SpikeFit: Towards Optimal Deployment of Spiking Networks on Neuromorphic Hardware

Ivan Kartashov[†*]   **Mariia Pushkareva**[*]   **Iakov Karandashev**[*]

[†]HSE University
[*]Center of Optical Neural Technologies

✉ iakartashov@edu.hse.ru

Abstract

This paper introduces SpikeFit, a novel training method for Spiking Neural Networks (SNNs) that enables efficient inference on neuromorphic hardware, considering all its stringent requirements: the number of neurons and synapses that can fit on a single device, and lower bit-width representations (e.g., 4-bit, 8-bit). Unlike conventional compressing approaches that address only a subset of these requirements (limited numerical precision and limited number of neurons in the network), SpikeFit treats the allowed weights' discrete values themselves as learnable parameters co-optimized with the model, allowing for optimal Clusterization-Aware Training (CAT) of the model's weights at low precision (2-, 4-, or 8-bit) which results in higher network compression efficiency, as well as limiting the number of unique synaptic connections to a value required by neuromorphic processor. This joint optimization allows SpikeFit to find a discrete weight set aligned with hardware constraints, enabling the most complete deployment across a broader range of neuromorphic processors than existing methods of SNN compression support. Moreover, SpikeFit introduces a new hardware-friendly Fisher Spike Contrubution (FSC) pruning method showing the state-of-the-art performance. We demonstrate that for spiking neural networks constrained to only four unique synaptic weight values (M = 4), our SpikeFit method not only outperforms state-of-the-art SNNs compression methods and a conventional baselines combining extreme quantization schemes, clustering algorithms, but also meets a wider range of neuromorphic hardware requirements and provides the lowest energy use in experiments.

1   Introduction

Spiking Neural Networks (SNNs) are brain-inspired models that promise exceptional energy efficiency when run on specialized neuromorphic hardware (1). Neuromorphic processors have been developed to emulate the event-driven and massively parallel nature of biological neural systems, with prominent examples including Intel's Loihi (digital asynchronous many-core processor (2)), SpiNNaker (ARM-based massively parallel platform with event-routing (3)), BrainScaleS (mixed-signal accelerated analog wafer-scale system (5)), and mixed-signal chips such as DYNAP-SE2 (43). These platforms promise orders-of-magnitude reductions in energy consumption compared to von Neumann accelerators by co-locating memory and computation and processing information through sparse spikes. Nevertheless, deploying Spiking Neural Networks (SNNs) on current neuromorphic hardware faces stringent architectural and algorithmic constraints. In other words, a significant gap exists between SNNs in simulation and their deployment on physical hardware.

First, neuromorphic hardware offers only limited on-chip memory and connectivity, which restricts the number of neurons and synapses that can be implemented. As a result, networks must often be

reduced in size to fit a single device. Second, most platforms support only low-precision or discrete weight values (for example, 2-, 4-, or 8-bit representations), making it difficult to deploy models trained with continuous or high-precision parameters.

Moreover, several neuromorphic systems, designed for density and superior energy efficiency, restrict each synapse to a very small number of available states. For instance, studies on IBM TrueNorth processor (37) highlighted its use of just four integer values to represent synaptic weights (10). This is a fundamental constraint, reducing a synapse's state to a choice from a small, programmable look-up table.

Training SNNs to perform well under such extreme constraints is a major challenge. Therefore, further research is required on SNN compression methods that allow full deployment on the hardware.

This paper proposes **SpikeFit**, an efficient training methodology tailored for producing resource-friendly efficient spiking neural networks deployable on a wide range of neuromorphic hardware. Our method offers another way of reducing the spiking model size, its energy use, and complying with most hardware requirements. SpikeFit method effectively combine efficient, hardware-friendly clusterization, quantization and pruning schemes allowing for a rapid deployment of spiking networks even on the most demanding hardware (e.g. on IBM TrueNorth processor (37) requiring to represent SNN's synaptic weights using a fixed set of integer values). The method treats the M allowable discrete weight values not as fixed constants, but as learnable parameters. This can be viewed as a "smart" or adaptive clustering method that operates during training.

The core contributions of this work are:

   (i) Clusterization Aware Training (CAT) method finding optimal sets of discrete values to represent weights in low numerical precision where the M unique discrete weight values for each layer are learned end-to-end, allowing the model to discover its own optimal resource-friendly weight representation.

  (ii) Fisher Spike Contribution (FSC) Pruning method approximating the diagonal Fisher information of channel-wise gates in spiking network.

 (iii) A comparison focused on models with the same level of precision: our SpikeFit method against baselines: Quantization-Aware Training (QAT) plus clustering and Trained Ternary SNN baselines, versus recent state-of-the-art SNNs compressing approaches.

The key characteristics of SpikeFit are: a) The values in the codebook (the set of M discrete levels) are not fixed but shift to positions that are most advantageous for minimizing the global loss function b) The network's underlying latent weights begin to cluster around these learned codebook values as training progresses c) Optimal clustering technique (CAT) allows to reduce performance loss on post-hoc weight quantization step acting like a quantization regularization technique (making the network less sensetive to noise or drift) d) The method allows to represent synaptic weights in accordance to available number of unique synaptic states (e.g. 4 available synaptic weight states in IBM TrueNorth (37)) d) The Fisher Spike Contribution (FSC) Pruning method precisely removes less important parts of the spiking network according to the Fisher Information of both its spiking activity, and its effect on a global loss function.

2 Related Work

Model compression is essential for deploying deep neural networks on resource-constrained devices. Quantization and pruning remain key techniques reducing the precision of model weights, its size, and energy use.

Quantization in ANNs Quantization-Aware Training (QAT) is the standard method for high-performance quantized ANN models, outperforming Post-Training Quantization (PTQ) by simulating quantization in the training loop. Extreme cases include binary and ternary quantization allowing to achieve extreme model's compression via conversion into low numerical precision (1-bit (11), 2-bit (15; 16)).

