 (30.73)	91.37 (50.42)	75.84 (69.99)
CIFAR-100	71.22	70.50 (30.42)	69.18 (50.59)	61.41 (69.99)
IN-100	66.84	63.82 (30.02)	44.66 (49.99)	4.18 (70.00)
SEW-ResNet18				
IN-100	79.50	79.50 (30.01)	77.72 (50.01)	76.98 (70.06)
S-ResNet18				
IN-100	77.90	76.26 (30.19)	75.32 (50.05)	71.98 (70.00)
IN-1k	64.16	62.05 (32.44)	62.11 (50.38)	56.37 (70.12)

Across the 12 benchmarks, SkewSpike preserves representational fidelity under the 30% target (SS_{30}), with fully connected models exhibiting negligible accuracy loss and SEW-ResNet18 matching its uncompressed ImageNet-100 baseline (79.50%). Under higher compression levels (SS_{50} and SS_{70}), network architecture becomes a decisive factor. Fully connected models and residual networks remain highly resilient. SEW-ResNet18 degrades by only 1.78% at 50% compression and retains 76.98% at 70% compression on ImageNet-100, while SpikingResNet18 maintains 62.11% on ImageNet-1k. Conversely, sequential VGG backbones show vulnerability beyond 30% compression on temporally complex DVS streams and high-dimensional ImageNet tasks. Because plain feedforward models lack skip connections, heavy pruning of intermediate residual spikes starves downstream layers of membrane integration, whereas residual shortcuts in ResNets provide essential alternate signal routes that protect against feature collapse under aggressive compression.

4.3 Ablation Study

To evaluate how individual hyperparameters and framework phases govern compression and accuracy, we conduct both one-factor-at-a-time (OFAT) parameter sweeps and component-level ablation analyses on SEW-ResNet18 evaluated on ImageNet-100 under the balanced 50% compression regime (SS_{50}).

OFAT Hyperparameter Sensitivity Analysis We systematically evaluate the sensitivity of each hyperparameter while holding the remaining three fixed at the autotuned baseline configuration: $(m^*, p^*, \sigma^*, \omega^*) = (0.2, 0.9, 0.5, 0.4)$. The evaluated ranges are $m \in [0.0, 0.6]$, $p \in [0.0, 1.0]$, $\sigma \in [0.1, 1.0]$, and $\omega \in [0.2, 1.0]$. As shown in Figure 2, each parameter exhibits distinct characteristics. In the MSR and PMT sweeps, performance collapses at low values ($m < 0.2$ or $p < 0.3$) due to dictionary noise and spurious pattern matching, then rapidly recovers and plateaus above these



(a) m (MSR) (b) p (PMT)

(c) σ (SP) (d) ω (ST)

Figure 2: OFAT sensitivity analysis of hyperparameters on SEW-ResNet18 (ImageNet-100). All non-varying parameters are held fixed at the autotuned SS_{50} baseline: $(m^*, p^*, \sigma^*, \omega^*) = (0.2, 0.9, 0.5, 0.4)$.

thresholds at stable accuracy and 50% compression. In the SP sweep, compression decreases strictly linearly, while accuracy follows an inverted U-shaped curve, peaking at $\sigma = 0.4$. The ST sweep similarly yields an inverted U-shaped accuracy curve peaking at $\omega = 0.5$, while maintaining constant compression at 50.01%, confirming that threshold calibration decouples accuracy recovery from data volume. Isolated parameter settings can yield marginally higher accuracy than the autotuned anchor because the autotuner prioritizes fast convergence via beam search with early termination on the calibration subset rather than exhaustive global grid search.

Component Ablation Analysis To isolate the contribution of each framework phase, we conduct an incremental ablation on SEW-ResNet18 (ImageNet-100) under the 50% compression target, summarized in Table 3. Using dictionary substitution alone ($m = 0.2, p = 0.9$) while retaining all residuals ($\sigma = 1.0, \omega = 1.0$) achieves 79.48% accuracy, identical to the uncompressed baseline. This indicates that dictionary substitution introduces virtually no representational distortion. Enabling stochastic residual pruning ($\sigma = 0.5$) achieves the targeted 50.01% compression, but drops accuracy to 70.08% due to residual feature loss. Finally, adding spike threshold calibration ($\omega = 0.4$) recovers +7.64% accuracy, restoring performance to 77.72% with zero compression penalty. This confirms that threshold calibration effectively compensates for accuracy degradation caused by residual pruning.

Table 3: Component-level ablation study on SEW-ResNet18 (ImageNet-100) under 50% target compression.

Configuration	**MSR** (m)	**PMT** (p)	**SP** (σ)	**ST** (ω)	**Acc. (%)**	**Comp. (%)**
Baseline (Uncompressed)	–	–	–	–	79.50	0.00
Dictionary Only	0.2	0.9	1.0	1.0	79.48	0.03
Dictionary + SP	0.2	0.9	0.5	1.0	70.08	50.01
Full SkewSpike	0.2	0.9	0.5	0.4	77.72	50.01

4.4 ENERGY EVALUATION

To quantify the practical energy benefit of spike compression, we evaluate its impact on spike communication in neuromorphic hardware across two-dimensional (2D) and three-dimensional (3D) mesh Network-on-Chip (NoC) architectures implemented in 45-nm Nangate CMOS technology.

