SPIKE REGULARIZATION AND CURRICULUM BASED DIRECT TRAINING OF SPIKING NEURAL NETWORKS FOR ENERGY-EFFICIENT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking neural networks emit binary events that can be exploited for higher energy efficiency on neuromorphic hardware. A spiking neuron that does not fire costs no energy, and hence, spike count is an important objective to optimize. We propose SRSC (Spike Regularization with Spike Curriculum) that applies a time-weighted penalty on every layer's spike tensor, held negligible until the validation accuracy crosses a threshold, and then ramped geometrically to a ceiling. This does not incur additional parameters or architectural changes, and only requires the sparsity ceiling to be tuned to obtain the desired accuracy-sparsity trade-off. We validate the proposed method on VGG, ResNet, and transformer-based architectures across CIFAR-10, CIFAR-100, DVS-CIFAR10, and ImageNet datasets. SRSC matches state-of-the-art accuracy ($\pm 0.5\%$ for ResNet-19 on CIFAR-10, $\pm 0.75\%$ for VGGSNN on DVS-CIFAR10), while reducing the firing rate by 49% and 44%, and synaptic operations (SOPs) by 24% and 16%, respectively, relative to state-of-the-art methods. Code is available at https://anonymous.4open.science/r/SRSC-BD6F

1 INTRODUCTION

Spiking neural networks (or SNNs) carry information in binary spiking events instead of continuous activations. On neuromorphic hardware (Akopyan et al., 2015; Davies et al., 2018) that consumes no dynamic energy in the absence of a spike or synaptic operation, inference energy scales with the number of spikes emitted by the network, making spike count a critical optimization objective during training. Deep SNNs are typically trained using the following methods. ANN-to-SNN conversion trains a conventional ANN and transfers its weights to a spiking network. QCFS-based conversion framework (Bu et al., 2022) assigns layer-wise thresholds so that each layer's firing rate approximates the source activation. Direct training instead optimizes the spiking network itself, replacing the derivative of the Heaviside step function with a surrogate gradient (Neftci et al., 2019). Temporal Efficient Training (TET) (Deng et al., 2022) is one of the state-of-the-art methods for training deep rate-coded SNNs and achieves competitive accuracy using fewer inference timesteps.

Direct training of SNNs has the promise of providing competitive accuracy at reduced spiking activity and timesteps compared to conversion based approaches. However, we found that TET-based direct training, with unit threshold fixed a priori for all layers, yielded SNNs that incurred higher spiking activity for comparable or superior accuracy at reduced timesteps (see Table 9 in Appendix B). Using a learnable per-layer threshold resulted in a further increase in spiking activity due to TET's per-timestep loss that encourages early firing. We also observed higher spiking activity with standard direct training (Wu et al., 2018), as shown in Table 9. Hence, this behavior is not limited to a specific SNN training method, and indicates that temporal dynamics and spike efficiency are not aligned if accuracy is the only optimization objective during training.

We propose **SRSC** (Spike Regularization with Spike Curriculum) based direct training, which penalizes the spike tensor of every spiking layer, weighted linearly in time so that late spikes cost more than early ones. The spike regularization coefficient is held negligible until the validation accuracy crosses a threshold or training crosses a certain number of epochs, and then ramped up geometrically to a ceiling. Our results show that SRSC consistently yields favorable trade-off between accuracy and spiking activity across convolutional and Transformer-based SNN architectures on event-based

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.

and static vision datasets including ImageNet. We use the same spike regularization coefficient to optimize the spiking activity uniformly for all layers. Previous work on foundational model pruning suggests that regularizing each layer according to its sensitivity yields better overall efficiency rather than using a uniform optimization strategy across layers (Yin et al., 2024a). We incorporate this by ranking the spiking layers according to their sensitivity and spiking activity, and suitably adapting the regularization coefficient. We find that layer-wise spike regularization leaves the accuracy–spike-suppression trade-off essentially unchanged from a uniform coefficient (see Section 5.5). The key contributions of our work are as follows.

1. We present SRSC, which introduces a time-weighted penalty on the spike tensors to obtain competitive accuracy with reduced spiking activity.
2. We validate SRSC across vision datasets such as CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10, on SNN architectures including ResNet-18, ResNet-19, SEW-ResNet-34, VGGSNN, and Spikeformer (Li et al., 2022).
3. We estimate and report the energy efficiency benefits obtained by the proposed approach using a hardware architectural simulator of an SNN accelerator in Section 5.4.

2 Related Work

Direct training of SNNs. Direct training treats rate-coded SNN as a recurrent network unrolled over T timesteps and applies backpropagation through time, substituting a smooth surrogate for the derivative of the Heaviside step function (Neftci et al., 2019). Work in this line has largely pursued accuracy: TET (Deng et al., 2022) reweights the loss across timesteps, and several methods make neuron parameters learnable: DIET-SNN (Rathi & Roy, 2020) optimizes the leak and the threshold jointly with weights, and LT-SNN (Hassan et al., 2024) learns per-layer thresholds directly.

ANN-to-SNN conversion. ANN-to-SNN conversion trains an ANN and transfers its weights to a spiking network, setting each layer's firing threshold so that its spike rate approximates the ANN's activations. QCFS (Bu et al., 2022) does this with the quantize-clip-floor-shift activation in place of ReLU, which drives the expected conversion error to zero and reaches competitive accuracy at low latency.

Reducing spike activity. Methods that reduce spike activity generally fall into two branches. The larger branch acts indirectly on parameters such as weights or normalization scales, and is dominated by weight pruning. Classical methods typically rely on saliency and magnitude (LeCun et al., 1989; Han et al., 2016). Building on these approaches, Grad R (Chen et al., 2021) learns connectivity and weights together through gradient rewiring. STDS (Chen et al., 2022) prunes by re-parameterizing weights through a nonlinear spine-size mapping. UniSparse's own weight-pruning component (Shi et al., 2026) adapts the soft-threshold reparameterization of Kusupati et al. (2020), updating the threshold directly from the energy constraint rather than STDS's heuristic schedule. OWL (Yin et al., 2024a) shows that allocating sparsity non-uniformly across layers, in proportion to a per-layer outlier statistic, substantially outperforms uniform allocation.

Related methods regulate other proxies for firing: UniSparse (Shi et al., 2026) constrains the affine parameters of batch normalization, observing that they serve as an effective, fixed-strength threshold for the subsequent spiking layer, while Sakemi et al. (2023) penalize the membrane potential under time-to-first-spike coding.

The second body of work optimizes the spike tensors directly, which determine the effective number of synaptic operations on neuromorphic hardware. Allred et al. (2020) add a spike count to the classification loss to achieve better accuracy with reduced spike count. Pellegrini et al. (2021) apply an L_2 penalty to each layer's spikes, while Suetake et al. (2023) shape the penalty to match the synaptic-operation count. Yan et al. (2022) combine spike penalty with weight sparsity and Deng et al. (2021) pair an ADMM pruning and quantization pipeline with a penalty on spike events. We extend this second branch by implementing the spike-penalty differently; while such penalties are typically held constant for the whole of training, SRSC schedules it during the course of training.

3 PRELIMINARIES

3.1 SPIKING NEURON MODEL

A spiking neuron typically belongs to one of two types: the leaky integrate-and-fire (LIF) neuron, which decays its membrane potential between inputs, and the integrate-and-fire (IF) neuron, which does not. For a neuron in layer ℓ receiving input current $x[t]$ at timestep t, with membrane potential $u[t]$ before charging and $v[t]$ after, the LIF neuron charges, fires, and resets as

$$\begin{aligned} v[t] &= u[t] + \tfrac{1}{\tau}\big(x[t] - (u[t] - u_{\text{rest}})\big) && \text{(charge)}, \\ s[t] &= H\Big(v[t] - \theta^{(\ell)}\Big) && \text{(fire)}, \\ u[t{+}1] &= s[t]\, u_{\text{rest}} + (1 - s[t])\, v[t] && \text{(reset)}, \end{aligned} \tag{1}$$

where $H(\cdot)$ is the Heaviside step function, $s[t] \in \{0, 1\}$ indicates a spike, and τ is the LIF neuron's membrane time constant. The IF neuron follows the same equation with the leak term removed, i.e. $v[t] = u[t] + x[t]$. We set $\tau = 2$ and $u_{\text{rest}} = 0$, and simulate for T timesteps. The firing threshold $\theta^{(\ell)}$ is fixed at 1 for all layers. The learnable-threshold variant we examine in Section 3.3 makes $\theta^{(\ell)}$ a trainable parameter per layer. Since H has zero derivative almost everywhere, we replace $\partial s/\partial v$ with a surrogate in the backward pass (Neftci et al., 2019).