Quantization in SNNs Quantizing SNNs is challenging due to their temporal dynamics and the non-differentiable nature of spiking. While one can convert a trained ANN to an SNN, subsequent quantization can amplify conversion errors. Directly training SNNs using surrogate gradients enables

standart QAT-like approaches, although these often focus on standard 4-bit or 8-bit quantization (19). Extreme cases like binary (20; 21; 27) and ternary SNNs use predefined, fixed value sets (e.g., $\{-1, 0, 1\}$ (28; 29)) to achieve extreme compression of the SNN model. The latter methods allow for a fast and complete deployment on a wide range of neuromorphic processors without the use of complicated training schemes, although they often lead to performance degradation (44).

Layer-wise Quantization vs Global Quantization Layer-wise quantization – i.e. tuning the fixed-point or reduced-precision format of each layer separately – reliably outperforms a single global quantization scale because different layers have very different numeric ranges. Moreover, neuromorphic hardware like Intel's Loihi chips is built for this flexibility – Loihi's synapses can each use 1–9 bits (2) – so quantizing per layer maps directly onto its architecture.

Pruning techniques Pruning in spiking neural networks (SNNs) can be broadly categorized into unstructured and structured approaches. Unstructured pruning removes individual weights or neurons based on local saliency metrics. Magnitude-based strategies (40) and fine-grained pruning frameworks (39) have demonstrated that extremely sparse connectivity can be achieved without substantial performance degradation. Similarly, biologically inspired schemes such as synaptic rewiring (24; 25) adaptively prune and regrow weak connections. While these methods provide high compression ratios and improved energy efficiency, their irregular sparsity patterns disrupt memory alignment, often hindering hardware-friendliness and requiring specialized hardware for efficient deployment.

On current neuromorphic processors, unstructured pruning yields little or no hardware benefit because synaptic storage, routing, and device allocation are implemented at coarse-grained levels rather than individual connections. For example, IBM TrueNorth (37) implements neurons and synapses in fixed neurosynaptic cores (256 neurons and 64k synapses per core) with event-driven routing tables. Eliminating scattered synaptic weights does not free entries in these tables or reduce reserved per-core synapse banks, making unstructured sparsity ineffective. Similarly, Intel Loihi (2) employs hierarchical routing and allocates its 16 MB of on-chip synaptic memory in block formats. Consequently, weight-level sparsification does not reclaim memory or routing resources unless pruning is performed in routing-aware, structured groups.

In contrast, structured pruning eliminates entire channels or kernels, thus preserving the network's computational regularity and compatibility with off-the-shelf hardware accelerators. The most impactful works on structured pruning combine principled pruning criteria with adaptive or automated strategies, and increasingly consider hardware constraints. The most efficient recent approaches are Spiking Channel Activity-based (SCA) pruning presented in (38) and Singular Value of Spike activity pruning criterion (SVS) presented in (36). However, both methods have issues. SCA-based pruning criterion exhibits a high dependency on inputs. SVS pruning criterion solves the latter problem effectively measuring spatio(-temporal) complexity but ignores the loss signal which can favor redundant patterns or background structures.

Compression of SNNs To further maximize the compression of SNNs recent works include combining pruning and layer-wise quantization. QP-SNN architecture presented in (36) achieves an incredible efficiency by integrating hardware-friendly SVS-based structured pruning technique with quantization technique and weight rescaling strategy applied beforehand. However, this method do not offer a full deployment on neuromorphic processors that limit the number of unique synaptic states (e.g. IBM TrueNorth (37) or Kaspersky Alt-AI (30)). It is important to also mention its quantization technique (ReScaW) applies rescaling strategy to weights only solving imbalanced weight distribution problem (i.e. long-tailed weight values) out of all weight representation problems affecting uniform quantization accuracy: large dynamic-range differences across layers or channels highlighted in (34), large loss-sensitivity of some groups of weights (33). Moreover, weight-only rescaling fix proposed by ReScaW ignore how quantized activations interact with quantized weights (although the joint distribution matters for end-to-end error), as well as ignoring the fact that scalar rounding treats each weight independently, not correlated weight patterns. For comparison, several works addressing similar challenges in LLM weight compression have proposed more effective rescaling strategies. For instance, the Activation-aware Weight Quantization (AWQ) (4) method introduces a simple rescaling scheme based on activation-aware patterns, which appears particularly relevant to our case and yield higher efficiency. In contrast, ReScaW requires manual selection of the gamma parameter based on data-specific insights, which complicates its application and, most importantly, it prevents leveraging activation-aware patterns to achieve state-of-the-art performance. We believe weight representation could be further improved and novel approaches could be proposed to investigate more

robust weight representation strategies allowing for better quantization regularization. Specifically, applying clustering algorithms to model's weights has been shown to improve the robustness of ANN models (42). This makes clustering a promising option for further maximizing SNN model compression, particularly given that clustering methods also align well with certain neuromorphic hardware requirements by limiting available synaptic states. Consequently, there is a clear need to explore new hardware-friendly efficient approaches for SNN compression.

3 Quantized and Clustered SNN Baseline

Baseline requirements We see the main focus of our contribution is to allow for an optimal and fast SNN compression that is available for a wide range of neuromorphic processors. Therefore, for baseline we take two different classic SNN compression pipelines that have the most complete deployment on the hardware observing requirements: a limited number of neurons in the neural network, a limited set of values (synaptic states) to represent weights, and a lower bit-width representations (e.g., 2-bit). Finally, our baselines are: Ternary Baseline and Clustered Baseline.

The LIF Spiking Neuron Model. We employ the widely used Leaky Integrate-and-Fire (LIF) neuron model across all SNN architectures. The continuous-time LIF dynamics describe the membrane potential $U_{\mathrm{mem}}(t)$ of an RC circuit:

$$\tau \frac{\mathrm{d}U_{\mathrm{mem}}(t)}{\mathrm{d}t} = -U_{\mathrm{mem}}(t) + R\,I_{\mathrm{in}}(t),$$

where $\tau = RC$ is the membrane time constant. Using forward Euler discretization with step Δt:

$$U[t+1] = U[t] + \Delta t\tau(-U[t] + R\,I_{\mathrm{in}}[t]).$$

For deep learning, we set $\Delta t = R = 1$ and absorb parameters into learnable weights W, yielding a discrete form with soft reset:

$$U[t+1] = \beta U[t] + W X[t+1] - \beta S[t]U_{\mathrm{thr}},$$

where $X[t]$ is the presynaptic spike train, $\beta = e^{-\Delta t/\tau}$ the decay rate, and $S[t] = \mathbf{1}(U[t] > U_{\mathrm{thr}})$ the output spike (41). The soft reset, scaled by β, preserves spike timing while enhancing training stability compared to a hard reset.

