STRIDE: AN EFFICIENT SNN TRAINING-TO-DEPLOYMENT FRAMEWORK ON REAL-WORLD VISION TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking Neural Networks (SNNs) have attracted increasing attention in recent years for their potential for high-efficiency and low-energy computation. The community largely agrees that realizing this potential depends on improvements in SNN training, including low-bit computing and sparse excitation, and adaptation to dedicated hardware platforms. However, evidence connecting SNN training to practical deployment benefits remains fragmented: algorithmic studies often use synaptic-operation-based analytical energy estimates, while hardware-focused studies often provide limited validation of deployed accuracy on practical tasks. In this paper, we bridge this gap with an efficient **S**NN **TRa**I**n**ing-to-**DE**ployment (STRIDE) framework. STRIDE combines low-bit weights with Integer-LIF neurons to represent activations as compact integer codes, which are unfolded into binary temporal spikes for native, event-driven execution on dedicated SNN hardware. During training, we apply quantization-aware distillation to optimize low-bit weights under task and teacher supervision without backpropagation through time. During deployment, we develop native 1-bit spike $\times$ 4-bit weight execution on a jointly developed 28-nm SNN accelerator, skipping redundant spike-weight computations when later spikes match the first-timestep reference. This work evaluates the effectiveness of STRIDE on three real-world vision tasks, achieving 61.95% mIoU on UDD6 segmentation, 31.71%/16.04% mAP_{50}/$\text{mAP}_{50:95}$ on UDD detection, and 67.88% top-1 accuracy on ImageNet classification. It outperforms the strongest evaluated compact-code baselines by 3.82, 1.05/0.03, and 1.28 percentage points, respectively, while the evaluated BPTT baseline requires $4.59\times$, $4.66\times$, and $3.03\times$ as much peak training memory. We also evaluate hardware energy consumption by deploying one frozen STRIDE-trained SpikingLETNet checkpoint across the accelerator, GPU, and CPU. Command-block measurements yield an accelerator energy of 2.15 mJ per inference, 97.64% and 99.96% lower than on the evaluated INT8 GPU and CPU, respectively. These results also show that hardware-based energy measurements are far more accurate and reliable than conventional operation-count estimates, which capture some relative energy trends but show significant differences across architectures. Our work provides a feasible SNN implementation for real-world vision tasks and opens the door to developing the training-to-deployment collaboration scheme of SNNs.

1 INTRODUCTION

Spiking Neural Networks (SNNs) offer an approach to energy-efficient computation through sparse binary spikes and event-driven processing (Roy et al., 2019). Realizing this potential requires coordinated model design and hardware execution; the trained model must expose numerical and temporal structure that the execution platform can exploit. Dedicated dataflows and arithmetic support determine how spike sparsity translates into reduced computation and data movement (Narayanan et al., 2020). Software-hardware collaboration is therefore essential to turning the computational properties of SNNs into practical energy savings.

Efficient SNN deployment requires three key factors. First, the model needs low-bit weights and neuronal representations that fit the memory and arithmetic constraints of resource-constrained hard-

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.

ware while retaining task accuracy. Quantizing both weights and neuronal states reduces their storage requirements, but also introduces errors that must be addressed during training (Yin et al., 2024; Liu et al., 2026). Second, the hardware needs execution mechanisms that exploit sparse spikes and temporal structure to reduce unnecessary computation and data movement (Narayanan et al., 2020). Third, the training representation must map consistently to the numerical formats and spike encoding supported by the target platform. This mapping connects the accuracy achieved during training with the computations performed at deployment.

Existing studies have made progress on different aspects of this challenge, including low-bit SNN training, train-to-deployment representation mapping, and specialized hardware execution. MINT reduces weight and membrane-state precision through shared-scale quantization (Yin et al., 2024), while Integer-LIF supports accurate object detection by combining integer-valued training with binary spike-driven inference (Luo et al., 2024). On the hardware side, SpinalFlow develops a dataflow that exploits spike sparsity and operand reuse, and characterizes efficiency across workload and sparsity configurations (Narayanan et al., 2020). HardF-SNN further connects hardware-friendly quantization with dedicated FPGA execution (Liu et al., 2026). These advances highlight the remaining challenge of jointly maintaining low-bit task accuracy, mapping trained representations to efficient spike execution, and establishing the resulting energy benefit on real hardware. Hardware-independent accuracy and operation counts provide only part of this evidence; system-level evaluation must also account for the deployed model and its execution platform (Yik et al., 2025).

We propose a software-hardware collaboration framework, named STRIDE, for accurate low-bit SNN execution on real-world vision tasks of semantic segmentation, object detection, and image classification. STRIDE combines low-bit weights with Integer-LIF neurons to represent activations as compact integer codes; its workflow comprises the STRIDE-T and STRIDE-D phases for compact-code training with Quantization-Aware Distillation (QAD) and hardware deployment with event-driven execution, respectively. The whole procedure is carried out under a W4A4 configuration, where training operates in the compact-code domain without Back-Propagation Through Time (BPTT), and deterministic unfolding generates binary temporal spikes for deployment. The hardware execution path further exploits similarity across timesteps by skipping redundant spike-weight computations when later spikes match the first-timestep reference.

We first evaluate STRIDE-T on three real-world vision tasks: UDD6 semantic segmentation with SpikingLETNet, UDD object detection with SpikingYOLOv5, and ImageNet classification with SpikingResNet-18. Under W4A4 constraints, the evaluated models achieve 61.95% mIoU, 31.71%/16.04% mAP_{50}/$\text{mAP}_{50:95}$, and 67.88% top-1 accuracy, respectively. Compared with the metric-wise strongest compact-code baselines, STRIDE-T improves segmentation mIoU by 3.82, detection mAP_{50}/$\text{mAP}_{50:95}$ by 1.05/0.03, and classification top-1 accuracy by 1.28 percentage points. The evaluated BPTT-based baseline requires $4.59\times$, $4.66\times$, and $3.03\times$ as much peak training memory on these tasks, respectively. These results show that the proposed STRIDE-T achieves higher task accuracy than all contenders while using substantially less peak training memory than the BPTT-based baseline.

We then evaluate the hardware-deployment efficiency of STRIDE-D by deploying one frozen W4A4 SpikingLETNet checkpoints trained with STRIDE-T on an SNN-aware accelerator. Specifically, the checkpoints are mapped to native W4A1T8 execution on a jointly developed 28-nm SNN accelerator and to compact-code FP32/INT8 workloads on a Blackwell GPU and a VNNI CPU. Practical evaluation shows a maximum mIoU decrease of 0.34 percentage points across the evaluated GPU and CPU configurations, while accelerator execution is validated through operator-level comparisons with a software reference. Under platform-specific measurement protocols, sampled command-block measurements yield the accelerator energy of 2.15 mJ per inference, 97.64% and 99.96% lower than those of the evaluated INT8 GPU and CPU. The above results not only confirm the deployment efficiency of STRIDE-D, but also highlight the importance of hardware-based evaluation.