Because the static leakage power of the synthesized NoC remains constant at 7.64×10^{-4} W for each router regardless of workload, we focus our evaluation on dynamic communication energy. From post-synthesis gate-level simulation and power analysis, we obtain a dynamic communication energy of $E_{\text{bit/hop}}^{\text{dynamic}} = 0.92546$ pJ/bit/hop. The total dynamic communication energy is calculated as $E_{\text{dynamic}} = \sum_{s=1}^{N_{\text{spike}}} B_{\text{spike}} H_s E_{\text{bit/hop}}^{\text{dynamic}}$, where B_{spike} is the spike bitwidth (16 bits) and H_s is the number of hops traversed across the NoC (complete derivations and synthesis configurations are detailed in Appendix A.6). Table 4 summarizes the spike activity and estimated dynamic NoC communication energy before and after compression across the evaluated networks.

As shown in Table 4 (evaluated using random seed 42), the reduction in spike volume achieved by SkewSpike directly translates into proportional dynamic communication energy savings across both 2D and 3D NoC topologies. For fully connected architectures under the 70% target, transmitted spike volume decreases by 71.2% to 80.0%, reducing dynamic energy by over 71% on MNIST and Fashion-MNIST, and by 80.0% on N-MNIST (from 86.85 mJ down to 17.36 mJ in 2D), while maintaining accuracy within 1.3% of the uncompressed baseline. On the deeper Spiking VGG11 model (CIFAR-10, 50% target), SkewSpike eliminates 54.6% of spike traffic, cutting dynamic communication energy by more than half (from 10.99 J to 4.98 J in 2D, and from 6.16 J to 2.80 J in 3D) with only a 2.04% accuracy drop.

Table 4: Spike activity, classification accuracy, and estimated dynamic NoC communication energy before and after spike compression across 2D and 3D mesh architectures.

Dataset	Target	Accuracy (%)		Spike Activity		Dynamic Energy			
		Original	Compressed	Original Spikes	Compressed Spikes	2D		3D	
						Original	Compressed	Original	Compressed
784-10k-5k-10									
MNIST	70%	98.45	97.16	126,526,334	36,436,321	231.62 mJ	66.70 mJ	220.48 mJ	63.48 mJ
F-MNIST	70%	86.77	87.21	214,975,914	61,052,581	393.29 mJ	111.71 mJ	374.90 mJ	106.47 mJ
2312-10k-5k-10									
N-MNIST	70%	98.98	98.27	46,785,653	9,353,975	86.85 mJ	17.36 mJ	81.73 mJ	16.34 mJ
Spiking VGG11									
CIFAR-10	50%	90.04	88.00	109,956,242	49,932,400	10.99 J	4.98 J	6.16 J	2.80 J

5 CONCLUSION

In this paper, we presented SkewSpike, a hierarchical compression framework designed to reduce Address Event Representation (AER) spike event volume in rate-coded SNNs. By mapping synchronous co-firing patterns into right-skewed tree dictionaries, SkewSpike aggregates redundant neural activity into compact symbols. The framework combines stochastic residual pruning with adaptive threshold calibration to protect salient features in the distribution tail, supported by a deterministic autotuner that automates parameter configuration. Extensive evaluations across 12 benchmarks confirm that SkewSpike substantially lowers transmitted spike event volume with minimal impact on task accuracy, providing a practical foundation for reducing transmission load in neuromorphic systems.

Limitation Key limitations include evaluating compression only between a single pair of layers, sharp accuracy drops on VGG models and neuromorphic datasets, the autotuner trading global optimality for search speed, and the initial overhead of k-means dictionary construction. Future work includes extending SkewSpike across multiple network layers, reducing accuracy loss on VGG and neuromorphic benchmarks, and improving the autotuner to reach near-global solutions.

AI USE STATEMENT

In this work, we used generative AI tools for coding assistance, paraphrasing, LaTeX formatting, and assistance in articulating interpretations of experimental results that had been independently derived by the authors. We did not use generative AI tools for core problem formulation, mathematical proofs, experimental design, or generation of experimental data. All scientific concepts, framework designs, experimental setups, and empirical analyses were conceived and directed by the authors. AI-assisted code was reviewed, verified, and thoroughly tested by the authors. All AI-assisted text was likewise reviewed and revised by the authors. We take full responsibility for the final content of this work, including all claims, analyses, code, and other artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the empirical results of this work are reproducible, we provide the source code, training scripts, and evaluation pipelines for the SkewSpike framework as supplementary material, covering the experiments presented in Section 4.2 (Evaluation on Diverse Models and Datasets) and Section 4. Detailed descriptions of the experimental setups, data preprocessing pipelines, and hardware environments are provided in Section A.2. Furthermore, complete training recipes, network architectural modifications, simulation timesteps, and optimization hyperparameters for the evaluated models (fully connected MLPs, SpikingVGG, SEW-ResNet, and SpikingResNet) are detailed in Appendix A.5 and Appendix A.4. The deterministic autotuner search algorithm and its operational logic are formally defined in Appendix A.1 (Algorithm 1), with the exact autotuned hyperparameter configurations cataloged in Appendix Table 6. The NoC hardware synthesis and energy evaluation details are provided in Appendix A.6.