Implementation of LIF, Simulation, and ANN Conversion Protocols. In our experiments, we utilize the LIF neuron with a decay rate of $\beta = 0.5$ and a firing threshold of $U_{\mathrm{thr}}=1$. To implement SNN versions of ANN architectures ReLU activations are replaced with surrogate-gradient LIF neurons, pooling and classifiers are applied per time step via SeqToANNContainer from SpikingJelly Python Library (45), inputs are replicated across T steps by adding an extra input dimensions, and ResNet shortcuts use time-distributed projection with pretrained ANN weights remapped into the wrapped modules. To enable backpropagation through the non-differentiable spiking function, we employ an arctangent surrogate gradient.

Clustered Baseline One common approach of combining clustering and quantization techniques involves using Post-Training Quantization (PTQ) with weight clustering of a pre-trained, full-precision network using an algorithm like k-Means. However, a stronger baseline, which we use, first fine-tunes a model with Quantization-Aware Training (QAT) and then applies different clustering algorithms.

This baseline approach generates weights for the spiking architecture as follows: (i) train an FP32 CNN, (ii) fine-tune with QAT, (iii) cluster the resulting FP32 layer weights into M centroids (e.g., k-means, mini-batch k-means, uniform levels) and replace weights by their nearest centroid, (iv) convert to INT8 and briefly fine-tune, and (v) perform ANN→SNN conversion by dequantizing and loading the clustered/quantized weights into the target SNN. This preserves topology, reduces synaptic states and memory, and yields an SNN readily deployable on neuromorphic hardware.

Ternary Baseline Ternary quantization in Spiking Neural Networks assign weights to a discrete set of low-precision values (2-bit) with fixed set of 3 unique values (e.g., $\{-1, 0, 1\}$) observing hardware requirements listed in baseline requirements section. For this reason, we integrated Trained Ternary Quantization algorithm presented in (32) for our SNN architectures for further method evaluation.

4 SpikeFit Method

When deploying SNNs on recent neuromorphic devices researchers face many limitations on neural network size, its weights representation, energy use. This makes SNNs compressing techniques such as quantization and pruning a promising approach to comply with the limitations. Existing research describes different pruning and quantization schemes to comply with: a limited number of neurons in the neural network, a lower bit-width representations (e.g., 4-bit, 8-bit), limitations on energy use. However, listed limitations are a small subset of all requirements that neuromorphic systems list. For instance, the core architecture of the widely known neuromorphic chip IBM TrueNorth lays out in binary synapse connectivity and 4 axon types. Thus, it supports 4 various synaptic weight values and requires proposing a LUT of 4 unique values per neuron (7). Some other neuromorphic platforms also report a limitation of a fixed set of synaptic states: Kaspersky Alt-AI (30) (limits synaptic weight to 4 unique values), Darwin-3 (9) (use axon or weight indexes to encode topology and reduce the per-synapse memory footprint). Hybrid processors report similar requirements (e.g. Tianjic chip (8) uses a weight-index (codebook) approach in its FCore). Moreover, biologically plausible spiking networks (mostly used in tasks related to neuroscience field, computational simulations) have shown to implement synapses with a finite number of discrete weight states (e.g. 2-, 4-, 8- or 16-state synapses) that are sufficient (and sometimes necessary) for memory, learning, dynamics, and plasticity to behave in biologically realistic ways, while reducing weight-storage and compute cost (46; 48; 47) which is a very similar limitation that will require proposing a set of discrete values if the network weights were not directly trained. Therefore, all the latter cases require special techniques to comply with the restrictions. SpikeFit offers a novel method combining state-of-the-art quantization, pruning and clusterization schemes directly for spiking networks training to run inference on a wide range of processors achieving state-of-the-art performance and maintaining resource-friendly efficiency.

4.1 Clusterization Aware Training (CAT)

Inspired by Vector Quantization (VQ) techniques, popularized by the Vector-Quantized Variational Autoencoders (VQ-VAEs) (31) we propose Clusterization Aware Training (CAT) method for efficient compression of the spiking model, reducing weight numerical precision and overcoming the neuromorphic hardware requirement of using a fixed set of values to represent weights. CAT is a clusterization-aware training technique that learns the optimal set of M discrete values for the network weights, rather than imposing a fixed structure like most clusterization algorithms. This allows the network to adapt its weight representation to the specific task and architecture, build its own regularization scheme that yields the robust weight representation for post-hoc quantization along the way.

Learnable Codebooks and Latent Weights For each layer that uses Clusterization Aware Training (CAT), we maintain two sets of parameters:

1. **Latent Weights (W_{latent}):** A full-precision weight tensor, identical to weights in a standard neural network. These are updated via backpropagation and represent the "ideal" weights the network aims to learn.

2. **Codebook (C):** A small, learnable vector $C = (c_1, c_2, \ldots, c_M)$ of M floating-point values. This vector, unique to each layer, defines the entire set of discrete values that the final quantized weights can take. The codebook itself is a learnable parameter.

Weight Transformations and Gradient Flow The core idea is to map each latent weight to the closest value in the codebook during the forward pass, while ensuring gradients can flow back to both the latent weights and the codebook during the backward pass.