Let $O(t)$ denote the output logits at timestep t and y the label. Standard direct training applies the cross-entropy loss to the mean output over the simulation window, whereas TET (Deng et al., 2022) applies it at each timestep and averages the resulting losses:

$$\mathcal{L}_{\text{SDT}} = \mathcal{L}_{\text{CE}}\Big(\tfrac{1}{T}\textstyle\sum_{t=1}^{T} O(t),\, y\Big), \qquad \mathcal{L}_{\text{TET}} = \tfrac{1}{T}\textstyle\sum_{t=1}^{T} \mathcal{L}_{\text{CE}}(O(t),\, y)\,. \tag{2}$$

Averaging the outputs before the loss allows a confident late timestep to compensate for an uninformative early one. Averaging the losses removes this compensation.

3.2 METRICS

Network Average Spikes (NAS) is a measure of spiking activity that estimates the mean number of spikes a neuron emits over the simulation window, averaged across layers. For a validation set $\mathcal{D}$ and a network of L spiking layers,

$$\text{NAS} = \frac{1}{L}\sum_{\ell=1}^{L} \frac{1}{|\mathcal{D}|\, N_\ell} \sum_{b\in\mathcal{D}} \sum_{t=1}^{T} \sum_{n=1}^{N_\ell} s_{b,t,n}^{(\ell)}. \tag{3}$$

NAS lies in $[0, T]$, since a neuron fires at most once per timestep. For example, a value of 0.40 at $T = 2$ corresponds to a per-timestep firing rate of 20%.

Firing Rate, following Shi et al. (2026), is the fraction of neuron-timesteps at which a spike is emitted, pooled over all L spiking layers and the validation set:

$$\text{FR} = \frac{\sum_{\ell=1}^{L} \sum_{b\in\mathcal{D}} \sum_{t=1}^{T} \sum_{n=1}^{N_\ell} s_{b,t,n}^{(\ell)}}{\sum_{\ell=1}^{L} |\mathcal{D}|\, T\, N_\ell}. \tag{4}$$

Unlike NAS, which normalizes each layer by its own neuron count before averaging layers equally, Firing Rate pools spike counts and neuron-timestep counts across all layers before dividing, so larger layers contribute proportionally more to the reported value.

Spike sparsity is the fraction of neurons that emit no spike over all T timesteps for a given input, pooled over all spiking layers and the validation set:

$$\text{SS} = \frac{\sum_{\ell=1}^{L} \sum_{b\in\mathcal{D}} \sum_{n=1}^{N_\ell} \mathbb{1}\Big[\sum_{t=1}^{T} s_{b,t,n}^{(\ell)} = 0\Big]}{\sum_{\ell=1}^{L} |\mathcal{D}|\, N_\ell}. \tag{5}$$

Whereas NAS and Firing Rate measure how much the network fires, spike sparsity counts neurons that remain silent for an entire input.

Weight sparsity measures the fraction of network weights reduced to zero.

Synaptic Operations (SOPs), following Shi et al. (2026), is the operation count that dominates compute energy on spike-driven neuromorphic hardware. For every convolutional or fully connected layer fed by a binary spike tensor, we count one operation per (nonzero weight, firing spike) pair. We sum these operations across all neurons, network layers, and T timesteps, reporting the average count per image. The stem layer is excluded, since its input is the raw image rather than a spike tensor.

3.3 Problem analysis

We motivate the method in Section 4 with two observations.

Direct training fires more than conversion. We found that TET-based direct training offers comparable or better accuracy than QCFS's conversion-based approach, but at a higher NAS: on ResNet-19/CIFAR-10 at $T = 2$, TET reached 95.21% accuracy at NAS 0.316 against QCFS's 87.35% at NAS 0.210. We then made TET's thresholds learnable, to see whether giving every layer its own threshold the way QCFS does would close this gap. The change in accuracy was at most 0.6 points, while NAS increased further on all three architectures. We also swapped TET loss for SDT loss to test whether the gap was specific to TET's per-timestep objective, and found that it was not (see Table 9 in Appendix B).

A fixed-strength spike penalty collapses accuracy. We penalized spike tensors directly at a fixed strength from epoch 0, and found that it collapsed accuracy. On ResNet-19/CIFAR-100, applying it at λ=10^{-6} reduced accuracy from 73.0% to 50.0%. Every other architecture–dataset pair we tested also lost accuracy, by 2.4 to 7.5 points (see Table 10 in Appendix B).

4 Method

An SNN trained with a spike penalty optimizes two loss functions that drive the same weights in opposite directions. For the training objective $\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda\,\mathcal{L}_{\text{SR}}$ (see Section 4.1), the gradient on any shared weight w is

$$\frac{\partial \mathcal{L}}{\partial w} = \underbrace{\frac{\partial \mathcal{L}_{\text{task}}}{\partial w}}_{\text{encourages firing}} + \lambda \underbrace{\frac{\partial \mathcal{L}_{\text{SR}}}{\partial w}}_{\text{suppresses firing}} . \tag{6}$$

Transfer learning faces a similar situation: pretrained knowledge must be adapted to new data without destroying what it already encodes. The transfer-learning literature offers several ways to resolve this; one is to control when and where the competing signal is applied (Howard & Ruder, 2018).

4.1 Time-weighted spike regularization

The penalty operates on the spike tensors directly rather than on any downstream proxy. Let $s^{(\ell)}_{b,t,n} \in \{0, 1\}$ be the spike emitted by neuron n of spiking layer ℓ at timestep t for sample b of a batch of size B, and let N_ℓ be the number of neurons in layer ℓ. We define

$$\mathcal{L}_{\text{SR}} = \sum_{\ell} \sum_{t=1}^{T} \frac{t}{T} \cdot \frac{1}{B} \sum_{b=1}^{B} \sum_{n=1}^{N_\ell} s^{(\ell)}_{b,t,n}, \qquad \mathcal{L} = \mathcal{L}_{\text{task}} + \lambda\,\mathcal{L}_{\text{SR}}. \tag{7}$$

Here $\mathcal{L}_{\text{task}} \in \{\mathcal{L}_{\text{TET}}, \mathcal{L}_{\text{SDT}}\}$ is the task loss. The weight t/T grows linearly with the timestep, so a spike at the final timestep is penalized T times more heavily than one at the first. Under $\mathcal{L}_{\text{TET}}$, every timestep must support a correct prediction, which encourages early and sustained firing. Weighting the penalty by t/T makes sustained firing more costly while leaving early spikes relatively inexpensive. Unlike TRT (Zhang et al., 2025), which acts on the weights with a coefficient that decays with t, SRSC acts on the spikes with a weight that grows with t.

4.2 ACCURACY GATE

The coefficient λ is held at a negligible value λ_{base} from the start of training until validation accuracy reaches a threshold ρ, which we use as a proxy for convergence of the task loss:

$$\lambda = \lambda_{\text{base}} \quad \text{until} \quad \text{acc}_{\text{val}} \geq \rho. \tag{8}$$

Before the gate fires, the penalty does not slow learning, and accuracy follows that of the unpenalized run. Figure 1 shows the effect of deferring the penalty, and Table 18 lists the underlying values. A fixed penalty of $\lambda = 10^{-6}$ applied from epoch 0 suppresses firing before the task loss has decreased. After the first epoch, its NAS is 0.06 against 0.37 without the penalty. By epoch 10, it reaches 3.6% accuracy against 20.8% without the penalty, and it does not recover. The penalty term remains below 5% of the total loss in this run, which suggests that the collapse is caused by when the penalty acts rather than by its magnitude.

SRSC, on the other hand, closely tracks accuracy of the unpenalized run (65.1% against 66.7%) until the gate is triggered at epoch 116. The ramp then reduces spikes from a network whose accuracy is already close to that of the unpenalized run. With $\lambda_{\text{ceil}} = 10^{-6}$, the same final coefficient as the fixed run, SRSC ends at 70.4% accuracy against 50.2%, at a similar NAS (0.22 against 0.20). With $\lambda_{\text{ceil}} = 5 \times 10^{-6}$, NAS falls from 0.30 to 0.14 while accuracy peaks at 70.2% and ends at 68.9%. This run ends with fewer spikes than the fixed run (0.14 against 0.20) and higher accuracy (68.9% against 50.2%).

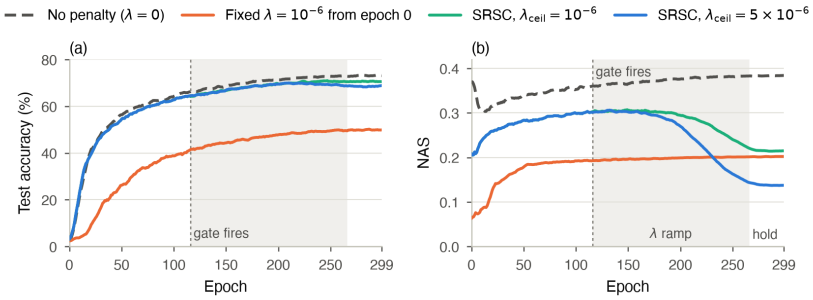


Figure 1: Test accuracy (a) and NAS (b) per epoch for ResNet-19/CIFAR-100 trained with $\mathcal{L}_{\text{TET}}$ using $T = 2$ timesteps. The two SRSC runs coincide until the SRSC gate fires.