The rest of this paper is organized as follows. Section 2 introduces related studies. Section 3 formally presents the STRIDE framework. Section 4 conducts experiments to verify the effectiveness of STRIDE in terms of task accuracy, training memory, hardware-deployment consistency, and cross-platform energy. Section 5 concludes the paper.

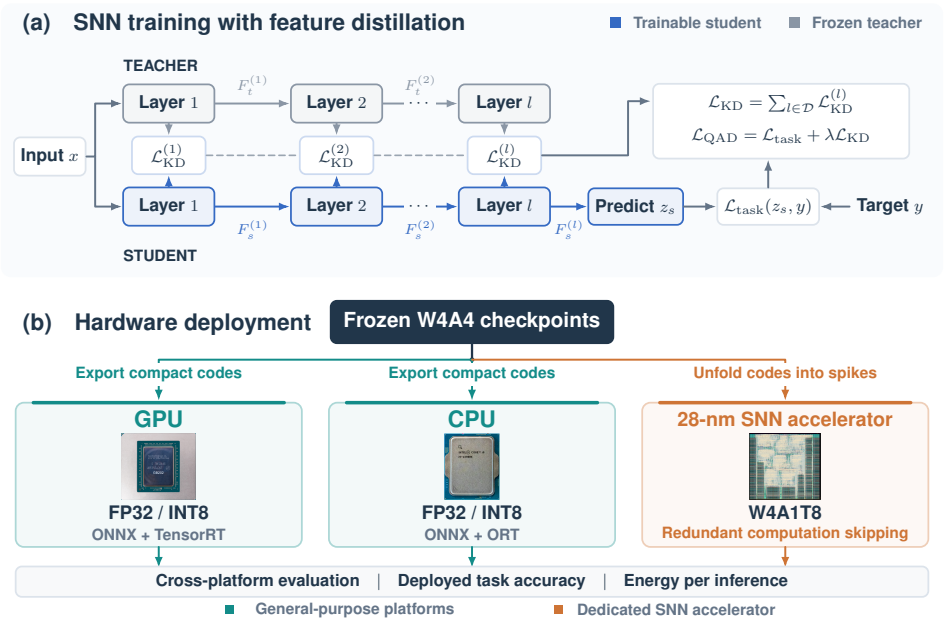

Figure 1: Overview of the STRIDE training-to-deployment framework. (a) STRIDE-T trains the Integer-LIF and low-bit SNNs with feature distillation in the compact neuronal-code domain. (b) STRIDE-D exports frozen W4A4 checkpoints as compact-code FP32/INT8 workloads for GPU/CPU execution and a W4A1T8 spike workload for the SNN accelerator.

2 Related Work

Low-bit SNN training. The technical progress on training low-bit SNNs mainly follows the quantization of deep learning. Quantization-aware training incorporates quantization into optimization (Jacob et al., 2018; Esser et al., 2020; Lee et al., 2021). Q-SNNs and SQUAT address membrane-state quantization (Wei et al., 2024; Venkatesh et al., 2024); MINT and HardF-SNN develop shared-scale quantization and integer-compatible operations (Yin et al., 2024; Liu et al., 2026). Recent advances in distillation targets show promise. For instance, QAD-SNN focuses on logits distillation (Lee et al., 2022a), QSD-Transformer uses fine-grained ANN-to-SNN supervision (Qiu et al., 2025), and MD-SNN supervises membrane potentials (Lee et al., 2026). Inspired by these advances, STRIDE incorporates feature distillation to better search for low-bit weights and activations in the Integer-LIF compact-code domain.

Efficient SNN execution and energy evidence. SpinalFlow reduces data movement through compressed spikes and specialized dataflow (Narayanan et al., 2020); parallel time batching improves reuse and systolic-array utilization across timesteps (Lee et al., 2022b). Their efficiency must be assessed at the appropriate level: NeuroBench distinguishes algorithm from hardware-dependent system evaluation (Yik et al., 2025). Analytical energy estimates often combine Multiply-ACcumulate (MAC) or Synaptic-OPeration (SOP) counts with reference per-operation energy costs. SpikeYOLO reports operation-count estimates (Luo et al., 2024), MD-SNN uses SpikeSim (Lee et al., 2026), and HardF-SNN evaluates FPGA deployment (Liu et al., 2026). These approaches differ in accounting for static power, neuronal-state updates, and data movement (Ostrau et al., 2022), motivating our comparison of analytical estimates with energy results across real architectures.

Deployment on resource-constrained edge devices. Edge deployment couples model design with execution constraints. MCUNet co-optimizes compact architectures and memory-efficient inference (Lin et al., 2020), while HAQ selects bit widths using hardware feedback (Wang et al., 2019). Hardware also limits numerical formats: MAX78000 supports 1-, 2-, 4-, and 8-bit weights on an 8-bit data path (Max, 2021). These ANN deployment examples motivate us to further explore low-bit SNN training and efficient deployment on dedicated accelerators.

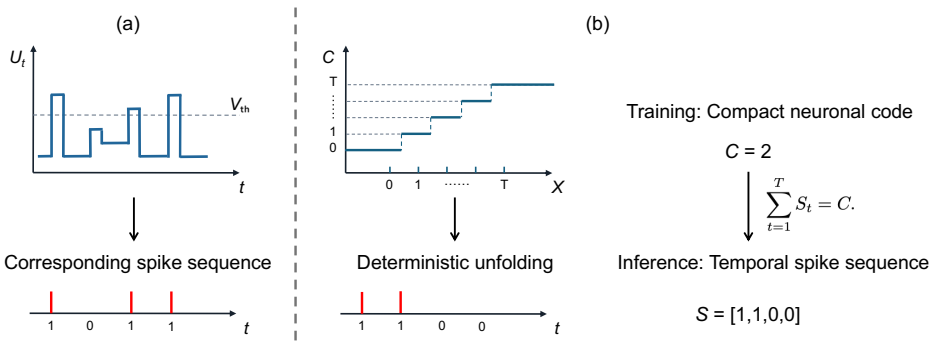

Figure 2: (a) Conventional LIF produces binary spikes by thresholding the membrane potential, whereas (b) Integer-LIF produces compact codes during training and unfolds them into temporal spikes at deployment.