Forward Pass: Each latent weight w^{latent} is quantized to its nearest neighbor in the codebook C. The resulting weight, $w^{\text{quantized}}$, is used for the layer's operation (e.g., convolution).

$$k^* = \arg\min_{k \in \{1, \ldots, M\}} \|w^{\text{latent}} - c_k\|^2 \tag{1}$$

$$w^{\text{quantized}} = c_{k^*} \tag{2}$$

Backward Pass: The $\arg\min$ operation is non-differentiable. To overcome this, we use the Straight-Through Estimator (STE) (35). For the backward pass, the gradient is passed directly to the latent
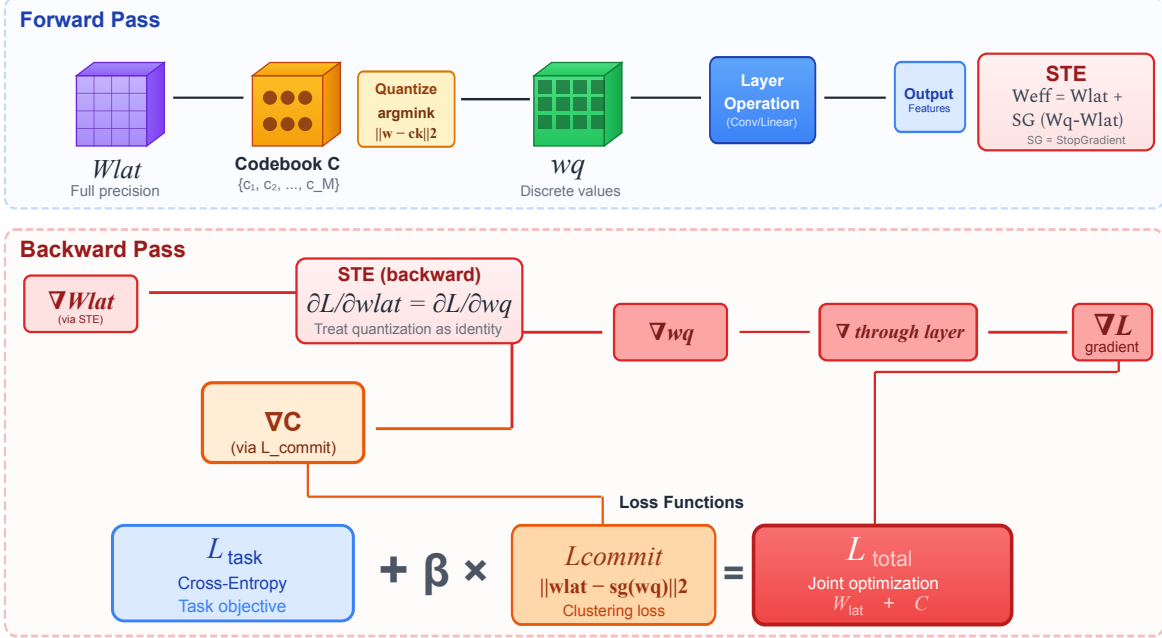
Figure 1: Overview of the Clusterization Aware Training (CAT) pipeline.

weights W_{latent} as if no quantization occurred. This is achieved with the following formulation for the effective weight W_{eff} used in computation:

$$w^{\text{eff}} = w^{\text{latent}} + \text{StopGradient}(w^{\text{quantized}} - w^{\text{latent}}), \qquad (3)$$

where StopGradient indicates that gradients are not propagated through the the term when computing this loss. This trick ensures that gradients with respect to the loss flow to W_{latent}. The codebook values c_k are part of the computation graph and are updated directly.

Composite Loss for Smart Clustering   Training is guided by a composite loss function. The first component is the standard task loss, $\mathcal{L}_{\text{task}}$ (e.g., Cross-Entropy). The second component is a **commitment loss**, $\mathcal{L}_{\text{commit}}$, which enables the "smart clustering" behavior. It penalizes the distance between the latent weights and their corresponding chosen codebook values:

$$\mathcal{L}_{\text{commit}} = \frac{1}{N_w}\left\| w^{\text{quantized}} - \text{StopGradient}(w^{\text{latent}}) \right\|_2^2, \qquad (4)$$

where N_w is the number of weights in the layer (e.g., a convolutional layer), and StopGradient indicates that gradients are not propagated through w^{latent} when computing this loss. This loss encourages each codebook entry c_k to move toward the centroid of the latent weights assigned to it. By stopping the gradient on w^{latent}, only the codebook C is updated during backpropagation. As a result, the codebook vectors c_k adapt to represent optimal cluster centers, and the latent weights W_{latent} are indirectly guided to cluster around them.

The final loss function is a weighted sum:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \beta \cdot \mathcal{L}_{\text{commit}} \qquad (5)$$

where β is a hyperparameter balancing the two objectives. By minimizing $\mathcal{L}_{\text{total}}$, we jointly optimize the latent weights and the codebooks.

Codebook Quantization   To reduce memory footprint and accelerate inference, we quantize the codebook into a low-bit integer representation.

For a codebook $\mathbf{C}$, we first compute the dynamic range and corresponding scaling factor:

$$s = \max\big(\max(\mathbf{C}) - \min(\mathbf{C}), 1\big).$$

6

Given a target bit-width b, the representable integer range is

$$\mathcal{R}_b = \left[-2^{b-1} + 1,\ 2^{b-1} - 1\right].$$

The quantized codebook is then obtained as

$$\hat{\mathbf{C}} = \mathrm{clip}\left(\mathrm{round}(\mathbf{C}s)\,,\ \min(\mathcal{R}_b),\ \max(\mathcal{R}_b)\right).$$

During inference, the quantized weights are reconstructed as $w^{\text{quantized}} = s\hat{c}_{k^*}$ where $k^* = \arg\min_{k \in \{1,\dots,M\}} \|w^{\text{latent}} - s\hat{c}_k\|^2$.

4.2 Fisher Spike Contribution (FSC) Pruning

Problem analysis. Pruning in spiking neural networks (SNNs) can be broadly categorized into unstructured and structured approaches. Unstructured pruning removes individual weights or neurons based on local saliency metrics. Magnitude-based strategies (40) and fine-grained pruning frameworks (39) have demonstrated that extremely sparse connectivity can be achieved without substantial performance degradation. Similarly, biologically inspired schemes such as synaptic rewiring (24; 25) adaptively prune and regrow weak connections. While these methods provide high compression ratios and improved energy efficiency, their irregular sparsity patterns disrupt memory alignment, often requiring specialized hardware for efficient deployment. Structured pruning has long leveraged curvature-aware criteria to minimize the loss increase after parameter removal. Classical methods such as Optimal Brain Damage (OBD) (22) and Optimal Brain Surgeon (OBS) (23) use (diagonal or exact) second-order information to estimate deletion of certain groups of parameters' impact. Similarly, first-order Taylor pruning (26) captures loss sensitivity, although might be brittle due to sign cancellations.