We set $\lambda_{\text{base}} = 10^{-8}$ throughout and set ρ per dataset (Table 7). The gate fires between epochs 55 and 72 on CIFAR-10 and between epochs 84 and 129 on CIFAR-100. We ablate the accuracy gate against a fixed epoch-75 gate on CIFAR-10, whose results are shown in Table 1. The two perform alike, which suggests that the key factor is delaying the penalty until after the early stages of training.

Table 1: Accuracy gate vs. a fixed epoch-75 gate on CIFAR-10, $T = 4$.

Architecture	Gate	Acc (%)	NAS	SOPs (M)	Firing Rate
ResNet-18 ($\lambda_{\text{ceil}} = 10^{-6}$)	Accuracy	93.97	0.252	262.41	12.68%
	Epoch-75	94.18	0.249	258.40	12.66%
ResNet-19 ($\lambda_{\text{ceil}} = 5 \times 10^{-7}$)	Accuracy	94.79	0.200	773.11	8.55%
	Epoch-75	95.00	0.196	760.49	8.40%
VGG-16 ($\lambda_{\text{ceil}} = 5 \times 10^{-7}$)	Accuracy	92.33	0.154	164.69	11.92%
	Epoch-75	92.38	0.149	163.72	11.93%

4.3 GEOMETRIC RAMP

After the gate fires, λ grows geometrically over N_{ramp} epochs toward a ceiling λ_{ceil}:

$$\lambda_{k+1} = \min(\lambda_k \cdot g,\ \lambda_{\text{ceil}}), \qquad g = \left(\frac{\lambda_{\text{ceil}}}{\lambda_{\text{base}}}\right)^{1/N_{\text{ramp}}}, \tag{9}$$

with k indexing epochs after the gate. Once λ reaches λ_{ceil}, it is held there for the rest of training. We use $N_{\text{ramp}} = 150$ epochs in all experiments, so once ρ is set, λ_{ceil} is the only hyperparameter tuned for a given dataset and architecture.

4.4 PER-LAYER ALLOCATION

Per-layer spike regularization requires a measure of how much each layer contributes to the spiking activity and the accuracy. After the gate fires, a single forward–backward pass on one training batch gives, for each spiking layer ℓ,

$$S_\ell = \left\langle \left| \frac{\partial \mathcal{L}_{\text{TET}}}{\partial s^{(\ell)}} \odot s^{(\ell)} \right| \right\rangle, \tag{10}$$

where $\odot$ is the elementwise product and $\langle\cdot\rangle$ averages over batch, timesteps and neurons. The score S_ℓ is the product of two terms: a gradient term, $\partial \mathcal{L}_{\text{TET}}/\partial s^{(\ell)}$, which measures the sensitivity of the task loss to the layer's spikes, and an activity term, $s^{(\ell)}$, which measures the layer's firing rate. The score is therefore large only when a layer is both accuracy-sensitive and spike-active (see Appendix E). We use S_ℓ in two ways.

GR Binary Split. Gradient ranking (GR) ranks layers by S_ℓ and splits them at the median, with the sensitive half protected and the insensitive half suppressed:

$$\lambda_\ell = \begin{cases} \lambda \cdot \alpha, & S_\ell \geq \text{median}(S) \quad \text{(sensitive)} \\ \lambda \cdot \beta, & S_\ell < \text{median}(S) \quad \text{(insensitive)} \end{cases} \qquad \alpha = 0.7,\ \beta = 1.3. \tag{11}$$

OWL GradProxy. Every layer instead receives its own continuous scale, obtained by z-scoring the sensitivity distribution following OWL (Yin et al., 2024a):

$$z_\ell = \frac{S_\ell - \bar{S}}{\sigma_S}, \qquad \text{scale}_\ell = \text{clamp}(1 - k\, z_\ell,\ 0.1,\ 2.0)\,, \qquad \lambda_\ell = \lambda \cdot \text{scale}_\ell, \tag{12}$$

where $\bar{S}$ and σ_S are the mean and standard deviation of $\{S_\ell\}$ and k sets the sharpness of the allocation. A layer well above the mean receives $\text{scale}_\ell < 1$ and is protected. A layer well below the mean receives $\text{scale}_\ell > 1$ and is suppressed more strongly. The clamp bounds the allocation to $[0.1, 2.0]\times$ the global coefficient. We use $k = 0.3$ unless stated otherwise.

In both variants, λ_ℓ follows the global λ as it ramps, and the ratios λ_ℓ/λ change only when the scores are recomputed, which happens at the gate and every 25 epochs thereafter until λ reaches its ceiling. Setting $\alpha = \beta = 1$, or $k = 0$, recovers uniform SRSC exactly, which is the comparison described in Section 5.5.

5 EXPERIMENTAL SETUP AND RESULTS

We evaluate SRSC on standard image and event-driven benchmarks, and test whether its curriculum generalizes beyond our own codebase by applying it unchanged inside three independent, externally hosted training pipelines. Full training configuration, architectures, datasets, and baseline definitions are in Appendix A.

5.1 CIFAR-10 AND CIFAR-100

No-SR denotes the unregularized baseline against which every spike reduction below is computed (Table 2). SOPs, Firing Rate and Wt. Sp. are measured under 8-bit post-training quantization. No-SR and SRSC are evaluated at $T = 4$ timesteps.

Dataset	Model	Method	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
CIFAR-10	ResNet-18	No-SR	94.32	0.376	392.18	22.51%	2.62
		SRSC	93.97	0.252	262.41	12.68%	2.54
	ResNet-19	No-SR	95.12	0.368	1519.41	19.91%	13.31
		SRSC	94.79	0.200	773.11	8.55%	14.05
	VGG-16	No-SR	92.23	0.182	217.74	17.44%	24.66
		SRSC	92.33	0.154	164.69	11.92%	16.64
	SEW-ResNet-19, combined with weight pruning ($\lambda_{\text{weight}} = 5 \times 10^{-7}$)						
		UniSparse, dense (reproduced)	95.16	0.247	923.01	6.83%	0.00
		UniSparse, pruned (reproduced)	94.71	0.218	39.47	5.52%	92.71
		SRSC (ours), $\lambda_{\text{ceil}} = 10^{-6}$	94.31	0.134	29.84	2.82%	92.66
		SRSC (ours), $\lambda_{\text{ceil}} = 1.5 \times 10^{-6}$	94.18	0.115	27.53	2.37%	92.59
		SRSC (ours), $\lambda_{\text{ceil}} = 3 \times 10^{-6}$	93.50	0.090	24.47	1.80%	92.32
		ESLSNN (Shen et al., 2023)*	91.46	—	178.10	—	—
		STDS (Chen et al., 2022)*	91.64	—	56.49	—	—
		UPF (Shi et al., 2024)*	92.05	—	16.47	—	—
CIFAR-100	ResNet-18	No-SR	73.06	0.417	433.37	24.89%	16.23
		SRSC	72.94	0.347	354.93	18.44%	16.84
	ResNet-19	No-SR	74.69	0.383	1591.22	20.59%	19.72
		SRSC	72.85	0.238	929.41	10.56%	22.58
	VGG-16	No-SR	64.86	0.181	215.41	18.38%	8.47
		SRSC	64.66	0.162	189.66	14.46%	8.53
	SEW-ResNet-19, combined with weight pruning ($\lambda_{\text{weight}} = 5 \times 10^{-7}$)						
		UniSparse, dense (our run)	76.33	0.311	1131.41	8.26%	0.00
		UniSparse, pruned (our run)	75.97	0.285	77.63	7.27%	84.05
		SRSC (ours), $\lambda_{\text{ceil}} = 10^{-6}$	75.24	0.183	61.78	3.80%	84.06
		SRSC (ours), $\lambda_{\text{ceil}} = 1.5 \times 10^{-6}$	74.96	0.166	59.63	3.38%	83.84
		SRSC (ours), $\lambda_{\text{ceil}} = 3 \times 10^{-6}$	74.24	0.151	59.67	2.98%	83.13

Table 2: Accuracy, NAS, SOPs, Firing Rate and weight sparsity on CIFAR-10 and CIFAR-100, comparing SRSC against No-SR and, with weight pruning, against UniSparse Shi et al. (2026) and other state-of-art methods.