3 METHOD

In this section, we formally present the collaboration workflow of the training and deployment phases of STRIDE, as illustrated in Figure 1. We establish the low-bit SNN with neuronal codes and Integer-LIF neurons in Section 3.1. Section 3.2 and Section 3.3 introduce the key techniques for training and deployment in our STRIDE framework, respectively.

3.1 LOW-BIT SNN MODELING IN COMPACT-CODE DOMAIN

We replace hidden-layer activations with Integer-LIF neurons (Luo et al., 2024) and quantize the model to the W4A4 format.

Activation quantization with Integer-LIF. In contrast to the conventional LIF neuron that produces binary spikes over recurrent timesteps and unrolls BPTT during training as detailed in Appendix A.1, the Integer-LIF neuron directly quantizes neuronal activations as bounded compact integer codes during training (Luo et al., 2024), thus enabling optimization without temporal unfolding. Figure 2 displays the differences between LIF and Integer-LIF neurons. We formulate the Integer-LIF neuron with bounded compact integer codes $C \in \{0, \ldots, T\}$, as follows

$$\begin{aligned} U &= H^{\text{in}} + X \ , \\ C &= \text{clip}(\text{round}(U),\, 0,\, T) \ , \\ H^{\text{out}} &= \beta U \odot (1 - \mathbb{I}[C > 0]) \ , \end{aligned} \tag{1}$$

where U is the pre-spike membrane potential, H^{in} and H^{out} separately are the input and output membrane states, X is the input current, $\beta \in (0, 1)$ is the decay factor, and $\odot$ denotes element-wise multiplication. $\mathbb{I}[\cdot]$ denotes the indicator function, which returns 1 when the condition is true and 0 otherwise. The indicator acts element-wise on C; hence, a nonzero code resets the corresponding state. Rounding and clipping U restrict the neuronal activation to the integer code range, implementing activation quantization within Integer-LIF. It is common to set $T = 2^{b-1}$ for b-bit neuronal codes; this work adopts the 4-bit configuration, including $T = 8$ and $C \in \{0, \ldots, 8\}$.

Notice that we keep activations in this compact-code form during training, while unfolding each code at deployment as

$$S_t = \mathbb{I}[t \le C] \ , \quad t = 1, \ldots, T \ , \quad \sum_{t=1}^{T} S_t = C \ , \tag{2}$$

where t indicates the deployment timestep and S_t is a binary spike. The expansion preserves each code's value and produces eight binary timesteps for $b = 4$. Together with 4-bit weights, this maps the W4A4 training representation to W4A1T8 on the accelerator.

Low-bit weights. Here, we equip the model with low-bit weights. The forward pass uses uniform fake quantization based on latent floating-point weights $W^{(l)}$ at the l-th layer as follows

$$\widehat{W}^{(l)} = s_l \, \mathrm{clip}\left(\mathrm{round}\left(\frac{W^{(l)}}{s_l} \right), q^w_{\min}, q^w_{\max} \right) , \tag{3}$$

where $\widehat{W}^{(l)}$ denotes the quantized weights in the 4-bit format, s_l is the quantization scale, and $[q^w_{\min}, q^w_{\max}]$ signs the 4-bit code range. Appendix A.3 provides detailed quantization configurations.

3.2 STRIDE-T: Training with Quantization-Aware Distillation

This subsection introduces the training of STRIDE, where we exploit QAD to enable low-bit optimization and accuracy maintenance under W4A4 constraints. As shown in Figure 1(a), both task supervision and teacher features guide this optimization procedure in the compact-code domain.

Distillation setup and objectives. The frozen teacher f_T shares the student's architecture and Integer-LIF code domain but retains FP32 weights, permitting direct alignment of corresponding features. The student uses the compact codes C and quantized weights $\widehat{W}^{(l)}$ defined in Section 3.1. Let z_s denote student predictions, and y indicates task targets. The proposed STRIDE-T minimizes the QAD objective

$$\mathcal{L}_{\mathrm{QAD}} = \mathcal{L}_{\mathrm{task}}(z_s, y) + \lambda \mathcal{L}_{\mathrm{KD}} , \tag{4}$$

where $\mathcal{L}_{\mathrm{task}}$ is the supervised loss for a certain vision task, such as segmentation, classification, or detection, and λ denotes the distillation coefficient.

Notice that the distillation loss $\mathcal{L}_{\mathrm{KD}}$ is defined according to the task and model architecture. For semantic segmentation with LETNet (Xu et al., 2023) and image classification with ResNet-18 (He et al., 2016), we align teacher and student features $F_t^{(l)}, F_s^{(l)}$ at locations $\mathcal{D}$ as follows

$$\mathcal{L}_{\mathrm{KD}} = \sum_{l \in \mathcal{D}} \mathcal{L}^{(l)}_{\mathrm{KD}} \quad \text{with} \quad \mathcal{L}^{(l)}_{\mathrm{KD}} = \frac{1}{N_l} \left\| F_s^{(l)} - F_t^{(l)} \right\|_2^2 . \tag{5}$$

For each selected feature pair, $\mathcal{L}^{(l)}_{\mathrm{KD}}$ is averaged over N_l elements, and the losses are summed across locations. For object detection with YOLOv5 (Jocher, 2022), we employ $\mathcal{L}_{\mathrm{KD}} = \mathcal{L}^{\mathrm{det}}_{\mathrm{KD}} + 0.5\,\mathcal{L}^{\mathrm{feat}}_{\mathrm{KD}}$. The detection-head distillation loss $\mathcal{L}^{\mathrm{det}}_{\mathrm{KD}}$ matches teacher and student objectness, class, and bounding-box predictions, while the normalized intermediate-feature distillation loss $\mathcal{L}^{\mathrm{feat}}_{\mathrm{KD}}$ aligns their backbone representations. Task losses, feature locations, normalization and aggregation rules, and distillation weights and schedules are specified for each architecture in Appendix A.3.

Optimization without temporal unfolding. The forward pass uses the quantized weights $\widehat{W}^{(l)}$ and compact neuronal codes C, while optimization updates the latent floating-point weights $W^{(l)}$. Gradients through rounding use a straight-through estimator (Bengio et al., 2013). The binary sequence $S_1, \ldots, S_T$ in Eq. (2) is generated only for deployment and is not unfolded during training. STRIDE-T therefore optimizes the student in the compact-code domain without backpropagation through the T deployment timesteps. Conventional SNN quantization training in the temporal domain uses BPTT over T recurrent timesteps, retaining intermediate states for gradient computation. Section 4.2 compares STRIDE-T optimization in the compact-code domain with BPTT-based optimization in the temporal domain under the same W4A4 constraints, in order to evaluate the effectiveness of STRIDE-T in terms of both task accuracy and peak training memory.