For spiking networks, surrogate-gradient learning (17) enables backpropagation through non-differentiable spike functions, which makes gradient-based saliencies feasible. Therefore, we came up with a novel pruning criterion motivated by desire to include computationally cheap loss alignment into the pruning scheme reinforced by the equivalence between the expected Hessian and the Fisher information under regularity conditions (12; 13; 14), and by the wish to exploit temporal coding in SNNs that static activity measures (e.g., Spiking Channel Activity (SCA) criterion (38), Singular-Value Saliency (SVS) criterion (36)) may overlook by averaging pruning criterion over time steps (36). Recent pruning works exploring gradient flow at initialization (e.g., SynFlow (18)) further support using label-aware sensitivity. Our proposed FSC criterion is a label-aware, curvature-aligned pruning criterion adapted to SNNs. FSC retains channels with high loss sensitivity integrated over time, which better preserves accuracy at a fixed pruning ratio.

Problem setup. Consider a spiking layer with output activations $y \in \mathbb{R}^{N \times T \times C \times H \times W}$ (batch N, time T, channels C, spatial $H \times W$). Let L denote the task loss computed on logits downstream of this layer. Define the backpropagated gradient tensor $\delta := \partial L / \partial y$ with the same shape as y. We write $\delta_{b,t,c,h,w}$ and $y_{b,t,c,h,w}$ for sample b, time t, channel c and spatial location (h, w).
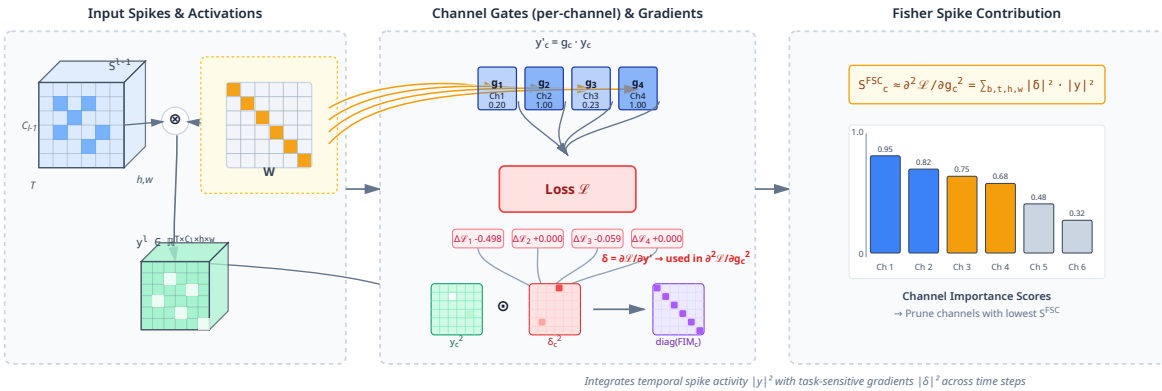

Figure 2: Overview of the Fisher Spike Contribution (FSC) pruning pipeline.

Channel-gate view and Fisher connection. Following the classic pipeline for estimation of parameters contribution to the global loss function in response to parameters deletion expressed by objective function incrementation mentioned in (22), let's introduce a multiplicative channel gate g_c acting on the output channel c:

$$y'_{b,t,c,h,w} = g_c\, y_{b,t,c,h,w}.$$

The objective function (the global loss function) could be approximated by a Taylor series. We would like to estimate the curvature property change in response to parameters deletion. Thus, for a small perturbation of g_c around 1, we estimate a second-order expansion of the loss the following way:

$$\Delta L \approx (g_c - 1)\frac{\partial L}{\partial g_c} + \frac{1}{2}(g_c - 1)^2\frac{\partial^2 L}{\partial g_c^2}.$$

By the chain rule,

$$\frac{\partial L}{\partial g_c} = \sum_{b,t,h,w} \delta_{b,t,c,h,w}\, y_{b,t,c,h,w}.$$

Since computing and storing the Hessian matrices directly is prohibitive, we will focus on efficient ways to approximate it. Under standard conditions for negative log-likelihood losses, the expected Hessian equals the Fisher information matrix (12). In order to prune individual channels, it is necessary and sufficient to have a scalar importance per parameter. The (diagonal) Fisher for gate g_c can be approximated as

$$F_{cc} \approx \mathbb{E}\left[\left(\frac{\partial L}{\partial g_c}\right)^2\right] = \mathbb{E}\left[\left(\sum_{b,t,h,w} \delta_{b,t,c,h,w}\, y_{b,t,c,h,w}\right)^2\right].$$

Expanding the square and neglecting cross-terms (or upper-bounding by Cauchy–Schwarz), a computable and robust surrogate is

$$F_{cc} \approx \kappa\, \mathbb{E}\left[\sum_{b,t,h,w} \delta^2_{b,t,c,h,w}\, y^2_{b,t,c,h,w}\right],$$

for some constant κ depending on the number of terms and correlations. This leads directly to our FSC saliency.

FSC saliency. We define the Fisher Spike Contribution score for channel c as

$$S_c^{\text{FSC}} = \frac{1}{N}\sum_{b=1}^{N}\sum_{t=1}^{T}\sum_{h=1}^{H}\sum_{w=1}^{W}\left(\delta_{b,t,c,h,w}\right)^2\left(y_{b,t,c,h,w}\right)^2. \tag{6}$$

Intuitively, S_c^{FSC} is large only when a channel spikes (large $|y|$) exactly where/when the loss is sensitive (large $|\delta|$).

5 Experiment

SNN Deployment Evaluation There are many efficient SNN compression methods introduced by other authors allowing to successfully deploy spiking models on specialized hardware. However, many of the works lack clear and objective evaluation methods as most experiments include task performance metrics only (accuracy, F1, and similar) showing whether the model handles the task well, but not representing the degree of improvement of the SNN model's performance on neuromorphic hardware. To estimate a success of both compression and deployment of SNN models we propose a novel metric and include it in our experiments.