REFERENCES

Abderazek Ben Abdallah and Khanh N. Dang. *Neuromorphic computing principles and organization (1st edition)*. Springer, 1 edition, 2022. ISBN 978-3-030-92524-6.

Andreas G. Andreou, Andrew A. Dykman, Kate D. Fischl, Guillaume Garreau, Daniel R. Mendat, Garrick Orchard, Andrew S. Cassidy, Paul Merolla, John Arthur, Rodrigo Alvarez-Icaza, Bryan L. Jackson, and Dharmendra S. Modha. Real-time sensory information processing using the truenorth neurosynaptic system. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2911–2911, 2016. doi: 10.1109/ISCAS.2016.7539214.

Tong Bu, Xinyu Shi, and Zhaofei Yu. Activity pruning for efficient spiking neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 1713–1721. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/236. Main Track.

Khanh N Dang et al. Hetergenmap: An evolutionary mapping framework for heterogeneous noc-based neuromorphic systems. *IEEE Access*, 11:144095–144112, 2023.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios 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. doi: 10.1109/MM.2018.112130359.

Michael V. DeBole, Brian Taba, Arnon Amir, Filipp Akopyan, Alexander Andreopoulos, William P. Risk, Jeff Kusnitz, Carlos Ortega Otero, Tapan K. Nayak, Rathinakumar Appuswamy, Peter J. Carlson, Andrew S. Cassidy, Pallab Datta, Steven K. Esser, Guillaume J. Garreau, Kevin L. Holland, Scott Lekuch, Michael Mastro, Jeff McKinstry, Carmelo di Nolfo, Brent Paulovicks, Jun

Sawada, Kai Schleupen, Benjamin G. Shaw, Jennifer L. Klamo, Myron D. Flickner, John V. Arthur, and Dharmendra S. Modha. Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5):20–29, 2019. doi: 10.1109/MC.2019.2903009.

Yongqi Ding, Lin Zuo, Mengmeng Jing, Pei He, and Yongjun Xiao. Shrinking your timestep: Towards low-latency neuromorphic object recognition with spiking neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:11811–11819, 03 2024. doi: 10.1609/aaai.v38i10.29066.

Jason K. Eshraghian et al. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023. doi: 10.1109/JPROC.2023.3308088.

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. doi: 10.1126/sciadv.adi1480.

Satvik Ganesh et al. FSPC: A Lossy Spike Compression Through Correlated-AER Merging in Spiking Neural Networks. In *2025 IEEE 18th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 386–393, 12 2025. doi: 10.1109/MCSoC67473.2025.00068.

Wenzhe Guo, Mohammed E. Fouda, Hasan Erdem Yantır, Ahmed M. Eltawil, and Khaled Nabil Salama. Unsupervised adaptive weight pruning for energy-efficient neuromorphic systems. *Frontiers in Neuroscience*, 14, 2020.

Wenzhe Guo et al. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:638474, 2021.

Jiawei Han et al. Mining frequent patterns without candidate generation. In *ACM SIGMOD Conference*, pp. 1–12, 2000.

Ofer Hasson. PyTorch KMeans, July 2025. URL `https://gitlab.com/hassonofer/pt_kmeans`.

David Kappel and Christian Tetzlaff. Synapses learn to utilize stochastic pre-synaptic release for the prediction of postsynaptic dynamics. *PLOS Computational Biology*, 20, 2024.

Sung-Eun Kim, Kwang-Il Oh, Taewook Kang, Sukho Lee, Hyuk Kim, Mi-Jeong Park, and Jae-Jin Lee. An efficient aer interface circuit for scalable spiking neural networks. In *2023 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*, pp. 1–3, 2023. doi: 10.1109/ITC-CSCC58803.2023.10212803.

Louis Lapique. Researches quantatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology, Pathology and Genetics*, 9:620–635, 1907.

Chen Li, Lei Ma, and Steve Furber. Quantization framework for fast spiking neural networks. *Frontiers in Neuroscience*, 16:918793, 07 2022. doi: 10.3389/fnins.2022.918793.

A. Linares-Barranco, G. Jimenez-Moreno, B. Linares-Barranco, and A. Civit-Balcells. On Algorithmic Rate-Coded AER Generation. *IEEE Transactions on Neural Networks*, 17(3):771–788, 2006. doi: 10.1109/TNN.2006.872253.

Hin Wai Lui and Emre Neftci. Hessian aware quantization of spiking neural networks. In *International Conference on Neuromorphic Systems 2021*, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386913. doi: 10.1145/3477145.3477158.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Electron. Colloquium Comput. Complex.*, TR96, 1996.

Misha A. Mahowald. *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function*. Phd dissertation, California Institute of Technology, Pasadena, CA, USA, May 1992.

James Malkin, Cian O'Donnell, Conor J Houghton, and Laurence Aitchison. Signatures of bayesian inference emerge from energy-efficient synapses. *eLife*, 12:RP92595, aug 2024. ISSN 2050-084X. doi: 10.7554/eLife.92595.

Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12434–12443, 06 2022. doi: 10.1109/CVPR52688.2022.01212.