3.3 STRIDE-D: Hardware Deployment with Event-Driven Execution

The trained W4A4 model represents activations as compact integer codes. Deployment on the target SNN accelerator requires converting these codes into binary spikes while preserving their values. In the resulting binary spike sequences, sparsity yields high similarity across timesteps, which the hardware can exploit to reduce redundant spike-weight computation. We combine code unfolding with computation skipping to support native low-bit, event-driven execution.

Code unfolding and workload mapping. We deploy the frozen W4A4 SpikingLETNet checkpoint on a jointly developed, synchronously clocked 28-nm SNN accelerator. Eq. (2) unfolds each compact code C into $T = 8$ binary spikes, yielding a W4A1T8 workload. The accelerator's native 1-bit spike $\times$ 4-bit weight path performs weight-gated accumulation. The host partitions each layer into command blocks and schedules them across the compute cores. Figure 1(b) summarizes the deployment paths, including the compact-code FP32/INT8 workloads derived from the same checkpoint for GPU and CPU execution. Workload export, command-block mapping, and backend configurations are detailed in Appendix B.1, while the physical layout of an accelerator tile is shown in Appendix C.1 (Figure 4).

Redundant computation skipping. At each mapped position, the weight remains fixed across timesteps, so identical spike values produce the same spike-weight term. The accelerator uses the first spike at that position as a fixed reference. It computes all spike-weight terms at $t = 1$ and recomputes a later term only when its spike differs from the reference:

$$G_{i,t} = \begin{cases} 1\,, & t = 1\,, \\ \mathbb{I}[S_{i,t} \neq S_{i,1}]\,, & t = 2, \ldots, T\,, \end{cases} \tag{6}$$

where $S_{i,t}$ indicates the spike at mapped position i and timestep t. The gate $G_{i,t} = 1$ denotes a computed spike-weight term, and $G_{i,t} = 0$ denotes a skipped recomputation. Without exploiting spike sparsity, evaluating N spike positions over T timesteps requires NT spike-weight computations. Our STRIDE-D reduces this count to

$$N_{\text{eval}} = N + \sum_{t=2}^{T} \sum_{i=1}^{N} \mathbb{I}[S_{i,t} \neq S_{i,1}]\ . \tag{7}$$

Positions that remain silent require only the first-timestep evaluation. More generally, agreement with the fixed reference reduces repeated computation across timesteps. This mechanism translates the temporal structure of the spike workload into event-driven computation on synchronously clocked hardware.

4 Experiments

We evaluate STRIDE on real-world segmentation, detection, and classification tasks. We first assess STRIDE-T in terms of task accuracy and training memory, then evaluate STRIDE-D through deployment consistency and energy comparisons across the SNN accelerator, GPU, and CPU.

4.1 Configurations

Tasks and models. We evaluate STRIDE on three real-world vision tasks. The first is semantic segmentation on the six-class Urban Drone Dataset (UDD6) (Chen et al., 2018), with 25,700 training and 8,478 validation patches. The second is object detection on the 10-class UDD detection dataset. The third is image classification on ImageNet-1K (Deng et al., 2009; Russakovsky et al., 2015), which contains 1,281,167 training images and 50,000 validation images across 1,000 classes. For these tasks, we apply STRIDE to LETNet, YOLOv5, and ResNet-18, yielding SpikingLETNet, SpikingYOLOv5, and SpikingResNet-18, respectively. These three models maintain 1.30 M, 1.78 M, and 11.69 M parameters, respectively. SpikingYOLOv5 is pretrained on VisDrone2019-DET (Zhu et al., 2021) and then fine-tuned on UDD before applying QAD within STRIDE-T. Input sizes and preprocessing are detailed in Appendix A.2, and architecture-specific training settings are provided in Appendix A.3.

Deployment and evaluation protocol. Deployment uses one frozen W4A4 SpikingLETNet checkpoint trained with STRIDE-T. The SNN accelerator executes W4A1T8, whereas TensorRT on a Blackwell GPU and ONNX Runtime on a VNNI CPU execute compact-code FP32/INT8 workloads. The general-purpose workloads are exported from PyTorch to ONNX and verified against the PyTorch reference. The FP32 and explicit-Q/DQ INT8 are the two end-to-end verified configurations on both backends, which support INT8 execution of weights and activations (Corporation, 2026; 2023; Microsoft, 2025). We report deployed mIoU and energy per inference. GPU and CPU

Table 1: Comparison between STRIDE-T and its contenders on UDD6 segmentation, UDD detection, and ImageNet-1K classification, where "Memory" denotes peak GPU memory measured during training with a batch size of 4, and bold values indicate the best result in each metric column.

	UDD6 / SpikingLETNet		UDD / SpikingYOLOv5			ImageNet / SpikingResNet18		
Method	mIoU (%)↑	Memory (GiB)↓	mAP_{50} (%)↑	$mAP_{50:95}$ (%)↑	Memory (GiB)↓	Top-1 (%)↑	Top-5 (%)↑	Memory (GiB)↓
Training in compact-code domains, without BPTT								
STE-QAT	58.13	**2.48**	30.60	14.20	0.802	64.18	85.59	**0.347**
LSQ-QAT	53.79	4.21	30.54	16.01	**0.792**	64.83	85.95	0.393
EWGS-QAT	47.28	4.86	30.66	15.25	0.804	66.60	87.01	1.772
STRIDE-T (ours)	**61.95**	3.05	**31.71**	**16.04**	1.029	**67.88**	**87.87**	0.432
Training in temporal domains, with BPTT								
LIF-SQUAT	31.55	14.01	0.24	0.04	4.800	53.81	78.32	1.307

energy is measured over repeated executions after warm-up, subtracting the idle-power baseline. Accelerator energy is obtained by extrapolating sampled command-block measurements to layer totals and summing across layers. Analytical estimates combine MAC or spike-driven SOP counts with the 45-nm, 0.9-V arithmetic energy costs of Horowitz (2014), with INT4 costs extrapolated by bit-width scaling. Workload export and backend configurations, measurement boundaries, detailed protocols, and analytical-estimation procedures are provided in Appendix B.1, Appendix B.3, and Appendix C.2, respectively.