On CIFAR-10, SRSC reduces NAS by 45.7% for a 0.33% accuracy drop on ResNet-19, and by 33.0% for a 0.35% drop on ResNet-18. On VGG-16, SRSC reduces NAS by 15.4% with no loss of accuracy. Combined with UniSparse's weight pruning ($\lambda_{\text{weight}} = 5 \times 10^{-7}$), SRSC reduces SOPs by 24.4% relative to pruned UniSparse for a 0.40% accuracy drop on SEW-ResNet-19. SRSC is ahead of ESLSNN (91.46%, 178.10M SOPs) and STDS (91.64%, 56.49M) in both accuracy and SOPs, while UPF reaches fewer SOPs (16.47M) at lower accuracy (92.05%).

On CIFAR-100, SRSC lowers NAS by 37.9% for a 1.84% accuracy drop on ResNet-19, by 16.8% for a 0.12% drop on ResNet-18, and by 10.5% for a 0.20% drop on VGG-16. Combined with weight pruning, SRSC reduces SOPs by 20.4% relative to pruned UniSparse for a 0.73% accuracy drop. Across CIFAR-10 and CIFAR-100, SRSC, with and without weight pruning, yields the best accuracy–sparsity trade-off among the compared methods.

5.2 DVS-CIFAR10 (Event-based Dataset)

On DVS-CIFAR10 with VGGSNN at $T = 10$, SRSC reduces NAS by 67.9% and SOPs by 66.7% relative to *No-SR*, with no loss of accuracy (77.80% to 78.23%, Table 3). At the highest ceiling, $\lambda_{\text{ceil}} = 10^{-5}$, SRSC reaches 77.13% accuracy with an 87% NAS reduction, 156.47M SOPs, and a Firing Rate of 1.46%; the full sweep is reported in Appendix C).

Combined with weight pruning ($\lambda_{\text{weight}} = 2 \times 10^{-6}$), SRSC ($\lambda_{\text{ceil}} = 5 \times 10^{-6}$) reaches 82.50% accuracy against 83.22% for pruned UniSparse, while reducing SOPs by 16.5% (32.80M against 39.26M) and Firing Rate from 4.69% to 2.64%. ESLSNN (80.60%, 266.17M SOPs), STDS (81.70%, 225.34M) and UPF (79.00%, 10.02M) are included for context. They are not evaluated on comparable VGG-based architectures, so direct comparison is limited.

Dataset	Model	Method	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
DVS-CIFAR10	VGGSNN	No-SR	77.80	1.1866	841.75	8.98%	2.05
		SRSC	78.23	0.3805	280.33	2.91%	2.22
	VGGSNN, combined with weight pruning ($T = 10$, $\lambda_{\text{weight}} = 2 \times 10^{-6}$)						
		UniSparse, dense (reproduced)	83.50	0.6904	646.25	5.38%	0.00
		UniSparse, pruned (reproduced)	83.22	0.5545	39.26	4.69%	88.56
		SRSC (ours), $\lambda_{\text{ceil}} = 10^{-6}$	82.80	0.6042	45.79	4.28%	88.62
		SRSC (ours), $\lambda_{\text{ceil}} = 3 \times 10^{-6}$	82.40	0.4928	36.95	3.13%	88.64
		SRSC (ours), $\lambda_{\text{ceil}} = 5 \times 10^{-6}$	82.50	0.4336	32.80	2.64%	88.56
		ESLSNN (Shen et al., 2023)*	80.60	—	266.17	—	—
		STDS (Chen et al., 2022)*	81.70	—	225.34	—	—
		UPF (Shi et al., 2024)*	79.00	—	10.02	—	—

Table 3: Accuracy, NAS, SOPs, Firing Rate, and weight sparsity for VGGSNN on DVS-CIFAR10 at $T = 10$, comparing SRSC against No-SR and, with weight pruning, against UniSparse Shi et al. (2026).

5.3 Generalization to ImageNet Dataset and Transformer Architecture

Dataset	Model	Method	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
ImageNet ($T = 6$)	SEW-ResNet-34	No-SR	66.10	0.919	6909.59	14.70%	0.00
		SRSC	65.29	0.735	5354.51	10.55%	0.00
DVS-CIFAR10 ($T = 4$)	Spikeformer	No-SR	78.1	0.141	322.56	4.25%	84.02
		SRSC	78.3	0.086	200.49	2.22%	80.56

Table 4: Accuracy, NAS, SOPs, Firing Rate, and weight sparsity for SRSC against No-SR on ImageNet with SEW-ResNet-34 and on DVS-CIFAR10 with SpikeformerLi et al. (2022).

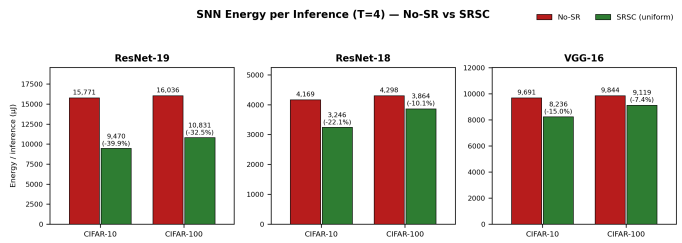
On ImageNet with SEW-ResNet-34 (IF neurons, TET loss), SRSC reduces NAS by 20.0%, SOPs by 22.5% and Firing Rate by 28.2% without relying on weight pruning (Table 4), at a minor accuracy cost consistent with the CIFAR results on ResNet architectures.

For a Transformer-based architecture, we evaluate Spikeformer (PLIF neurons) on DVS-CIFAR10, trained with SDT loss instead of TET loss. Accuracy is essentially unchanged (78.1% to 78.3%), while Firing Rate falls from 4.25% to 2.22%, NAS from 0.141 to 0.086, and SOPs by 37.8%. This track uses a 9:1 train/test split rather than the 7:3 split used in Table 6, so its absolute accuracy is not directly comparable to the DVS-CIFAR10 results in Section 5.2.

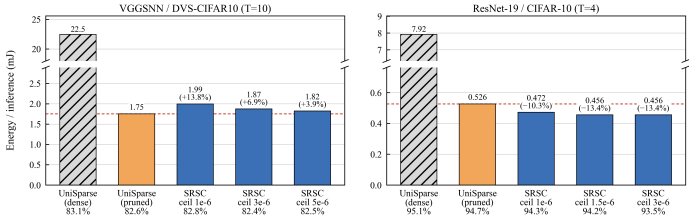
5.4 Energy

We estimate inference energy with the LoAS dual-sparse SNN accelerator model (Yin et al., 2024b) at 8-bit weights, using the measured weight sparsity and Firing Rate of each trained checkpoint at $T = 4$ for the CIFAR architectures and $T = 10$ for VGGSNN on DVS-CIFAR10 (Figure 2). Relative to the unregularized baseline, SRSC lowers energy on every architecture–dataset pair, by 39.9% on ResNet-19/CIFAR-10 at most and by 7.4% on VGG-16/CIFAR-100 at least.

With weight pruning held fixed, SRSC lowers energy relative to pruned UniSparse on SEW-ResNet-19/CIFAR-10 (472 against 526 μJ). On VGGSNN/DVS-CIFAR10, SRSC incurs slightly higher energy ($1{,}821$ against $1{,}752$ μJ) despite lower SOPs and Firing Rate. These figures are estimates from an analytical accelerator model, not measurements on hardware.



(a) No SR vs SRSC



(b) SRSC vs UniSparse

Figure 2: Estimated energy under the LoAS accelerator model (Yin et al., 2024b).

5.5 Does per-layer allocation help?

Section 4.4 scores each layer by S_ℓ (Equation 10), which is large when a layer is both accuracy-critical and spike-active, and allocates the penalty so that high-scoring layers are protected. The score separates layers by up to two orders of magnitude, and its ranking changes little across recomputations (Spearman ≥ 0.82). Allocating the penalty by this score does not improve on a uniform coefficient. Table 5 compares uniform SRSC against both allocations on ResNet-19 at matched λ_{ceil}. The three are within 0.8% accuracy of each other with no consistent ordering, comparable to the 0.03%–0.36% run-to-run variation of our sweeps. ResNet-18 and VGG-16 behave the same way, and neither higher ceilings nor stronger protection of the sensitive layers produces an advantage. Across six architecture–dataset pairs, a uniform coefficient performs as well as either allocation, so once ρ is set, λ_{ceil} remains the only hyperparameter tuned. Appendix E reports the full analysis and sweeps.

CIFAR-10				CIFAR-100			
λ_{ceil}	Uniform SRSC	GR Binary Split	OWL GradProxy	λ_{ceil}	Uniform SRSC	GR Binary Split	OWL GradProxy
5×10^{-7}	94.39 / 0.243	94.13 / 0.246	94.26 / 0.246	5×10^{-7}	70.58 / 0.260	70.60 / 0.269	71.01 / 0.270
10^{-6}	94.06 / 0.176	94.23 / 0.193	94.11 / 0.183	10^{-6}	69.98 / 0.249	70.09 / 0.251	69.70 / 0.260
3×10^{-6}	93.75 / 0.210	93.73 / 0.164	93.46 / 0.224	5×10^{-6}	70.18 / 0.227	69.53 / 0.171	70.34 / 0.271

Table 5: Uniform SRSC vs. the two per-layer allocations. Each cell reports accuracy (%) / NAS.