DeployRatio: A Composite Deployability Metric We propose **DeployRatio (DR)**, a simple composite metric to quantify the deployability of Spiking Neural Networks (SNNs) by jointly capturing task performance, latency, energy, and model footprint:

$$\text{DR}_x = \frac{\text{Perf}_x}{L\,E\,S}, \quad x \in \{\text{acc}, \text{f1}\}, \quad \text{Perf}_{\text{acc}} = \text{Accuracy}[\%]100, \ \text{Perf}_{\text{f1}} = \text{F1}[\%]100.$$

Here L is inference latency (s/input), E energy per inference (mJ), and S model size (MB). A higher DR indicates a model that is accurate, fast, energy-efficient, and compact. The components are:

- **Task performance:** $\mathrm{Perf}_x \in [0, 1]$ from accuracy or macro-F1 (%/100).

- **Latency:** $L = \frac{1}{B}\sum_{b=1}^{B}\Delta t_b$, mean inference time over B batches.

- **Model size:** $S = \frac{P\,b}{8\times 10^6}$ MB, with parameter count P and precision b (bits/weight).

- **Energy:** E (mJ) estimated from operation counts and per-operation energy constants.

DR is dimensionless and monotonic—higher with better performance and lower with increased latency, energy, or size. $\mathrm{DR_{acc}}$ and $\mathrm{DR_{f1}}$ target different objectives (balanced vs. imbalanced classes).

5.1 Analysis of Deployment Results

Table 1: Performance comparison of compression methods. Accuracy is reported as **mean ± std** over five runs. **DR (DeployRatio)** indicates the deployment cost-performance ratio. Bold blue indicates our proposed **SpikeFit** results.

Dataset	Network	Method	Precision	Size (MB)	Energy (mJ)	Timestep	Accuracy (%)	F1 (%)	DR (%)
CIFAR-10	VGG-16	Clustered Baseline	8	14.723	**0.004**	4	77.57±0.49	77.54±0.44	68.649
		Ternary Baseline	8	14.723	0.014	4	76.244±0.12	76.51±0.04	22.051
		QP-SNN	8	14.723	0.032	4	86.58±0.32	86.91±0.32	10.883
		SpikeFit (ours)	8	14.725	**0.004**	4	**89.14±0.82**	**89.99±0.84**	**89.640**

The evaluation results across all experiments (Table 1) prove a consistent result on SpikeFit-optimized architecture (VGG-16) performing best and showing state-of-the-art performance in the widely-known benchmark of multi-label classification on CIFAR-10 dataset. Moreover, SpikeFit allows for the most complete deployment of the SNN models by considering the limitation on unique synaptic states number required by some neuromorphic processors that focus on extreme energy efficiency (e.g. IBM TrueNorth) and showing the best performance according to the proposed DeployRatio (DR) metric.

5.2 Evaluation of FSC pruning

Table 2: Comparison of pruning and compression methods across networks and strategies. Bold blue indicates our proposed FSC method.

Method	Network	Pruning	Size (MB)	Energy (mJ)	Acc (%)	Prec (%)	Rec (%)	F1 (%)
QP-SNN	VGG-16	SVS (36)	14.723	**0.03**	86.656±0.116	86.842±0.107	86.656±0.116	86.628±0.118
		SCA (38)	14.723	**0.03**	83.275±0.37	84.927±0.27	83.278±0.37	83.373±0.39
		FSC (ours)	14.725	**0.03**	**87.546±0.059**	**87.621±0.073**	**87.546±0.059**	**87.553±0.065**

Why FSC performs best in our experiments. Empirically, FSC consistently outperforms activity- or rank-based criteria (e.g., SVS) across sparsity regimes in our QP-SNN setting. We hypothesize the reason is the **loss alignment**: FSC is directly proportional to (a diagonal approximation of) the Fisher information F_{cc} of a channel gate, hence it targets channels whose removal minimally increases expected loss, unlike SVS which is label-agnostic and rather measures representational complexity, but not task relevance. Two channels with similar SVS score can have vastly different impacts on the loss: one may spike during decisive moments (large $|\delta|$), while another spikes in background regions (small $|\delta|$).

6 Conclusion

SpikeFit training methodology demonstrated that combining a novel Clusterization-Aware-Training (CAT) method with quantization and the Fischer Spike Contribution (FSC) structured pruning scheme might be an effective way to deploy hardware-friendly SNNs that operate fast, with low-energy use at state-of-the-art performance. Hopefully, our contribution will motivate researchers to introduce methods offering even better energy efficiency and performance on a wide range of neuromoprhic processors.

References

[1] Schuman, C.D., Kulkarni, S.R., Parsa, M., Mitchell, J.P., Date, P., Kay, B.: Opportunities for neuromorphic computing algorithms and applications. Nat. Comput. Sci. **2**(1), 10–19 (2022). `https://doi.org/10.1038/s43588-021-00184-y`

[2] Davies, M., et al.: Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **38**(1), 82–99 (2018).

[3] Furber, S., Galluppi, F., Temple, S., Plana, L.: The SpiNNaker project. *Proceedings of the IEEE* **102**(5), 652–665 (2014). doi:10.1109/JPROC.2014.2304638.

[4] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. 2025. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. GetMobile: Mobile Comp. and Comm. 28, 4 (December 2024), 12–17. https://doi.org/10.1145/3714983.3714987.

[5] Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., Millner, S.: A wafer-scale neuromorphic hardware system for large-scale neural modeling. *ISCAS 2010 – IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, 1947–1950 (2010). doi:10.1109/ISCAS.2010.5536970.

[6] Moradi, S., Ning, Q., Stefanini, F., Indiveri, G.: A scalable multi-core architecture with heterogeneous memory structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs). *CoRR* **abs/1708.04198** (2017). arXiv:1708.04198.

[7] [1]Moran, S., Gaonkar, B., Whitehead, W., Wolk, A., Macyszyn, L., and Iyer, S. S., "Deep learning for medical image segmentation - using the IBM TrueNorth neurosynaptic system", in <i>Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications</i>, 2018, vol. 10579, Art. no. 1057915. doi:10.1117/12.2286419.

[8] Pei, J., Deng, L., Song, S. et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. Nature 572, 106–111 (2019). https://doi.org/10.1038/s41586-019-1424-8.

[9] Ma D, Jin X, Sun S, Li Y, Wu X, Hu Y, Yang F, Tang H, Zhu X, Lin P, Pan G. Darwin3: a large-scale neuromorphic chip with a novel ISA and on-chip learning. Natl Sci Rev. 2024 Mar 18;11(5):nwae102. doi: 10.1093/nsr/nwae102. PMID: 38689713; PMCID: PMC11060491.