4.2 Verification on STRIDE-T

STRIDE-T versus compact-code baselines. Table 1 compares STRIDE-T, which uses QAD, with STE-QAT (Jacob et al., 2018), LSQ (Esser et al., 2020), and EWGS (Lee et al., 2021) under W4A4 constraints. STRIDE-T reaches 61.95% mIoU on UDD6 and 67.88% top-1 accuracy on ImageNet, improving on the strongest compact-code baseline by 3.82 and 1.28 percentage points, respectively. On UDD detection, STRIDE-T achieves 31.71% mAP_{50} and 16.04% $mAP_{50:95}$, exceeding EWGS-QAT by 1.05 percentage points in mAP_{50} and LSQ-QAT by 0.03 percentage points in $mAP_{50:95}$, the strongest compact-code baselines for the respective metrics.

Compact-code versus temporal-domain quantization. LIF-SQUAT (Venkatesh et al., 2024) uses the same W4A4 constraint but unrolls eight recurrent timesteps during training. Table 1 also compares the compact-code methods with BPTT-based LIF-SQUAT. On UDD6, it trails STRIDE-T by 30.4 mIoU points and requires $4.59\times$ its peak training memory. On UDD detection, LIF-SQUAT achieves 0.24% mAP_{50} and 0.04% $mAP_{50:95}$, trailing STRIDE-T by 31.47 and 16.00 percentage points, respectively; its peak training memory is 4.800 GiB, or $4.66\times$ that of STRIDE-T. On ImageNet, LIF-SQUAT trails STRIDE-T by 14.07 percentage points in top-1 accuracy and 9.55 percentage points in top-5 accuracy, while requiring $3.03\times$ its peak training memory. Across all three tasks, STRIDE-T achieves higher task accuracy and lower peak training memory than the evaluated timestep-unrolled BPTT baseline.

Task accuracy across bit-width configurations. Table 2 compares task accuracy across the evaluated bit-width configurations. The W32A32 baseline achieves 67.85% mIoU on UDD6 segmentation, 34.69%/19.01% mAP_{50}/$mAP_{50:95}$ on UDD detection, and 70.11%/89.15% top-1/top-5 accuracy on ImageNet. Moving from W8A8 to W4A4 reduces segmentation mIoU from 65.83% to 61.95%, detection mAP_{50}/$mAP_{50:95}$ from 32.06%/16.13% to 31.71%/16.04%, and ImageNet top-1/top-5 accuracy from 69.95%/89.17% to 67.88%/87.87%. These changes correspond to decreases of 3.88, 0.35/0.09, and 2.07/1.30 percentage points, respectively. Further reducing precision to W1.58A1.58 causes substantially larger losses across all three tasks: segmentation mIoU falls to 22.33%, detection mAP_{50}/$mAP_{50:95}$ to 5.87%/1.86%, and ImageNet top-1/top-5 accuracy to 1.41%/4.28%. Among the evaluated configurations, these results support W4A4 as the deployment operating point, reducing weight and neuronal-code precision relative to W8A8 while avoiding the severe accuracy degradation observed with W1.58 quantization.

Table 2: Task performance under different bit-width configurations on UDD6 with SpikingLETNet, on UDD-detection with SpikingYOLOv5, and on ImageNet with SpikingResNet18.

	UDD6 / SpikingLETNet	UDD-detection / SpikingYOLOv5		ImageNet / SpikingResNet18	
W/code format	mIoU (%)↑	mAP_{50} (%)↑	$mAP_{50:95}$ (%)↑	Top-1 (%)↑	Top-5 (%)↑
W32A32	67.85	34.69	19.01	70.11	89.15
W8A8	65.83	32.06	16.13	69.95	89.17
W4A4	61.95	31.71	16.04	67.88	87.87
W1.58A1.58	22.33	5.87	1.86	1.41	4.28

Table 3: System-level deployment results for the same trained checkpoint under platform-specific measurement protocols, together with the analytical estimates of Eq. (13). Bold values indicate the best result in each metric column.

Platform/Measurement	Backend	Representation	Precision	Δ mIoU (pp)	Energy per inference ↓
28-nm SNN platform (ours)	Custom Host Runtime	Temporal spikes	W4A1T8	-0.00†	**2.15** mJ
Blackwell RTX PRO 6000	TensorRT	Compact codes	W32A32	-0.02	175.58 mJ
Blackwell RTX PRO 6000	TensorRT	Compact codes	W8A8	-0.34	91.06 mJ
VNNI i9-14900K	ONNX Runtime	Compact codes	W32A32	-0.00	61.24 J
VNNI i9-14900K	ONNX Runtime	Compact codes	W8A8	-0.32	5.24 J
SOP estimate	-	Temporal spikes	W32A32	-	13.05 mJ
SOP estimate	-	Temporal spikes	W4A1T8	-	**0.38** mJ
MAC estimate	-	Compact codes	W32A32	-	33.83 mJ
MAC estimate	-	Compact codes	W8A8	-	1.69 mJ
MAC estimate	-	Compact codes	W4A4	-	0.59 mJ

†Expected mIoU preservation based on operator-level computational consistency (Appendix B.2).

4.3 VERIFICATION ON STRIDE-D

Checkpoint deployment consistency. Table 3 summarizes changes in deployment mIoU relative to the PyTorch reference. For GPU and CPU deployments, Δ mIoU is obtained by directly evaluating each backend against the trained checkpoint. The largest measured deviation is 0.34 percentage points on the INT8 GPU, indicating tiny task-accuracy degradation across the evaluated GPU and CPU deployments. For the SNN accelerator, computational consistency is assessed through operator-level comparisons between accelerator outputs and a PyTorch reference under the deployment numerical configuration. The test protocol is reported in Appendix B.2.

Measured accelerator energy across platforms. Table 3 also reports energy per inference for the GPU, CPU, and SNN accelerator deployments under the evaluated numerical precision configurations. Within each general-purpose platform, moving from FP32 to INT8 reduces GPU energy per inference from 175.58 to 91.06 mJ, saving 48.14% ($1.93\times$ reduction), and CPU energy from 61.24 to 5.24 J, saving 91.44% ($11.69\times$ reduction). Although both platforms benefit from INT8, the much larger relative saving on the CPU shows that the energy benefit of the same precision reduction depends on the execution stack.

For the cross-platform comparison, we use the more energy-efficient INT8 configurations of the GPU and CPU as references. The W4A1T8 SNN accelerator requires an estimated 2.15 mJ per inference while the INT8 GPU and CPU consume $42.35\times$ and $2{,}437\times$ as much energy, respectively, corresponding to 97.64% and 99.96% lower energy for the accelerator. Thus, even relative to the INT8 implementations, deploying the trained low-bit SNN on the dedicated accelerator provides a substantial energy advantage.