6 Conclusion

Direct training of SNNs with both task loss and spike penalty leads to opposing gradients on the same weights. To resolve this, we proposed SRSC, which schedules when the spike penalty triggers. SRSC holds the penalty negligible until validation accuracy reaches a threshold or training crosses a fixed number of epochs, after which the penalty is ramped up geometrically to a ceiling. The proposed curriculum reduces spiking activity substantially across convolutional and Transformer-based SNN architectures, on both static and event-based datasets.

A few limitations remain. The gate threshold ρ is set per architecture and dataset and λ_{ceil} per architecture, both by hand, though a fixed-epoch trigger performs alike (see Table 1). Energy reductions (Section 5.4) are estimated from the LoAS accelerator model at 8-bit weights, not measured on hardware. Main results (Tables 2, 3) use a single training seed, with no reported variance.

Reproducibility Statement

Appendix A gives the training configuration for every experiment: the home-codebase settings (Table 6), the SRSC hyperparameters for each architecture and dataset (Table 7), the hosted-codebase tracks (Table 8), and the baselines (Appendix A.4). Section 3.2 defines every reported metric, and Appendix C gives the full λ_{ceil} sweeps. Code and configuration files for every experiment will be released on publication.

Disclosure of AI Use

In this work, we used generative AI tools to implement and modify implementation code. We did not use generative AI tools to propose hypotheses or to design the method. Additionally, we used generative AI tools to draft and edit parts of the paper for readability and consistency of terminology, to format tables, and to search for information. All AI-assisted text was reviewed by the authors, and AI-modified code was verified against its outputs before any result was reported. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Filipp Akopyan, Jun Sawada, Andrew S. Cassidy, Alvarez-Icaza Rodrigo, John V. Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi Joon Nam, Brian Taba, Michael P. Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan L. Jackson, and Dharmenda S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. In *IEEE Transactions on Computer-Aided Design og Integrated Circuits and Systems (TCADIS)*, 2015. doi: 10.1109/TCAD.2015.2474396.

Jason M. Allred, Steven J. Spencer, Gopalakrishnan Srinivasan, and Kaushik Roy. Explicitly trained spiking sparsity in spiking neural networks with backpropagation, 2020. URL https://arxiv.org/abs/2003.01250.

Tong Bu, Wei Fang, Jianhao Ding, PengLin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2303.04347.

Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1713–1721, 2021. URL https://arxiv.org/abs/2105.04916.

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 *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 3701–3715. PMLR, 2022. URL https://proceedings.mlr.press/v162/chen22ac.html.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, and Sri Harsha Choday. Loihi: A neuromorphic manycore processor with on-chip learning. In *IEEE MICRO, Vol. 38*, 2018. doi: 10.1109/MM.2018.112130359.

Lei Deng, Yujie Wu, Yifan Hu, Ling Liang, Guoqi Li, Xing Hu, Yufei Ding, Peng Li, and Yuan Xie. Comprehensive SNN compression using ADMM optimization and activity regularization. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. URL https://arxiv.org/abs/1911.00822.

Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2202.11946.

Song Han, Huizi Mao, and William Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016. URL https://arxiv.org/abs/1510.00149.

Ahmed Hassan, Jian Meng, and Jae-sun Seo. Spiking neural network with learnable threshold for event-based classification and object detection. In *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024. doi: 10.1109/IJCNN60899.2024.10650320.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339. Association for Computational Linguistics, 2018. URL https://arxiv.org/abs/1801.06146.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5544–5555. PMLR, 2020. URL https://arxiv.org/abs/2002.03231.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2, 1989. URL https://proceedings.neurips.cc/paper/1989/hash/6c9882bbac1c7093bd25041881277658-Abstract.html.

Yudong Li, Yunlin Lei, and Xu Yang. Spikeformer: A novel architecture for training high-performance low-latency spiking neural network. *arXiv preprint arXiv:2211.10686*, 2022. URL https://arxiv.org/abs/2211.10686.

Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. doi: 10.1109/MSP.2019.2931595.

Thomas Pellegrini, Romain Zimmer, and Timothée Masquelier. Low-activity supervised convolutional spiking neural networks applied to speech commands recognition. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pp. 97–103. IEEE, 2021. URL https://arxiv.org/abs/2011.06846.

Nitin Rathi and Kaushik Roy. DIET-SNN: A low-latency spiking neural network with direct input encoding & leakage and threshold optimization. *arXiv preprint arXiv:2008.03658*, 2020. URL https://arxiv.org/abs/2008.03658.

Yusuke Sakemi, Kakei Yamamoto, Takeo Hosomi, and Kazuyuki Aihara. Sparse-firing regularization methods for spiking neural networks with time-to-first-spike coding. *Scientific Reports*, 13 (1):22897, 2023. doi: 10.1038/s41598-023-50201-5.

Jiangrong Shen, Qi Xu, Jian K. Liu, Yueming Wang, Gang Pan, and Huajin Tang. ESL-SNNs: An evolutionary structure learning strategy for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 86–93, 2023. URL https://arxiv.org/abs/2306.03693.

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. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/dd3c889922df2112a5b1769e3c19e28e-Abstract-Conference.html.

Xinyu Shi, Tong Bu, and Zhaofei Yu. Unisparse: Combining weight pruning and spike sparsification in spiking neural networks. *International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/64520.

Kazuma Suetake, Takuya Ushimaru, Ryuji Saiin, and Yoshihide Sawada. Synaptic interaction penalty: Appropriate penalty term for energy-efficient spiking neural networks. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2302.01500.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018. doi: 10.3389/fnins.2018.00331.

Yulong Yan, Haoming Chu, Yi Jin, Yuxiang Huan, Zhuo Zou, and Lirong Zheng. Backpropagation with sparsity regularization for spiking neural network learning. *Frontiers in Neuroscience*, 16: 760298, 2022. doi: 10.3389/fnins.2022.760298.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Kumar Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57101–57115. PMLR, 2024a. URL https://arxiv.org/abs/2310.05175.

Ruokai Yin, Youngeun Kim, Di Wu, and Priyadarshini Panda. Loas: Fully temporal-parallel dataflow for dual-sparse spiking neural networks. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1107–1121, 2024b. doi: 10.1109/MICRO61859.2024.00084.

Boxuan Zhang, Zhen Xu, and Kuan Tao. Enhancing generalization of spiking neural networks through temporal regularization, 2025. URL https://arxiv.org/abs/2506.19256.

A Experimental Setup

Expands on Section 5: full training configuration, SRSC hyperparameters, and baseline definitions for every experiment in the paper.

A.1 Home-codebase training configuration

Table 6: Home-codebase training configuration. All runs: Adam optimizer, cosine-annealed learning rate decaying to zero, batch size 128, single seed (1000). The 50-epoch fine-tuning phase (CIFAR only) uses Adam at 10^{-4}, applied after the base phase's cosine schedule reaches zero.

Architecture	Dataset	LR	Epochs	T (base $\rightarrow$ fine-tune)	Augmentation	TET aux-reg
ResNet-18	CIFAR-10	10^{-2}	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
ResNet-18	CIFAR-100	10^{-2}	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
ResNet-19	CIFAR-10	10^{-2}	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
ResNet-19	CIFAR-100	10^{-2}	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
VGG-16	CIFAR-10	$5{\times}10^{-4}$	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
VGG-16	CIFAR-100	$5{\times}10^{-4}$	300 (+50 fine-tune)	$2 \rightarrow 4$	crop+flip+Cutout	10^{-3}
VGGSNN	DVS-CIFAR10	10^{-3}	300	10	flip+crop	10^{-4}

A.2 SRSC HYPERPARAMETERS

Table 7: SRSC hyperparameters by architecture/dataset (home codebase). Shared across every run: $\lambda_{\text{base}} = 10^{-8}$, $N_{\text{ramp}} = 150$ epochs, and $\lambda_{\text{ceil}} \in \{5\times10^{-7}, 10^{-6}, 3\times10^{-6}, 5\times10^{-6}, 10^{-5}, 2\times10^{-5}\}$ swept as the only trade-off knob. Per-layer variants (Section 5.5): $\alpha = 0.7$, $\beta = 1.3$ for GR Binary Split, $k = 0.3$ for OWL GradProxy, scores recomputed every 25 epochs during the ramp.