Kenji Mizuseki and György Buzsáki. Preconfigured, skewed distribution of firing rates in the hippocampus and entorhinal cortex. *Cell Reports*, 4(5):1010–1021, 2013. ISSN 2211-1247. doi: https://doi.org/10.1016/j.celrep.2013.07.039.

James S. Plank, Charles P. Rizzo, Bryson Gullett, Keegan E. M. Dent, and Catherine D. Schuman. Alleviating the communication bottleneck in neuromorphic computing with custom-designed spiking neural networks. *Journal of Low Power Electronics and Applications*, 15(3), 2025. ISSN 2079-9268. doi: 10.3390/jlpea15030050.

Ning Qiao and Giacomo Indiveri. A bi-directional address-event transceiver block for low-latency inter-chip communication in neuromorphic systems. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018. doi: 10.1109/ISCAS.2018.8351623.

Sebastian Raschka. Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *The Journal of Open Source Software*, 3(24), April 2018. doi: 10.21105/joss.00638.

Alex Roxin, Nicolas Brunel, David Hansel, Gianluigi Mongillo, and Carl van Vreeswijk. On the distribution of firing rates in networks of cortical neurons. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 31:16217–26, 11 2011. doi: 10.1523/JNEUROSCI.1677-11.2011.

Xinyu Shi, Jianhao Ding, Zecheng Hao, and Zhaofei Yu. Towards energy efficient spiking neural networks: An unstructured pruning framework. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 50818–50838, 2024.

Massimo Antonio Sivilotti. *Wiring considerations in analog VLSI systems, with application to field-programmable networks*. California Institute of Technology, 1991.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Zhanglu Yan, Zhenyu Bai, Kaiwen Tang, and Weng-Fai Wong. Reconsidering the energy efficiency of spiking neural networks inference from analytical perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2026. doi: 10.1109/TCAD.2026.3718799.

A Appendix

A.1 Autotuner Search Algorithm and Pseudocode

The deterministic autotuning procedure is outlined in Algorithm 1. The autotuner systematically explores discrete parameter spaces across four hierarchical levels, using a beam width of $K = 4$ to balance optimization time and solution quality. All evaluations during the search are conducted on the calibration subset D. If the target constraints ($C \geq C_{\text{target}}$ and accuracy loss within 2%) are met during level 4 evaluation, the algorithm terminates immediately via early exit. Otherwise, the search backtracks across candidate branches, ultimately returning the global configuration that minimizes target divergence and accuracy loss.

Algorithm 1 Autotuner Parameter Search

Input: Model, Calibration subset D, Target compression C_{target}, Beam width $K = 4$
Output: Best parameter configuration $(m^*, p^*, \sigma^*, \omega^*)$
$A_{\text{base}} \leftarrow$ EvaluateBaseline(Model, D)
Initialize BestConfig $\leftarrow$ nil with score ∞
▷ **Level 1: Explore Top-K m**
for each candidate $m \in$ Top-$K(m \in [0, 1]$ by Accuracy) **do**
  ▷ **Level 2: Explore Top-K p for current m**
  for each candidate $p \in$ Top-$K(p \in [0, 1]$ by Accuracy) **do**
    ▷ **Level 3: Explore Top-K σ for current (m, p)**
    for each candidate $\sigma \in$ Top-$K(\sigma \in [0, 1]$ by $|C - C_{\text{target}}|)$ **do**
      ▷ **Level 4: Evaluate ω**
      for each $\omega \in [0.2, 1.0]$ **do**
        Model$' \leftarrow$ AdjustSpikingThreshold(Model, m, ω)
        $(A, C) \leftarrow$ EvaluateConfiguration(Model$', D, m, p, \sigma$)
        Update BestConfig with lowest $(|C - C_{\text{target}}|,\ A_{\text{base}} - A)$
        if $C \geq C_{\text{target}}$ **and** $(A_{\text{base}} - A) \leq 2\%$ **then**
          return (m, p, σ, ω) ▷ Target met: immediate early exit
        end if
      end for
      ▷ Dead-end: backtrack to next σ candidate
    end for
    ▷ Dead-end: backtrack to next p candidate
  end for
  ▷ Dead-end: backtrack to next m candidate
end for
return BestConfig ▷ Fallback to global best-effort configuration

A.2 Experimental Setup, Hardware, and Parameter Search Grids

AER Compression Metric Formulation: To quantify the reduction in transmitted spike volume across layer interfaces, we compute the compression percentage based on the cardinality of the AER event sets. Let $|E|$ denote the total number of spike events in the uncompressed baseline AER set, as formulated in Section 3.3. We denote $E_{\text{SkewSpike}}$ as the set of compressed units transmitted by our framework, comprising triggered pattern symbols and surviving residual spikes. The compression percentage is formally computed as:

$$\text{Compression (\%)} = \left(1 - \frac{|E_{\text{SkewSpike}}|}{|E|}\right) \times 100\% \tag{11}$$

Hardware and Software Environments: All training, fine-tuning, and simulation experiments were conducted on a server equipped with four NVIDIA H100 GPUs (80 GB VRAM each). SNN models were implemented in PyTorch using the SpikingJelly framework (Fang et al., 2023) and optimized via Backpropagation Through Time (BPTT) Werbos (1990) with surrogate gradient descent. Fast dictionary clustering during the offline calibration phase utilizes the GPU-accelerated PyTorch k-means implementation (Hasson, 2025). For the LIF-neuron, we use the default values $V_{\text{th}} = 1.0$ and $V_{\text{reset}} = 0$, with an ArcTangent (ATan) surrogate gradient (Werbos, 1990) during backpropagation through time (BPTT).