Analytical estimates versus measured energy. At matched representation-precision points in Figure 3, the SOP/MAC-based analytical estimates underpredict the corresponding measured energy by $5.66\times$ for the W4A1T8 accelerator, $5.19\times$ for the FP32 GPU, and 1.81×10^3 for the FP32 CPU. These discrepancies show that the operation count alone fails to predict the measured energy consistently across execution architectures.

However, analytical estimates can still capture some relative energy trends across platforms and precision configurations. Both the estimates and measurements show lower energy when moving

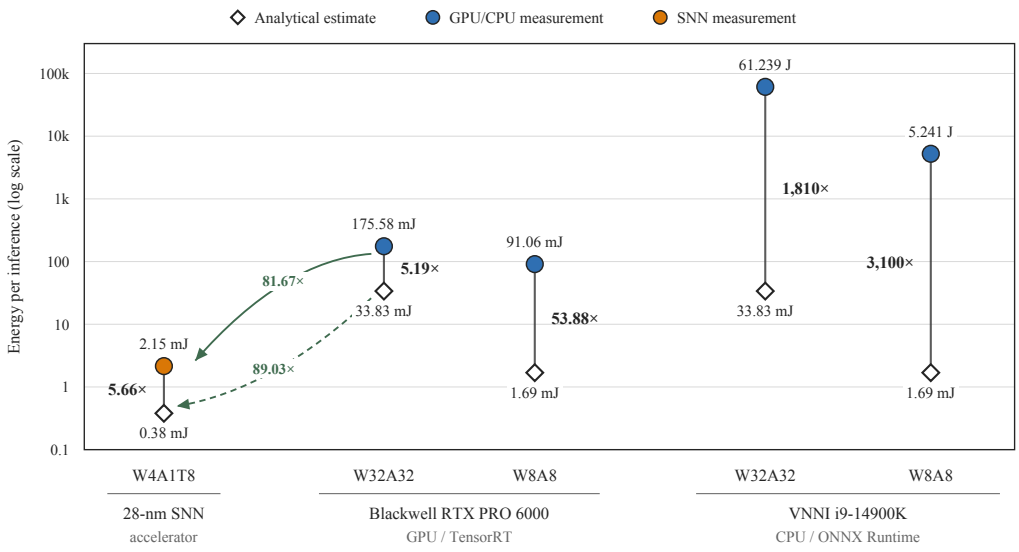

Figure 3: Measured accelerator energy versus analytical estimates across architectures and precisions. Vertical connectors pair each measured energy with its corresponding analytical estimate, while adjacent labels report their ratio.

from FP32 to INT8 on the GPU and CPU. For the transition from the FP32 GPU deployment to the W4A1T8 SNN accelerator, the analytical estimate predicts an $89.03\times$ energy reduction, compared with the $81.67\times$ reduction from hardware-based evaluation. The analytical ratio is only 9.0% higher, showing that, in this case, analytical estimates capture both the direction and approximate magnitude of the relative energy benefit despite discrepancies in absolute energy.

Discussion. The deployment results demonstrate the energy efficiency of STRIDE-D for the evaluated SpikingLETNet checkpoint. The measured accelerator energy is 2.15 mJ per inference, 97.64% and 99.96% below the evaluated INT8 GPU and CPU deployments, respectively. The measured energy reduction from FP32 GPU execution to native W4A1T8 execution on the SNN accelerator is $81.67\times$, close to the $89.03\times$ reduction predicted by the analytical estimates. By combining bit-width-dependent arithmetic costs with firing-rate-weighted SOP counts, these estimates describe an idealized low-bit, spike-driven workload. The close agreement suggests that the SNN accelerator effectively exploits the advantages of low-bit, event-driven execution.

Incontestably, the accuracy of the energy estimates varies substantially across execution platforms. The estimates underpredict energy by $5.66\times$ on the accelerator and $5.19\times$ on the FP32 GPU. On the FP32 CPU, the underprediction is far larger, reaching 1.81×10^3. These results appear because these analytical estimates do not account for memory access, data movement, or execution overhead that may contribute to the discrepancies. Thus, analytical estimates can help characterize relative trends, while realistic hardware-based implementation and measurement are in need to establish deployment energy.

5 Conclusions

We presented the STRIDE framework that collaborates STRIDE-T for optimizing W4A4 SNNs and STRIDE-D for native event-driven execution on a dedicated accelerator. The former employs QAD for SNN training in the compact-code domain, avoiding temporal backpropagation, while the latter enables efficient low-bit execution by skipping redundant spike-weight computations. Empirical results show that STRIDE-T achieves higher task accuracy with lower peak training memory than the BPTT baseline across all evaluated tasks. For SpikingLETNet deployed with STRIDE-D, the measured accelerator energy is 2.15 mJ per inference; in contrast, the evaluated INT8 GPU and CPU consume $42.35\times$ and $2{,}437\times$ as much, respectively.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript translation and polishing. We have not used generative AI tools for the following tasks, including generating synthetic data sets, developing theoretical models or conceptual frameworks, designing or providing feedback on research methodology or experiments, cleaning and reformatting datasets, and interpreting results. The following tasks do not apply to this work: Formulate mathematical claims; provide critical ingredients for proving mathematical claims; assist in the writing of proofs; propose or refine hypotheses; support qualitative and thematic data analysis. We have reviewed all AI-assisted work; even the AI-assisted translation was checked by the authors to ensure that it preserved the intended meaning. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

Yu Chen, Yao Wang, Peng Lu, Yisong Chen, and Guoping Wang. Large-scale structure from motion with semantic constraints of aerial images. In *Proceedings of the Chinese Conference on Pattern Recognition and Computer Vision*, pp. 347–359, 2018.

Intel Corporation. Intel core i9-14900k processor specifications. `https://www.intel.com/content/www/us/en/products/sku/236773/intel-core-i9-processor-14900k-36m-cache-up-to-6-00-ghz/specifications.html`, 2023.

NVIDIA Corporation. Quantization schemes—NVIDIA TensorRT 11.2.1. `https://docs.nvidia.com/deeplearning/tensorrt/11.2.1/inference-library/quantized-types-schemes.html`, 2026.

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

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Proceedings of IEEE International Solid-State Circuits Conference Digest of technical papers*, pp. 10–14, 2014.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Glenn Jocher. YOLOv5 by Ultralytics. `https://github.com/ultralytics/yolov5`, 2022.

Donghyun Lee, Guoqi Li, Hongsik Jeong, and Dong-Hyeok Lim. Qad-snn: Quantization-aware distillation on spiking neural network. In *Proceedings of Korea Artificial Intelligence Conference (KAIC)*, 2022a.