Architecture	Dataset	Gate threshold ρ
ResNet-18	CIFAR-10	90%
ResNet-19	CIFAR-10	90%
VGG-16	CIFAR-10	87%
ResNet-18	CIFAR-100	65%
ResNet-19	CIFAR-100	65%
VGG-16	CIFAR-100	58%
VGGSNN	DVS-CIFAR10	65% (70% at $\lambda_{\text{ceil}} = 10^{-5}$)

A.3 HOSTED-CODEBASE GENERALIZATION TRACKS

Each hosted run keeps its own codebase's architecture, optimizer, and data pipeline, adding only SRSC. The DVS-CIFAR10 tracks below use a 9:1 train/test split, against this paper's own 7:3 split (Table 6), so DVS-CIFAR10 numbers are not directly comparable across tracks. UniSparse's own codebase trains with SGD rather than the Adam optimizer used everywhere else in this paper.

Table 8: Hosted-codebase generalization tracks. LR/optimizer/batch size are not yet gathered for the ImageNet track (marked —).

Host	Architecture	Dataset	Loss	T	LR	optimizer	Batch	ρ	Split
UniSparse	SEW-ResNet-19	CIFAR-10	UniSparse's own	4	0.1	SGD	256	90%	standard
UniSparse	VGGSNN	DVS-CIFAR10	TET	10	0.025	SGD	64	80%	9:1
Spikeformer	PLIF-Spikeformer	DVS-CIFAR10	not TET[1]	4	0.001	Adam	32	65%	9:1
(independent)	SEW-ResNet-34	ImageNet	TET	6	—	—	—	—	standard

A.4 BASELINES

We compare against two references. *No-SR* (no spike regularization) is the same learnable-threshold TET model trained with $\lambda = 0$, and is the baseline against which all spike reductions are computed; it isolates the effect of the regularizer from every other training choice. *UniSparse* (Shi et al., 2026) represents the combination of weight pruning with a fixed-strength spike-suppression mechanism; we reproduce their published dense and pruned operating points and, holding their weight-pruning component fixed, substitute SRSC's curriculum for their own spike-suppression mechanism.

B PRELIMINARY EVIDENCE FOR THE DESIGN OF SRSC

This appendix gives the full data behind the two observations motivating SRSC in Section 3.3: that direct training fires more than conversion, and that an unscheduled spike penalty is not usable on its own.

Direct training fires more than conversion, across architectures and loss functions. Table 9 compares QCFS, plain TET, TET with learnable thresholds (TET-LT), and TET-LT trained with SDT loss instead of TET loss, on CIFAR-10 at $T = 2$. Every direct-training variant reaches comparable or better accuracy than QCFS on every architecture, but at a higher NAS; TET-LT further increases NAS over plain TET, and swapping in SDT loss does not close the gap to QCFS either.

[1]Standard cross-entropy on the time-averaged output; the codebase's own training script notes "TET-loss component intentionally omitted for this backbone."

Table 9: QCFS, TET, TET-LT, and SDT-LT on CIFAR-10, $T = 2$.

Architecture	Method	Acc (%)	NAS
ResNet-18	QCFS	88.70	0.2187
	TET	93.83	0.3970
	TET-LT	94.05	0.4030
	SDT-LT	94.15	0.3498
ResNet-19	QCFS	87.35	0.2104
	TET	95.21	0.3162
	TET-LT	94.93	0.3650
	SDT-LT	94.45	0.3202
VGG-16	QCFS	86.62	0.2229
	TET	91.49	0.2433
	TET-LT	92.10	0.2710
	SDT-LT	92.64	0.2331

A fixed-strength physical penalty trades accuracy for sparsity uncontrollably. Table 10 applies the spike-tensor penalty of Section 3.3 at a single fixed λ from epoch 0, with no gate and no **A fixed-strength spike penalty trades accuracy for sparsity uncontrollably.** TET-LT baseline, from a moderate cost on ResNet-18/VGG-16 to a severe one on ResNet-19/CIFAR-100.

Table 10: Fixed-strength spike regularization applied from epoch 0, no gate or ramp. "LT" columns are each architecture's TET-LT baseline (no penalty); "Test" columns are the fixed-λ run.

Arch	Dataset	LT Acc (%)	LT NAS	Test λ	Test Acc (%)	Test NAS	Δ Acc
ResNet-18	CIFAR-10	94.05	0.403	10^{-6}	91.62	0.267	-2.4%
ResNet-19	CIFAR-10	94.93	0.365	10^{-6}	87.47	0.209	-7.5%
VGG-16	CIFAR-10	92.10	0.351	10^{-6}	89.19	0.166	-2.9%
ResNet-18	CIFAR-100	73.08	0.443	10^{-7}	69.50	0.385	-3.6%
ResNet-19	CIFAR-100	73.00	0.384	10^{-6}	50.00	0.200	-23.0%
VGG-16	CIFAR-100	64.86	0.361	3×10^{-7}	58.31	0.289	-6.5%

C FULL λ_{ceil} SWEEPS ON CIFAR-10, CIFAR-100 AND DVS-CIFAR10

Tables 11–16 give every CIFAR-10 and CIFAR-100 run at the end of the $T = 2$ training phase, with NAS, SOPs, Firing Rate and Wt. Sp. measured in one consistent pass (8-bit post-training quantization for SOPs, Firing Rate and Wt. Sp.). All sweep runs use $T = 2$, $\lambda_{\text{base}} = 10^{-8}$ and $N_{\text{ramp}} = 150$. Table 2 reports one SRSC run per table after 50 epochs of fine-tuning at $T = 4$ (Table 6). The last block of each table repeats those $T = 4$ values, together with No-SR at $T = 4$. "SRSC" denotes the uniform coefficient; GR and OWL are the per-layer variants of Section 4.4. QCFS rows carry only Acc/NAS, since QCFS is a converted ANN rather than an SRSC-trained checkpoint and is outside this sweep's measurement pass. Table 17 gives the corresponding sweep for VGGSNN/DVS-CIFAR10, at $T = 10$ with the same λ_{base} and N_{ramp}; the accuracy gate differs by row (noted in the Config column), as set in Table 7.

Table 11: ResNet-18 / CIFAR-10.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 2$	88.70	0.2187	—	—	—
No-SR	—	94.06	0.3717	193.95	22.23%	2.62
SRSC	10^{-6}	93.46	0.2517	131.13	12.74%	2.54
SRSC	3×10^{-6}	92.98	0.2020	105.19	9.68%	2.55
SRSC	5×10^{-6}	93.40	0.2692	141.22	14.33%	2.47
GR (α=0.7, β=1.3)	10^{-6}	93.59	0.2490	129.00	12.11%	2.51
GR (α=0.7, β=1.5)	10^{-6}	93.50	0.2495	128.81	12.10%	2.22
GR (α=0.5, β=1.5)	10^{-6}	93.39	0.2538	130.85	12.21%	2.18
GR (α=0.7, β=1.3)	3×10^{-6}	92.82	0.2029	105.00	9.46%	2.17
GR (α=0.7, β=1.3)	5×10^{-6}	93.03	0.2753	144.13	14.26%	2.43
OWL (k=0.1)	10^{-6}	93.75	0.2562	132.83	12.94%	2.07
OWL (k=0.3)	10^{-6}	93.54	0.2455	126.83	12.03%	2.31
OWL (k=0.5)	10^{-6}	93.58	0.2513	129.64	12.11%	2.15
OWL (k=0.3)	3×10^{-6}	93.29	0.2890	151.06	15.34%	2.46
OWL (k=0.3)	5×10^{-6}	93.13	0.2950	154.65	15.92%	2.52
No-SR ($T = 4$)	—	94.32	0.376	392.18	22.51%	2.62
SRSC ($T = 4$)	10^{-6}	93.97	0.252	262.41	12.68%	2.54

Table 12: ResNet-19 / CIFAR-10.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 2$	87.35	0.2104	—	—	—
No-SR	—	94.93	0.3647	752.97	19.70%	13.31
SRSC	5×10^{-7}	94.39	0.2430	485.36	11.21%	14.05
SRSC	10^{-6}	94.06	0.1760	332.11	7.45%	14.81
SRSC	3×10^{-6}	93.75	0.2102	414.09	9.40%	13.73
SRSC	5×10^{-6}	93.64	0.2300	461.37	10.69%	13.62
SRSC	10^{-5}	93.36	0.2239	447.22	10.28%	13.28
SRSC	2×10^{-5}	93.27	0.2101	418.36	9.51%	13.06
GR (α=0.7, β=1.3)	5×10^{-7}	94.13	0.2463	490.28	11.25%	13.99
GR (α=0.7, β=1.3)	10^{-6}	94.23	0.1930	367.48	8.02%	14.65
GR (α=0.7, β=1.3)	3×10^{-6}	93.73	0.1636	307.98	6.66%	14.67
OWL (k=0.3)	5×10^{-7}	94.26	0.2462	489.43	11.35%	14.21
OWL (k=0.3)	10^{-6}	94.11	0.1827	344.16	7.83%	14.44
OWL (k=0.5)	10^{-6}	94.31	0.1672	308.21	7.02%	14.56
OWL (k=0.7)	10^{-6}	94.03	0.1803	339.41	7.70%	14.69
OWL (k=0.3)	3×10^{-6}	93.46	0.2236	445.80	10.06%	13.56
No-SR ($T = 4$)	—	95.12	0.368	1519.41	19.91%	13.31
SRSC ($T = 4$)	5×10^{-7}	94.79	0.200	773.11	8.55%	14.05