Autotuner Search Space and Target Regimes: The deterministic autotuner searches over discrete parameter grids defined as:

- Minimum Spike Rate: $m \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$
- Pattern Matching Threshold: $p \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$
- Survival Probability: $\sigma \in \{0.1, 0.2, \ldots, 1.0\}$
- Spiking Threshold scaling factor: $\omega \in \{0.2, 0.3, \ldots, 1.0\}$

The calibration subset D is set to 50% of the evaluation data. The autotuner optimizes these parameters to satisfy three target AER compression regimes: 30% (speed-oriented for high-throughput bandwidth constraints), 50% (balanced trade-off between compression and accuracy), and 70% (aggressive data reduction for severely resource-limited edge hardware).

A.3 Implementation Details for Comparison Against FSPC and Huffman Coding

To ensure a fair evaluation against prior work, all baseline implementations follow the experimental protocol described by Ganesh et al. (2025):

Simulation Configuration: Both fully connected architectures (784-100-10 and 784-100-60-10) are simulated with a fixed temporal window of $T = 20$ timesteps. Evaluations are conducted on 3,000 test images sampled from the MNIST and Fashion-MNIST datasets.

Network Architecture and SpikingJelly Translation: Models originally designed in snnTorch were adapted for SpikingJelly. Neurons are implemented as `LIFNode` instances with an ArcTangent surrogate gradient, threshold $V_{\text{th}} = 1.0$, and reset-by-subtraction (`v_reset=None`). The decay factor $\beta = 0.95$ from snnTorch is mapped to the SpikingJelly membrane time constant via $\tau = 1.0/(1.0 - \beta)$.

Training Recipe: Models are trained in full single precision (FP32) using the standard Adam optimizer with a learning rate of 5×10^{-4} and no weight decay. The objective function applies CrossEntropyLoss to the membrane potential at each timestep, summed over the full simulation window. An activity regularization penalty ($10.0 \times (r - 0.2)^2$) is applied to the first hidden layer if its average firing rate r exceeds 0.20 to prevent excessive co-firing. Neuronal states are reset between mini-batches.

FSPC Implementation: For FSPC (Ganesh et al., 2025), frequent co-activation patterns are extracted using the FP-Max (`fpmax`) algorithm (Han et al., 2000) implemented via the `mlxtend` library (Raschka, 2018). Following the authors' recommendations for optimal trade-offs, the minimum support threshold is set to 0.25 for the three-layer network on both datasets and for the four-layer network on MNIST. For the four-layer network on Fashion-MNIST, the minimum support threshold is set to 0.35.

FSPC Parameter Tuning and Runtime Complexity: Because FSPC does not include an automated parameter optimization mechanism, we conducted an exhaustive grid search over its discrete parameter space: FSP $\in \{1, 2, \ldots, 10\}$ and PMT $\in \{0.0, 0.1, \ldots, 1.0\}$. To ensure an objective evaluation, the optimal configuration was determined by maximizing the trade-off score, defined as the normalized harmonic mean of test accuracy and achieved compression across all grid combinations. Furthermore, frequent-pattern mining algorithms such as FP-Max incur high algorithmic complexity during inference, resulting in an average latency of 2.38 to 17.70 seconds per batch. In contrast, SkewSpike operates via vectorized dictionary lookups and simple arithmetic thresholding, achieving inference latencies of 2.36 to 2.79 milliseconds per batch (a speedup of over $1{,}000\times$). The offline autotuning search for SkewSpike completes in approximately 8 to 12 seconds across all cases, while offline dictionary construction requires only 1.04 to 1.93 seconds. A complete computational latency comparison is summarized in Table 5.

Huffman Coding Implementation: For static Huffman coding, a frequency histogram is calibrated over 50% of the test dataset. An optimal prefix-free binary tree is then constructed using a priority queue to assign exact variable-length codes to all active neurons, with codebook header overhead explicitly included by accounting for active symbol indices and code lengths. To avoid the software overhead of CPU-based bit manipulation and variable-length bit packing, compression is evaluated using a vectorized GPU lookup of the exact Huffman code length for each active AER event. Since Huffman coding is lossless, this approach provides an exact theoretical bit-length and storage footprint for the corresponding Huffman representation without modifying the underlying spike representation.

Layer Placement: All compression frameworks (SkewSpike, FSPC, and Huffman coding) are inserted at the intermediate hidden-layer interfaces during inference: between the 100-neuron and 10-neuron layers in the 784-100-10 model, and between the 100-neuron and 60-neuron layers in the 784-100-60-10 model.

Table 5: Inference latency and offline setup overhead for SkewSpike, FSPC, and Huffman coding. Inference time is reported as mean runtime per batch in seconds. Dictionary setup denotes the one-time offline calibration duration. Autotuning for SkewSpike requires approximately 8 to 12 seconds per benchmark.