Donghyun Lee, Abhishek Moitra, Youngeun Kim, Ruokai Yin, and Priyadarshini Panda. Md-snn: Membrane potential-aware distillation on quantized spiking neural network. In *Proceedings of Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2026.

Jeong-Jun Lee, Wenrui Zhang, and Peng Li. Parallel time batching: Systolic-array acceleration of sparse spiking neural computation. In *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 317–330, 2022b.

Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling, 2021.

Ji Lin, Wei-Ming Chen, Yujun Lin, john cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In *Advances in Neural Information Processing Systems*, volume 33, pp. 11711–11722, 2020.

Hanwen Liu, Kexin Shi, Jieyuan Zhang, Yimeng Shan, Jibin Wu, Wenyu Chen, and Malu Zhang. Hardf-snn: Hardware-friendly quantization for spiking neural networks with efficient integer-arithmetic-only inference. In *Proceedings of AAAI Conference on Artificial Intelligence*, volume 40, pp. 1937–1945, 2026.

Xinhao Luo, Man Yao, Yuhong Chou, Bo Xu, and Guoqi Li. Integer-valued training and spike-driven inference spiking neural network for high-performance and energy-efficient object detection. In *European Conference on Computer Vision*, pp. 253–272. Springer, 2024.

MAX78000: Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator. Maxim Integrated, 2021. URL `https://www.analog.com/media/en/technical-documentation/data-sheets/max78000.pdf`. Data Sheet, Rev. 1.

Microsoft. Quantize ONNX models. `https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html`, 2025.

Surya Narayanan, Karl Taht, Rajeev Balasubramonian, Edouard Giacomin, and Pierre-Emmanuel Gaillardon. Spinalflow: An architecture and dataflow tailored for spiking neural networks. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pp. 349–362, 2020.

Christoph Ostrau, Christian Klarhorst, Michael Thies, and Ulrich Rückert. Benchmarking neuromorphic hardware and its energy expenditure. *Frontiers in Neuroscience*, 16:873935, 2022.

Xuerui Qiu, Malu Zhang, Jieyuan Zhang, Wenjie Wei, Honglin Cao, Junsheng Guo, Rui-Jie Zhu, Yimeng Shan, Yang Yang, and Haizhou Li. Quantized spike-driven transformer, 2025.

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Sreyes Venkatesh, Razvan Marinescu, and Jason K Eshraghian. Squat: Stateful quantization-aware training in recurrent spiking neural networks. In *Proceedings of Neuro Inspired Computational Elements Conference*, pp. 1–10, 2024.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8604–8612. IEEE, 2019.

Wenjie Wei, Yu Liang, Ammar Belatreche, Yichen Xiao, Honglin Cao, Zhenbang Ren, Guoqing Wang, Malu Zhang, and Yang Yang. Q-snns: Quantized spiking neural networks. In *Proceedings of ACM International Conference on Multimedia*, pp. 8441–8450, 2024.

Guoan Xu, Juncheng Li, Guangwei Gao, Huimin Lu, Jian Yang, and Dong Yue. Lightweight real-time semantic segmentation network with efficient transformer and cnn. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):15897–15906, 2023.

Jason Yik, Korneel Van den Berghe, Douwe Den Blanken, Younes Bouhadjar, Maxime Fabre, Paul Hueber, Weijie Ke, Mina A Khoei, Denis Kleyko, Noah Pacik-Nelson, et al. The neurobench framework for benchmarking neuromorphic computing algorithms and systems. *Nature Communications*, 16(1):1545, 2025.

Ruokai Yin, Yuhang Li, Abhishek Moitra, and Priyadarshini Panda. Mint: Multiplier-less integer quantization for energy efficient spiking neural networks. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 830–835, 2024.

Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. Detection and tracking meet drones challenge. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 44(11):7380–7399, 2021.

A Training Details

A.1 Conventional LIF Dynamics

The discrete-time dynamics of a standard Leaky Integrate-and-Fire (LIF) neuron with hard reset are

$$\begin{aligned} U_t &= H_{t-1} + X_t \ , \\ S_t &= \Theta(U_t - V_{\mathrm{th}}) \ , \\ H_t &= \beta U_t \odot (1 - S_t) \ , \end{aligned} \tag{8}$$

where t denotes the timestep, U_t the pre-spike membrane potential, $\Theta(\cdot)$ the Heaviside function, and V_{th} the firing threshold. Each element of S_t is a binary spike. The membrane state H_t is reset when a spike occurs; otherwise, it decays by $\beta \in (0, 1)$ and is carried to the next timestep (Figure 2(a)).

A.2 Datasets and Preprocessing

UDD6 segmentation. UDD6 contains six semantic classes and uses 106 training and 35 validation images (Chen et al., 2018), processed into 25,700 training and 8,478 validation patches. SpikingLETNet uses 400×400 inputs. Training applies horizontal flips, vertical flips, and 90° rotations, each with probability 0.5, followed by resizing. Inputs use ImageNet mean and standard-deviation normalization.

UDD detection. The UDD detection dataset contains 10 object categories, with 94 training, 10 validation, and 12 test images. SpikingYOLOv5 uses 832×832 training inputs and is pretrained on VisDrone2019-DET before transfer learning on UDD and subsequent QAD.

ImageNet-1K classification. ImageNet-1K contains 1,281,167 training and 50,000 validation images across 1,000 classes. Training uses random resized crops of 224×224 pixels and horizontal flipping. Validation resizes the shorter image side to 256 pixels and applies a 224×224 center crop. Inputs use ImageNet mean and standard-deviation normalization.

A.3 Per-Architecture Distillation Settings

SpikingLETNet and SpikingResNet-18 use the feature-distillation loss in Eq. (5) under the objective in Eq. (4), with frozen FP32-weight teachers in the same Integer-LIF code domain. Teacher features are detached from gradient computation. Each mean-squared-error term averages over all feature elements, including the batch dimension, and the six terms are summed. Both use a fixed $\lambda = 0.1$ with no distillation warm-up or decay; neither enables logit distillation.

SpikingLETNet. The task loss is unweighted pixel-wise cross-entropy. Features are extracted after encoder DAB blocks 1 and 2, after encoder DAB block 3 before the transformer, and after decoder DAB blocks 4-6 and their associated upsampling operations. The sixth location includes the final convolution after the third upsampling operation. For 400×400 inputs, the six features returned by `forward_qat` have shapes $[T_{\mathrm{train}}, B, 128, 100, 100]$, $[T_{\mathrm{train}}, B, 128, 50, 50]$, $[B, 64, 25, 25]$, $[T_{\mathrm{train}}, B, 32, 50, 50]$, $[T_{\mathrm{train}}, B, 32, 100, 100]$, and $[T_{\mathrm{train}}, B, 32, 200, 200]$, where $T_{\mathrm{train}} = 1$ is the training input time dimension. The five-dimensional features are averaged over this dimension before computing the MSE terms.