Table 13: VGG-16 / CIFAR-10.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 2$	86.62	0.2229	—	—	—
No-SR	—	92.00	0.1783	105.99	16.92%	24.66
SRSC	5×10^{-7}	91.65	0.1467	78.93	11.36%	16.64
SRSC	10^{-6}	90.95	0.1289	69.68	9.65%	15.33
SRSC	3×10^{-6}	90.29	0.1019	57.25	7.50%	13.95
GR (α=0.7, β=1.3)	5×10^{-7}	90.94	0.1443	76.61	10.72%	14.69
GR (α=0.7, β=1.3)	10^{-6}	90.77	0.1250	65.75	8.71%	15.49
GR (α=0.7, β=1.3)	3×10^{-6}	90.00	0.1082	59.36	7.47%	14.42
OWL (k=0.3)	5×10^{-7}	91.07	0.1446	75.56	10.54%	14.32
OWL (k=0.3)	10^{-6}	90.63	0.1276	65.59	8.73%	13.69
OWL (k=0.3)	3×10^{-6}	89.73	0.1081	56.73	7.15%	15.45
No-SR ($T = 4$)	—	92.23	0.182	217.74	17.44%	24.66
SRSC ($T = 4$)	5×10^{-7}	92.33	0.154	164.69	11.92%	16.64

Table 14: ResNet-18 / CIFAR-100.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 4$	66.99	0.446	—	—	—
No-SR	—	71.68	0.4119	214.13	24.57%	16.23
SRSC	5×10^{-7}	71.46	0.3436	175.74	18.27%	16.84
SRSC	10^{-6}	71.88	0.3143	160.16	16.05%	17.25
SRSC	3×10^{-6}	69.81	0.2579	130.25	12.19%	17.09
SRSC	5×10^{-6}	70.49	0.3223	164.95	16.90%	17.28
GR (α=0.7, β=1.3)	5×10^{-7}	71.65	0.3355	170.76	17.37%	17.66
GR (α=0.7, β=1.3)	10^{-6}	71.25	0.3130	158.61	15.53%	16.97
GR (α=0.7, β=1.3)	3×10^{-6}	69.61	0.2562	128.40	11.63%	17.58
GR (α=0.7, β=1.3)	5×10^{-6}	70.46	0.3381	173.66	18.12%	16.95
OWL (k=0.3)	5×10^{-7}	71.40	0.3365	171.61	17.54%	18.06
OWL (k=0.3)	10^{-6}	71.09	0.3085	156.57	15.32%	17.90
OWL (k=0.3)	3×10^{-6}	70.83	0.3132	159.58	15.84%	17.04
OWL (k=0.3)	5×10^{-6}	70.23	0.3296	168.39	17.15%	16.90
No-SR ($T = 4$)	—	73.06	0.417	433.37	24.89%	16.23
SRSC ($T = 4$)	5×10^{-7}	72.94	0.347	354.93	18.44%	16.84

Table 15: ResNet-19 / CIFAR-100.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 4$	50.70	0.417	—	—	—
No-SR	—	73.21	0.3808	789.66	20.41%	19.72
SRSC	5×10^{-7}	70.58	0.2603	517.63	12.04%	22.58
SRSC	10^{-6}	69.98	0.2491	495.65	11.35%	22.45
SRSC	5×10^{-6}	70.18	0.2265	443.61	10.00%	22.21
SRSC	7×10^{-6}	70.14	0.2097	405.79	9.07%	22.69
SRSC	10^{-5}	70.04	0.2284	448.14	10.10%	22.13
SRSC	2×10^{-5}	69.85	0.2255	441.72	9.95%	22.43
GR (α=0.7, β=1.3)	5×10^{-7}	70.60	0.2687	534.23	12.33%	22.52
GR (α=0.7, β=1.3)	10^{-6}	70.09	0.2511	498.19	11.32%	23.35
GR (α=0.7, β=1.3)	5×10^{-6}	69.53	0.1714	321.47	6.94%	23.04
GR (α=0.7, β=1.3)	7×10^{-6}	70.23	0.2472	487.65	11.02%	22.34
GR (α=0.7, β=1.3)	10^{-5}	69.86	0.1839	345.34	7.51%	22.48
OWL (k=0.3)	5×10^{-7}	71.01	0.2696	529.23	12.35%	22.92
OWL (k=0.3)	10^{-6}	69.70	0.2603	512.28	11.81%	22.83
OWL (k=0.3)	5×10^{-6}	70.34	0.2708	537.39	12.64%	22.81
OWL (k=0.3)	7×10^{-6}	70.50	0.2392	464.90	10.63%	22.83
OWL (k=0.3)	10^{-5}	69.75	0.2549	501.51	11.61%	22.31
OWL (k=0.3)	2×10^{-5}	69.61	0.2195	421.64	9.56%	22.34
No-SR ($T = 4$)	—	74.69	0.383	1591.22	20.59%	19.72
SRSC ($T = 4$)	5×10^{-7}	72.85	0.238	929.41	10.56%	22.58

Table 16: VGG-16 / CIFAR-100.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
QCFS	$T = 4$	61.88	0.530	—	—	—
No-SR	—	63.45	0.1768	104.75	17.79%	8.47
SRSC	5×10^{-7}	62.90	0.1579	92.09	13.94%	8.53
SRSC	10^{-6}	62.26	0.1482	85.24	12.45%	8.06
SRSC	5×10^{-6}	62.64	0.1643	97.79	15.23%	8.25
SRSC	10^{-5}	62.74	0.1590	93.52	14.23%	8.26
GR (α=0.7, β=1.3)	5×10^{-7}	62.56	0.1552	89.73	13.44%	8.80
GR (α=0.7, β=1.3)	10^{-6}	61.62	0.1454	82.18	11.73%	8.48
GR (α=0.7, β=1.3)	5×10^{-6}	62.70	0.1588	92.74	13.97%	8.61
GR (α=0.7, β=1.3)	10^{-5}	62.28	0.1589	93.28	14.12%	8.32
OWL (k=0.3)	5×10^{-7}	62.33	0.1557	90.18	13.51%	8.06
OWL (k=0.3)	10^{-6}	62.09	0.1463	82.70	11.81%	7.84
OWL (k=0.3)	5×10^{-6}	62.57	0.1631	96.80	15.00%	7.78
OWL (k=0.3)	10^{-5}	62.22	0.1596	93.70	14.24%	8.17
No-SR ($T = 4$)	—	64.86	0.181	215.41	18.38%	8.47
SRSC ($T = 4$)	5×10^{-7}	64.66	0.162	189.66	14.46%	8.53

Table 17: VGGSNN / DVS-CIFAR10. SOPs, Firing Rate and Wt. Sp. are measured at the same final-epoch checkpoint as NAS; Acc is the peak validation accuracy reported elsewhere for this run (Section 5.2), from an earlier epoch than the other four columns.

Method	Config	Acc (%)	NAS	SOPs (M)	Firing Rate	Wt. Sp. (%)
No-SR	—	77.80	1.1866	841.75	8.98%	2.05
SRSC	5×10^{-7}, gate 65	78.23	0.3805	280.33	2.91%	2.22
SRSC	10^{-6}, gate 65	77.93	0.3128	258.01	2.59%	2.13
SRSC	5×10^{-6}, gate 65	77.27	0.1962	186.08	1.86%	2.19
SRSC	10^{-5}, gate 70	77.13	0.1538	156.47	1.46%	2.30

D Additional Results

D.1 Training dynamics

Table 18 lists the accuracy and NAS behind Figure 1 every 20 epochs, together with the epochs quoted in Section 4.2. Values are per epoch, without smoothing.

Table 18: Accuracy (%) and NAS per epoch for the runs in Figure 1 (ResNet-19/CIFAR-100, $\mathcal{L}_{\mathrm{TET}}$, $T = 2$). The two SRSC runs are identical until the gate fires at epoch 116.