Dataset	Architecture	Method	Comp. Runtime (s/batch)	Offline Dict. (s)
MNIST	784-100-10	Huffman	0.00184	0.176
		$FSPC_{fpmax}$	2.37780	–
		SS_{30}	0.00237	1.926
	784-100-60-10	Huffman	0.00111	0.176
		$FSPC_{fpmax}$	17.69570	–
		SS_{30}	0.00236	1.190
F-MNIST	784-100-10	Huffman	0.00054	0.180
		$FSPC_{fpmax}$	6.46790	–
		SS_{30}	0.00279	1.042
	784-100-60-10	Huffman	0.00050	0.205
		$FSPC_{fpmax}$	13.51160	–
		SS_{30}	0.00240	1.069

A.4 Implementation Details for Evaluation on Diverse Models and Datasets

Simulation Timesteps: Fully connected architectures (MNIST, Fashion-MNIST, and N-MNIST) are evaluated with a temporal window of $T = 20$ timesteps. For deep convolutional architectures (SpikingVGG11-BN, SpikingVGG16-BN, SEW-ResNet18, and SpikingResNet18), the simulation temporal length is set to $T = 12$.

Weight Initialization and Pretraining: For models evaluated on static image datasets (CIFAR and ImageNet benchmarks), networks were initialized with pretrained ANN weights fine-tuned on the respective datasets using surrogate gradient descent. Neuromorphic event-driven benchmarks (N-MNIST, DVS-CIFAR-10, and DVS128-Gesture) were trained directly from scratch. Fully connected networks are evaluated across 4 random seeds to ensure statistical consistency.

Model Architectural Adjustments:

- *SpikingVGG11-BN (for DVS-CIFAR-10 and DVS128-Gesture):* The first convolutional layer is set to 2 input channels to match DVS event polarities. All `MaxPool2d` layers are replaced with `AvgPool2d`. The classifier uses `AdaptiveAvgPool2d((1, 1))` followed by a two-layer linear network with dropout (0.4) and an intermediate integrate-and-fire node.
- *SpikingVGG16-BN (for CIFAR-10 and CIFAR-100):* Feature layers at indices 6 and 13 are replaced with `nn.Identity` layers. The classifier uses `AdaptiveAvgPool2d((1, 1))` followed by dropout (0.3) and a single linear layer to output class predictions.

SkewSpike Placement and Layer Insertion Points: During inference, the SkewSpike compression framework is integrated at specific intermediate layer interfaces:

- *Fully Connected SNNs (784-10000-5000-10 and 2312-10000-5000-10):* Inserted between the 10,000-neuron hidden layer and the 5,000-neuron hidden layer.
- *SpikingVGG11-BN:* Inserted at the output of intermediate feature layer `features[10]`.
- *SpikingVGG16-BN:* Inserted at the output of intermediate feature layer `features[26]`.
- *SEW-ResNet18 and SpikingResNet18:* Inserted at the first spiking neuron activation of the initial residual block (`layer1[0].sn1`).

Hyperparameter Configurations: Table 6 lists the autotuned hyperparameter configurations across all evaluated benchmarks for target compression levels SS_{30}, SS_{50}, and SS_{70}. Each entry reports the tuple (m, p, σ, ω), denoting the Minimum Spike Rate (m), Pattern Matching Threshold (p), Survival Probability (σ), and Spiking Threshold (ω). For models evaluated over four random seeds, the dominant configuration is presented with bracketed ranges indicating seed-to-seed variations.

Across all benchmarks, the autotuner systematically scales down σ to enforce higher data reduction while adjusting ω to preserve accuracy.

Table 6: Autotuned hyperparameter configurations (m, p, σ, ω) for all evaluated models and datasets across compression targets (SS_{30}, SS_{50}, SS_{70}). For 4-seed experiments, the dominant configuration is reported, with bracketed ranges indicating seed-to-seed variations.