[10] Esser, S.K., et al.: Convolutional networks for fast, energy-efficient neuromorphic computing. Proc. Natl. Acad. Sci. U.S.A. **113**(41), 11441–11446 (2016). `https://doi.org/10.1073/pnas.1604850113`

[11] Wang, H., Ma, S., Dong, L., Huang, S., Wang, H., Ma, L., Yang, F., Wang, R., Wu, Y., Wei, F.: BitNet: Scaling 1-bit transformers for large language models. arXiv:2310.11453 (2023). `https://arxiv.org/abs/2310.11453`

[12] Fisher Information and Natural Gradient Learning in Random Deep Networks. Shun-ichi Amari, Ryo Karakida, Masafumi Oizumi Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics, PMLR 89:694-702, 2019.

[13] WoodFisher: Efficient Second-Order Approximation for Neural Network Compression. Sidak Pal Singh, Dan Alistarh. NeurIPS, 2020. arXiv:2004.14340.

[14] New Insights and Perspectives on the Natural Gradient Method James Martens. 21(146):176, 2020. Journal of Machine Learning Research.

[15] Ma, S., Wang, H., Ma, L., Wang, L., Wang, W., Huang, S., Dong, L., Wang, R., Xue, J., Wei, F.: The era of 1-bit LLMs: All large language models are in 1.58 bits. arXiv:2402.17764 (2024). `https://arxiv.org/abs/2402.17764`

[16] Jain, H., Agarwal, A., Shridhar, K., Kleyko, D.: End to end binarized neural networks for text classification. arXiv:2010.05223 (2020). `https://arxiv.org/abs/2010.05223`

[17] Surrogate Gradient Learning in Spiking Neural Networks, 2019. Emre O. Neftci, Hesham Mostafa, Friedemann Zenke. arXiv:1901.09948.

[18] Pruning neural networks without any data by iteratively conserving synaptic flow, NeurIPS 2020. Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, Surya Ganguli. arXiv:2006.05467.

[19] Lui, H.W., Neftci, E.: Hessian aware quantization of spiking neural networks. arXiv:2104.14117 (2021). https://arxiv.org/abs/2104.14117

[20] Eshraghian, J.K., Lu, W.D.: The fine line between dead neurons and sparsity in binarized spiking neural networks. arXiv:2201.11915 (2022). https://arxiv.org/abs/2201.11915

[21] Pai-Yu Tan, Po-Yao Chuang, Yen-Ting Lin, Cheng-Wen Wu, Juin-Ming Lu: A Power-Efficient Binary-Weight Spiking Neural Network Architecture for Real-Time Object Classification. https://doi.org/10.48550/arXiv.2003.06310

[22] Yann LeCun, John S. Denker, Sara A. Solla. Optimal Brain Damage, 1989. Advances in Neural Information Processing Systems 2 (NIPS 1989). https://doi.org/10.48550/arXiv.2003.06310

[23] B. Hassibi, D. G. Stork and G. J. Wolff, "Optimal Brain Surgeon and general network pruning," IEEE International Conference on Neural Networks, San Francisco, CA, USA, 1993, pp. 293-299 vol.1, doi: 10.1109/ICNN.1993.298572.

[24] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. arXiv preprint arXiv:1711.05136, 2017.

[25] Yanqi Chen, Zhaofei Yu, Wei Fang, Zhengyu Ma, Tiejun Huang, and Yonghong Tian. State transition of dendritic spines improves learning of sparse spiking neural networks. In International Conference on Machine Learning, pp. 3701–3715. PMLR, 2022.

[26] Pruning Convolutional Neural Networks for Resource Efficient Inference, 2016. Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz. arXiv:1611.06440.

[27] Shymyrbay, A., Fouda, M.E., Eltawil, A.: Low precision quantization-aware training in spiking neural networks with differentiable quantization function. arXiv:2305.19295 (2023). https://arxiv.org/abs/2305.19295

[28] Sun, C., Chen, Q., Fu, Y., Li, L.: Deep spiking neural network with ternary spikes. In: *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 251–254 (2022). doi:10.1109/BioCAS54905.2022.9948581.

[29] Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., Modha, D.: A low power, fully event-based gesture recognition system. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7388–7397 (2017). doi:10.1109/CVPR.2017.781.

[30] Neuromorphic Processor "Altai" for Energy-Efficient Computing, 2020. GRISHANOV, ZVEREV, IPATOV, KANGLER, KATOMIN, KODENKO, KOSTRITSYN, MAKAROV, MAMYCHEV, PAVLOV, PANCHENKO, POLSTYANKIN. 96. 531-538. 10.22184/1993-8578.2020.13.3s.531.538.

[31] Neural Discrete Representation Learning, 2018. Aaron van den Oord, Oriol Vinyals, Koray Kavukcuoglu. arXiv:1711.00937.

[32] Trained Ternary Quantization, 2016. Chenzhuo Zhu, Song Han, Huizi Mao, William J. Dally. arXiv:1612.01064.

[33] Z. Dong, Z. Yao, A. Gholami, M. Mahoney and K. Keutzer, "HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision," in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South), 2019, pp. 293-302, doi: 10.1109/ICCV.2019.00038.

[34] Towards Superior Quantization Accuracy: A Layer-sensitive Approach, 2025. Feng Zhang, Yanbin Liu, Weihua Li, Jie Lv, Xiaodan Wang, Quan Bai. https://doi.org/10.48550/arXiv.2503.06518

[35] Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013).

[36] Wei, W., Zhang, M., Zhou, Z., Belatreche, A., Shan, Y., Liang, Y., Cao, H., Zhang, J., Yang, Y.: QP-SNN: Quantized and pruned spiking neural networks. In: *The Thirteenth International Conference on Learning Representations* (2025). `https://openreview.net/forum?id=MiPyle6Jef`

[37] F. Akopyan et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 10, pp. 1537-1557, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[38] Yaxin Li, Qi Xu, Jiangrong Shen, Hongming Xu, Long Chen, and Gang Pan. 2024. Towards efficient deep spiking neural networks construction with spiking activity based pruning. In Proceedings of the 41st International Conference on Machine Learning (ICML'24), Vol. 235. JMLR.org, Article 1168, 29063–29073.