	No penalty		Fixed, $\lambda = 10^{-6}$		SRSC, $\lambda_{\mathrm{ceil}} = 10^{-6}$		SRSC, $\lambda_{\mathrm{ceil}} = 5 \times 10^{-6}$	
Epoch	Acc	NAS	Acc	NAS	Acc	NAS	Acc	NAS
0	2.02	0.369	2.25	0.057	2.23	0.170	2.23	0.170
10	20.83	0.309	3.59	0.088	22.22	0.238	22.22	0.238
20	40.00	0.323	8.39	0.118	39.80	0.262	39.80	0.262
40	52.16	0.336	21.73	0.163	50.76	0.274	50.76	0.274
60	59.54	0.342	31.17	0.184	56.66	0.284	56.66	0.284
80	62.08	0.350	34.65	0.192	60.45	0.291	60.45	0.291
100	64.14	0.348	39.66	0.192	62.45	0.302	62.45	0.302
116	66.67	0.359	41.37	0.190	65.11	0.302	65.11	0.302
120	65.75	0.358	40.82	0.194	63.87	0.306	64.30	0.299
140	67.65	0.367	44.60	0.196	66.92	0.303	66.13	0.301
160	69.57	0.368	45.27	0.196	67.44	0.303	67.17	0.303
180	71.08	0.371	46.71	0.197	69.08	0.300	68.92	0.291
200	71.20	0.377	48.23	0.199	69.86	0.296	69.86	0.273
220	72.34	0.378	48.07	0.199	69.89	0.280	69.52	0.226
221	72.03	0.376	48.69	0.200	70.52	0.281	70.18	0.226
240	72.43	0.381	49.25	0.200	70.95	0.256	68.97	0.181
260	72.87	0.382	49.49	0.201	70.22	0.228	68.79	0.150
280	72.83	0.383	49.63	0.202	70.32	0.215	68.55	0.138
299	73.08	0.384	50.22	0.202	70.38	0.215	68.86	0.137

E Per-layer allocation: the negative result in full

Section 4.4 allocates the penalty per layer on the hypothesis that a layer whose spikes matter more to accuracy should be regularized less. This appendix tests that hypothesis in full: the sensitivity signal it relies on is strong and stable, yet the allocation it induces does not beat a uniform coefficient.

E.1 What the sensitivity score measures

The score of Equation 10, $S_\ell = \langle |\partial \mathcal{L}_{\mathrm{TET}} / \partial s^{(\ell)} \odot s^{(\ell)}| \rangle$, is the product of two distinct per-layer signals. The gradient $\partial \mathcal{L}_{\mathrm{TET}} / \partial s^{(\ell)}$ measures how strongly the task loss—and hence accuracy—responds to the layer's spikes; the spike trace $s^{(\ell)}$ measures how active the layer is. A layer scores high on S_ℓ only when it is both accuracy-critical and spike-heavy. The two terms pull in opposite directions: the gradient alone would protect accuracy-critical layers regardless of their activity, while

the activity alone would suppress the busiest layers regardless of their importance. We chose their product deliberately, to avoid silencing layers that carry accuracy while still targeting the busiest ones; it is the quantity both GR Binary Split and OWL GradProxy rank layers by.

E.2 THE SIGNAL IS LARGE AND STABLE

S_ℓ separates layers by well over an order of magnitude on every architecture we train. Table 19 lists the highest- and lowest-scoring LIF layer at a single recomputation for representative runs; the spread ranges from $3.3\times$ on ResNet-18/CIFAR-10 to $127\times$ on ResNet-19/CIFAR-100. The separation is structural rather than diffuse: on ResNet-19 it is dominated by the output-adjacent spiking layer $\ell = 8$, which scores 14–$100\times$ above the other layers across both datasets and is selected for protection at every recomputation of every ResNet-19 run.

Table 19: Per-layer sensitivity spread at a single recomputation: the highest- and lowest-scoring LIF layer, with the layer index in brackets and S_ℓ from Equation 10.

Arch / Dataset	Setting	Top layer S_ℓ	Bottom layer S_ℓ	Spread
ResNet-18 / CIFAR-10	OWL k=0.3	7.63×10^{-6} (ℓ6)	2.30×10^{-6} (ℓ7)	$3.3\times$
ResNet-18 / CIFAR-100	GR	5.68×10^{-6} (ℓ6)	1.00×10^{-6} (ℓ8)	$5.7\times$
ResNet-19 / CIFAR-10	GR	1.22×10^{-5} (ℓ8)	1.75×10^{-7} (ℓ7)	$70\times$
ResNet-19 / CIFAR-100	OWL k=0.3	8.06×10^{-6} (ℓ10)	2.14×10^{-7} (ℓ1)	$38\times$
ResNet-19 / CIFAR-100	GR	6.52×10^{-5} (ℓ8)	5.11×10^{-7} (ℓ0)	$127\times$

The induced ranking is also stable. Scores are recomputed every 25 epochs over the ramp, six times per run and 150 times across the logged runs. Between any two recomputations within a run the Spearman correlation of $\{S_\ell\}$ is at least 0.82 and averages 0.96, with only minor flipping among mid-network layers under OWL on ResNet-18. No layer moves from protected to suppressed or back within a run, so each scheme ends the ramp applying the allocation it chose at the first recomputation.

E.3 ALLOCATION DOES NOT TRANSLATE INTO AN ADVANTAGE

Table 20 compares uniform SRSC against both schemes on ResNet-19 at matched λ_{ceil}. At $\lambda_{\text{ceil}} = 5 \times 10^{-7}$ on CIFAR-10 the three land within 0.26pp of each other at NAS 0.243–0.246 and which of them leads changes at the next ceiling; over the whole table the best either scheme manages at a matched spike level is $+0.43$pp and the worst -0.65pp, with inconsistent sign, and ResNet-18 and VGG-16 behave the same way on both datasets. Configurations differing only in parameters we show to be inert span 0.03–0.36pp (Appendix C), so we read differences below roughly 0.2pp as within variation.

Table 20: Uniform SRSC against the two per-layer allocations at matched λ_{ceil}, ResNet-19, $T = 2$. GR uses $\alpha = 0.7$, $\beta = 1.3$; OWL uses $k = 0.3$. Differences of the size shown here are at or below the run-to-run variation established in the text (0.03–0.36pp).

		Uniform SRSC		GR Binary Split		OWL GradProxy	
Dataset	λ_{ceil}	Acc (%)	NAS	Acc (%)	NAS	Acc (%)	NAS
CIFAR-10	5×10^{-7}	94.39	0.243	94.13	0.246	94.26	0.246
	10^{-6}	94.06	0.176	94.23	0.193	94.11	0.183
	3×10^{-6}	93.75	0.210	93.73	0.164	93.46	0.224
CIFAR-100	5×10^{-7}	70.58	0.260	70.60	0.269	71.01	0.270
	10^{-6}	69.98	0.249	70.09	0.251	69.70	0.260
	5×10^{-6}	70.18	0.227	69.53	0.171	70.34	0.271
	10^{-5}	70.04	0.228	69.86	0.184	69.75	0.255
	2×10^{-5}	69.85	0.226	—	—	69.61	0.220

Two further hypotheses do not survive testing. Were the allocation of use only where the penalty is strong enough to cost accuracy, an advantage should emerge at aggressive ceilings; swept to $\lambda_{\text{ceil}} = 2 \times 10^{-5}$ on CIFAR-100, the three methods decline in step. Were the sensitive layers instead protected too weakly, relaxing them further should recover accuracy; sharpening OWL from $k = 0.3$ to $k = 0.5$ and $k = 0.7$ on ResNet-19/CIFAR-10—the first regime in which the most sensitive layers reach the clamp floor and are genuinely freed—does not deliver a reliable accuracy gain: $k = 0.5$ ticks up within the run-to-run noise band and $k = 0.7$ falls back toward it (Table 21), and no (α, β) setting we evaluated does better. Across six architecture–dataset pairs a uniform coefficient is as good as either allocation, and λ_{ceil} remains the only knob needed to move along that trade-off.

Table 21: OWL GradProxy on ResNet-19/CIFAR-10 at $\lambda_{\text{ceil}} = 10^{-6}$ as the sharpness k increases. Freeing the sensitive layers further does not deliver a reliable accuracy gain.

Method	Acc (%)	NAS
Uniform SRSC	94.06	0.176
OWL $k = 0.3$	94.11	0.183
OWL $k = 0.5$	94.31	0.167
OWL $k = 0.7$	94.03	0.180

E.4 Why a strong signal does not help

The signal is real and stable, yet it does not translate into an accuracy or sparsity advantage. Protecting the high-score layers does not reliably raise accuracy: freeing them further is inconsistent, with $k = 0.5$ ticking up within noise and $k = 0.7$ falling back toward the uniform baseline (Table 21), so those layers show no dependable benefit from being freed further. Suppressing the low-score layers does not lower spike activity: counts at matched ceilings are near-identical across the three methods. A plausible reason is that the score couples two properties an allocation would ideally handle separately—how much a layer matters for accuracy, and how much it fires—so a single high score does not isolate either. Whatever the cause, non-uniform allocation does not improve on a uniform coefficient across the settings we test, and we retain the uniform coefficient of Section 4.4.