Architecture	Dataset	$\mathbf{SS}_{30}$ (m, p, σ, ω)	$\mathbf{SS}_{50}$ (m, p, σ, ω)	$\mathbf{SS}_{70}$ (m, p, σ, ω)
784-10k-5k-10	MNIST	$(0.0, [0.6, 1.0], [0.7, 0.9], 1.0)$	$(0.0, [0.5, 0.8], [0.5, 0.7], 1.0)$	$(0.0, [0.2, 0.3], [0.8, 1.0], 1.0)$
	F-MNIST	$(0.0, [0.8, 1.0], 0.7, 1.0)$	$(0.0, [0.7, 1.0], 0.5, 1.0)$	$(0.0, [0.7, 1.0], 0.3, 1.0)$
2312-10k-5k-10	N-MNIST	$(0.0, [0.8, 1.0], [0.6, 0.7], 1.0)$	$(0.0, [0.8, 1.0], [0.4, 0.5], 1.0)$	$(0.0, [0.8, 1.0], [0.2, 0.3], 1.0)$
S-VGG11-BN	CIFAR-10	$(0.1, 1.0, 0.6, 0.6)$	$(0.2, 0.2, 0.5, 0.6)$	$(0.2, 1.0, 0.3, 0.5)$
	D-CIFAR10	$(0.2, 0.6, 0.7, 0.5)$	$(0.2, 1.0, 0.5, 0.5)$	$(0.2, 1.0, 0.3, 0.5)$
	DVS-Gesture	$(0.4, 0.7, 0.7, 0.7)$	$(0.4, 1.0, 0.5, 0.2)$	$(0.4, 1.0, 0.3, 0.6)$
S-VGG16-BN	CIFAR-10	$(0.1, 0.6, 0.7, 0.5)$	$(0.1, 0.6, 0.5, 0.4)$	$(0.4, 1.0, 0.3, 0.2)$
	CIFAR-100	$(0.1, 0.8, 0.7, 0.7)$	$(0.1, 0.6, 0.5, 0.5)$	$(0.4, 1.0, 0.3, 0.2)$
	IN-100	$(0.1, 0.9, 0.7, 0.7)$	$(0.4, 0.0, 0.5, 0.4)$	$(0.1, 1.0, 0.3, 0.8)$
SEW-ResNet18	IN-100	$(0.2, 0.9, 0.7, 0.7)$	$(0.2, 0.9, 0.5, 0.4)$	$(0.4, 0.4, 0.3, 0.2)$
S-ResNet18	IN-100	$(0.1, 0.5, 0.7, 0.6)$	$(0.2, 0.6, 0.5, 0.6)$	$(0.3, 0.8, 0.3, 0.2)$
	IN-1k	$(0.2, 0.6, 0.7, 0.6)$	$(0.3, 0.8, 0.5, 0.6)$	$(0.5, 0.6, 0.3, 0.2)$

Computational Overhead and Offline Calibration Complexity: Table 7 shows the setup time and inference latency across representative models under the SS_{50} target using seed 42. Building the dictionary is fast, taking from 8.63 seconds on DVS-CIFAR10 up to 175.61 seconds on ImageNet-1k. The autotuner needs only 1.2 to 1.4 minutes on digit datasets, 10 to 45 minutes on medium vision tasks, and about 9.1 hours on ImageNet-1k because of its large 25,000-image calibration set. Importantly, both dictionary building and autotuning run only once offline before deployment. During actual inference, the total time to encode and decode each batch is very low, ranging from 7.2 ms to 236 ms across all models.

Table 7: Offline setup time and per-batch inference latency of SkewSpike across representative architectures and datasets at 50% compression (SS_{50}), using random seed 42. Autotuning and dictionary construction are one-time offline procedures; latency includes encoding and decoding time per batch.

Architecture	Dataset	Offline Dict. (s)	Autotuner Time	Comp. Runtime (s/batch)
784-10k-5k-10	F-MNIST	35.75	76.91 s (1.3 min)	0.0190
2312-10k-5k-10	N-MNIST	142.43	86.02 s (1.4 min)	0.0072
SpikingVGG11-BN	D-CIFAR10	8.63	1389.90 s (23.2 min)	0.2267
SpikingVGG16-BN	CIFAR-100	24.11	2712.84 s (45.2 min)	0.0147
SEW-ResNet18	IN-100	23.87	646.56 s (10.8 min)	0.2360
SpikingResNet18	IN-1k	175.61	32696.89 s (9.1 h)	0.1570

A.5 Network Training Recipe and Optimization Details

Data Preprocessing and Memory Management: To optimize training throughput, input tensors are preprocessed before being transferred to the GPU:

- *Event Data:* Formatted into a time-first layout $[T, B, C, H, W]$ for convolutional SNNs, or flattened to $[T, B, C \cdot H \cdot W]$ for fully connected SNNs.
- *Static Images:* Transferred as 4D tensors and expanded along the temporal dimension to $[T, B, C, H, W]$ for convolutional models, or flattened and repeated across timesteps to $[T, B, N]$ for fully connected models.

All data transfers use non-blocking memory copies to prevent PCIe bottlenecks.

Optimization and Learning Rate Schedule: Models are trained using the `AdamW` optimizer. The default learning rate is set to 10^{-3} with a weight decay of 10^{-2}. For fine-tuned models, the weight decay is reduced to 10^{-4}. For SpikingResNet18 on ImageNet datasets, both the learning rate and

weight decay are set to 10^{-4}. We use a learning rate schedule with a linear warmup for the first 10% of epochs, followed by cosine annealing decay to zero.

Loss Function and Training Stability: We use standard Cross-Entropy loss calculated over the mean firing rate of the output layer across all timesteps. Models are trained using mixed precision (`bfloat16`) with gradient scaling. An effective batch size of 128 is maintained via gradient accumulation. We apply gradient clipping with a maximum norm of 1.0 to ensure stable training. Neuronal membrane potentials are reset after each batch to prevent temporal leakage between samples.

Reproducibility and Seeds: Fully connected models are evaluated across four random seeds (10, 42, 60, 85), reporting the mean and standard deviation. Convolutional models are evaluated using a fixed random seed of 42, except for SpikingResNet18 on ImageNet datasets, which uses standard initialization.

A.6 NoC Hardware Synthesis and Energy Evaluation Details

To quantify the practical energy benefit of spike compression, we evaluate its impact on spike communication in neuromorphic hardware. We consider both two-dimensional (2D) and three-dimensional (3D) Network-on-Chip (NoC) architectures implemented using the 45-nm Nangate CMOS technology. The NoC hardware models provide a hardware-level estimate of the energy required to transmit spike data, which is combined with the measured spike activity before and after compression.