[39] Xinyu Shi, Jianhao Ding, Zecheng Hao, and Zhaofei Yu. Towards energy efficient spiking neural networks: An unstructured pruning framework. In The Twelfth International Conference on Learning Representations, 2024.

[40] Hang Yin, John Boaz Lee, Xiangnan Kong, Thomas Hartvigsen, and Sihong Xie. Energy-efficient models for high-dimensional spike train classification using sparse spiking neural networks. In Proceedings of the 27th ACM SIGKDD conference on knowledge discovery data mining, pp. 2017–2025, 2021.
)

[41] Eshraghian, J.K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D.S., Lu, W.D.: Training spiking neural networks using lessons from deep learning. Proc. IEEE. **111**(9), 1100–1127 (2023). `https://doi.org/10.1109/JPROC.2023.3294285`

[42] Rethinking Clustering for Robustness, 2020. Motasem Alfarra, Juan C. Pérez, Adel Bibi, Ali Thabet, Pablo Arbeláez, Bernard Ghanem. arXiv:2006.07682.

[43] DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor, 2023. Ole Richter, Chenxi Wu, Adrian M. Whatley, German Köstinger, Carsten Nielsen, Ning Qiao, Giacomo Indiveri. arXiv:2310.00564.

[44] Low Precision Quantization-aware Training in Spiking Neural Networks with Differentiable Quantization Function. 2023. Ayan Shymyrbay, Mohammed E. Fouda, Ahmed Eltawil. arXiv:2305.19295.

[45] SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence, 2023. Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, Yonghong Tian. arXiv:2310.16620.

[46] Barrett AB, van Rossum MCW (2008) Optimal Learning Rules for Discrete Synapses. PLOS Computational Biology 4(11): e1000230. https://doi.org/10.1371/journal.pcbi.1000230.

[47] Dasbach S, Tetzlaff T, Diesmann M, Senk J. Dynamical Characteristics of Recurrent Neuronal Networks Are Robust Against Low Synaptic Weight Resolution. Front Neurosci. 2021 Dec 24;15:757790. doi: 10.3389/fnins.2021.757790. PMID: 35002599; PMCID: PMC8740282.

[48] Walt Woods, Jens Burger, and Christof Teuscher. 2015. Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware. IEEE Trans. Nanotechnol. 14, 6 (Nov. 2015), 945–953. https://doi.org/10.1109/TNANO.2015.2449835.

7 Ablation Study

To further prove the effectiveness of SpikeFit training methodology, we conduct extensive ablation studies. We perform thorough ablation experiments to validate the effectiveness of the proposed CAT weight clusterization strategy and FSC-based pruning criterion. All ablation experiments are conducted on the CIFAR-10 dataset using VGG-16 architecture.

We compare the performance of quantized SNNs (not involve pruning process) with different M settings to evaluate performance on a really small codebook. As depicted in Table 3, the quantized (clustered) SNN using M=2 (8-bit weight clustered to 2 unique values) outperforms state-of-the-art QP-SNN method combining ReScaW quantization strategy with SVS pruning scheme storing weights in 8-bit precision. After increasing the size of the codebook of CAT by setting M=4 the performance is significantly improved by 2.44% providing even better results showing superior performance and surpassing ReScaW quantization scheme results (without pruning). This proves CAT effectivness when compared to state-of-the-art ReScaW quantization method. (36)

Table 3: Ablation study on VGG-16 SNN architecture on the effectiveness of proposed CAT method.

Dataset	Network	Method	Precision	Size (MB)	Timestep	Accuracy (%)	F1 (%)
CIFAR-10	VGG-16	Full-precision	32	58.91	4	**90.97**	**90.92**
		ReScaW (36)	8	14.72	4	89.14	89.73
		QP-SNN (36)	8	14.72	4	86.94	86.92
		CAT (M=2)	8	14.72	4	87.56	85.24
		CAT (M=4)	8	14.72	4	**90.00**	**89.99**

8 Experiment Setup

8.1 Training and Optimization Details

We summarize the training hyperparameters for each dataset in Table 1, including time step, image resolution, optimizer, and other factors.

- **Optimizers and Schedulers:** AdamW with CosineAnnealingLR for all stages; weight decay 1×10^{-5}; $\eta_{\min} = 0$; $T_{\max}$ equals the stage's epoch count.
- **Loss Functions:** Cross-entropy for task loss. For SpikeFit, an additional codebook commitment loss is used, scaled by $\beta_{\text{commit}} = 0.5$.
- **Pipeline schedule:**
  - FP32 stage: 300 epochs, initial LR 1×10^{-2}.
  - ReScaW (SNN) stage: 150 epochs fine-tuning, initial LR 1×10^{-4}, $T=4$.
  - CAT stage: 150 epochs fine-tuning, initial LR 1×10^{-4}, $T=4$, $M=4$, codebook quantization=True, bias=False.
- **SNN Parameters:** LIF neuron with $\tau=0.5$ (a.k.a., β) and threshold 1.0; $T=4$ timesteps.
- **Number of Unique Values (M):** $M=4$ for CAT codebooks.
- **Dataset & Resolution:** CIFAR-10 at 32×32.
- **Batch Size & Workers:** Batch size 1256; 8 data-loader workers.
- **Data Augmentation (CIFAR-10):** RandomCrop(32, padding=4), RandomHorizontalFlip, CIFAR10Policy (AutoAugment); normalization mean $(0.4914, 0.4822, 0.4465)$, std $(0.2023, 0.1994, 0.2010)$.
- **Pruning (QP-SNN):** prune ratio 0.3; SVS limit 5; SVD rank $k=3$; latency/energy trade-offs $\lambda_{\text{lat}}=0.2$, $\beta_{\text{energy}}=0.1$;
- **Evaluation Protocol:** Each configuration evaluated over 5 validation repeats.
- **Deployment/Proxyf Metrics (DR settings):** Hardware energy mapping auto-selected; MAC energy 4.6 pJ, accumulate energy 0.9 pJ; spike rate 0.2; latency estimated over 3 batches.