We evaluate two NoC topologies: a conventional 2D mesh NoC and a 3D mesh NoC. The NoCs are described in Verilog HDL and synthesized using Synopsys Design Compiler with the 45-nm Nangate CMOS library. For energy evaluation, we perform post-synthesis simulations using QuestaSim to generate switching activity files, which are then analyzed using Synopsys PrimeTime. This flow enables the dynamic and static energy consumption of the NoC to be estimated from the synthesized hardware.

The total NoC energy is divided into dynamic and static components. The static component, or leakage power, remains constant at 7.64×10^{-4} W for the evaluated NoC configuration. Therefore, we focus primarily on dynamic energy, which directly depends on the amount of spike traffic. From the post-synthesis simulation and PrimeTime analysis, we obtain a dynamic communication energy of $E_{\text{bit/hop}}^{\text{dynamic}} = 0.92546$ pJ/bit/hop (9.2546×10^{-13} J/bit/hop). Here, one hop represents the traversal of one bit from one router to an adjacent router within the NoC. This hardware-derived energy-per-bit-per-hop is subsequently used to estimate the dynamic communication energy for the original and compressed spike streams. Table 8 summarizes the hardware configurations used in the evaluation.

Table 8: Hardware configurations used for the NoC energy evaluation.

Parameter	**Value**
Technology	45-nm CMOS
Topology	2D Mesh and 3D Mesh
Processing Element Configuration	256 neurons per PE
Neuron Mapping	Genetic Algorithm for minimizing traveling distances
Router Architecture	4-slot input buffer, matrix arbitration, stall-go flow control
Routing	XY (2D), XYZ (3D), nearest-neighbor (compressed AER)
Flit Width	16 bits
Clock Frequency	500 MHz
Voltage	1.1 V
Static Power	7.64×10^{-4} W
Dynamic Energy ($E_{\text{bit/hop}}^{\text{dynamic}}$)	9.2546×10^{-13} Joule/hop/bit

Spike Communication Energy Formulation: Let N_{spike} denote the number of spikes transmitted through the NoC, B_{spike} denote the number of bits required to represent one spike, and H_s denote the number of hops traversed by spike s. Here, one hop corresponds to the traversal of a spike from one router to an adjacent router, a router to a core, or from a core to a router. We define $E_{\text{bit/hop}}^{\text{dynamic}}$ as the dynamic energy required to transmit one bit across one NoC hop. The dynamic communication

energy of spike s is estimated as:

$$E_{\text{spike}}^{(s)} = B_{\text{spike}} H_s E_{\text{bit/hop}}^{\text{dynamic}} \tag{12}$$

The total dynamic communication energy is obtained by summing the energy of all transmitted spikes:

$$E_{\text{dynamic}} = \sum_{s=1}^{N_{\text{spike}}} B_{\text{spike}} H_s E_{\text{bit/hop}}^{\text{dynamic}} \tag{13}$$

For the original spike stream, the dynamic communication energy is:

$$E_{\text{dynamic}}^{\text{orig}} = \sum_{s=1}^{N_{\text{orig}}} B_{\text{spike}} H_s^{\text{orig}} E_{\text{bit/hop}}^{\text{dynamic}} \tag{14}$$

where N_{orig} is the number of spikes generated by the original network and H_s^{orig} is the number of hops traversed by the s-th spike.

After applying the proposed compression method, the dynamic communication energy becomes:

$$E_{\text{dynamic}}^{\text{comp}} = \sum_{s=1}^{N_{\text{comp}}} B_{\text{spike}} H_s^{\text{comp}} E_{\text{bit/hop}}^{\text{dynamic}} \tag{15}$$

where N_{comp} is the number of transmitted spikes after compression. The resulting dynamic-energy reduction is calculated as:

$$R_{\text{energy}} = 1 - \frac{E_{\text{dynamic}}^{\text{comp}}}{E_{\text{dynamic}}^{\text{orig}}} \tag{16}$$

This formulation accounts for both the number of transmitted spikes and the communication distance of each spike. Therefore, the hardware energy evaluation directly reflects the reduction in NoC traffic produced by the proposed compression method.

Neuron Mapping: For each neural network, neurons are mapped according to the configuration in Table 8. We use the open-source framework Dang et al. (2023) to determine the physical location of each neuron. The traveling distance is calculated using the Manhattan distance, assuming minimal routing. For compressed AER, spikes must travel through all PEs containing neurons whose AER packets are merged. Therefore, nearest-neighbor routing is applied to determine the traveling distance.

Evaluation Procedure: For each evaluated neural network, we first measure the spike activity before compression and record the number of spikes transmitted through the NoC. We then apply the proposed compression method and measure the resulting spike activity. For each spike, the corresponding NoC path and number of traversed hops are determined according to the network topology and source-destination mapping. The same neural networks, spike traces, and source-destination mappings are maintained across both the 2D and 3D evaluations, ensuring that the hardware energy evaluation reflects the reduction in NoC traffic produced by the proposed compression method under different physical topologies.