The recorded QAD runs use Adam with learning rate 10^{-3}, betas $(0.9, 0.999)$, epsilon 10^{-8}, weight decay 10^{-4}, and batch size 64. Polynomial learning-rate decay uses power 0.9 and a 150-epoch horizon without warm-up; the runs stop after 127 epochs. Training uses 401 full batches per epoch, and validation uses 8,478 patches with batch size 20. The recorded seeds are 1234, 2345, and 3456. Automatic mixed precision is disabled, and the best validation-mIoU checkpoint is retained.

SpikingResNet-18. The task loss is cross-entropy with label smoothing 0.1. The six feature locations are the stem after its QIF neuron and max pooling, the outputs of `layer1`-`layer4`, and the globally pooled feature before the classifier. Their shapes are $[B, 64, 56, 56]$, $[B, 64, 56, 56]$, $[B, 128, 28, 28]$, $[B, 256, 14, 14]$, $[B, 512, 7, 7]$, and $[B, 512]$.

SpikingYOLOv5. The detector uses three output scales, P3, P4, and P5. Its implementation combines the supervised detection loss $\mathcal{L}_{\text{det}}$ with detection-head and feature distillation, using $\lambda = s(e)$ in Eq. (4):

$$\mathcal{L}_{\text{QAD}}^{\text{det}} = \mathcal{L}_{\text{det}} + s(e)\left(\mathcal{L}_{\text{KD}}^{\text{det}} + 0.5\,\mathcal{L}_{\text{KD}}^{\text{feat}}\right) , \tag{9}$$

$$s(e) = \begin{cases} 0 , & e < 5 , \\ \dfrac{0.35}{2}\left[1 - \cos\left(\dfrac{\pi(e-5)}{55}\right)\right] , & 5 \le e < 60 , \\ 0.35 , & e \ge 60 , \end{cases} \tag{10}$$

where e is the zero-based epoch index in the 100-epoch run. At full weight, the effective coefficients are 0.35 for detection-head distillation and 0.175 for feature distillation. Over each detection scale, $\mathcal{L}_{\text{KD}}^{\text{det}}$ sums objectness binary cross-entropy against the teacher's sigmoid objectness (weight 2.0), class KL divergence from teacher to student with temperature $\tau = 2$ and the usual τ^2 multiplier (weight 1.0), and a smooth-ℓ_1 loss between sigmoid-transformed box outputs (weight 0.2). The class term is restricted to anchors whose teacher objectness exceeds 0.05.

Feature distillation uses the outputs of the first four backbone modules: the stride-2 6×6 convolution, the stride-2 3×3 convolution, the C3 block, and the following stride-2 3×3 convolution. Each student and teacher feature is divided by its own mean absolute value, with the normalization scale computed from a detached tensor. The loss averages the four element-wise mean-squared errors:

$$\mathcal{L}_{\text{KD}}^{\text{feat}} = \frac{1}{4}\sum_{l=1}^{4} \text{MSE}_{\text{mean}}\left(\widetilde{F}_s^{(l)}, \widetilde{F}_t^{(l)}\right) . \tag{11}$$

The teacher is a frozen copy of the UDD transfer checkpoint. Teacher batch-normalization layers use evaluation mode, while its detection head retains training mode to expose raw outputs. The student uses 56 quantized wrappers with per-tensor scales. Quantized student parameters are reconstructed from floating-point master parameters every iteration.

The UDD detection task uses 10 categories and a split of 94 training, 10 validation, and 12 test images. QAD uses 832×832 inputs, batch size 16, and 100 epochs. SGD uses initial learning rate 0.01, final learning-rate factor 0.01, momentum 0.937, weight decay 5×10^{-4}, a linear learning-rate schedule, and three warm-up epochs. The supervised detection-loss gains are 0.05 for boxes, 0.5 for classes, and 1.0 for objectness. The recorded seeds are 0, 1234, 2345, and 3456.

B Deployment Details

B.1 Workload Export and Backend Mapping

All deployments originate from the same frozen W4A4 checkpoint trained with QAD, but each backend consumes a representation matched to its execution model. For the accelerator, each code $C \in \{0, \ldots, 8\}$ is unfolded as $S_t = \mathbb{I}[t \le C]$ to form a value-preserving W4A1T8 workload. The host schedules this workload layer by layer: it packs the activation, weight, and operator descriptor, partitions each layer into command blocks, and distributes the blocks across eight tiles and 64 compute cores. Appendix C.1 details this mapping (Figure 5). The native path consumes 1-bit spikes and 4-bit weights and applies the temporal computation skipping in Section 3.3: all spike-weight terms are computed at $t = 1$, whereas a later computation is performed only if its spike differs from the first-timestep reference.

GPU and CPU deployments execute the same pair of ONNX models: an FP32 compact-code graph and its INT8 derivative. The INT8 model is obtained by static post-training quantization with MinMax calibration on 300 UDD6 training patches, per-channel weight scales, and per-tensor activation scales. It is stored as an S8S8 Q/DQ graph in which convolution, transposed-convolution, matrix-multiplication, and Gemm operators are quantized.

The GPU is an NVIDIA RTX PRO 6000 Blackwell Server Edition with driver 580.95.05 and a 600 W power limit, running TensorRT 11.1.0.106 with CUDA 12.8. Both engines are built as strongly typed networks with TF32 disabled and a 4 GiB workspace. The FP32 engine contains no INT8 layers. In the INT8 engine, 285 of 383 layers involve INT8 tensors, including all 74 convolution, 3 deconvolution, and 15 matrix-multiplication layers.

The CPU is an Intel Core i9-14900K in an HP Z2 Tower G9 workstation running Windows 11, with ONNX Runtime 1.24.1 and the CPU execution provider. Each session enables all graph optimizations and sequential execution with four intra-op threads and one inter-op thread. The process is pinned to four performance cores, using one logical processor per physical core and no efficiency cores. Among 1, 2, and 4 threads, four threads gave the lowest geometric mean of normalized FP32 and INT8 median latency. Latency and energy are measured on the models optimized and serialized by ONNX Runtime on the target machine. Profiling confirms integer execution: in the INT8 model, 57 of the 74 convolutions run as `QLinearConv`, together with 12 `QLinearMatMul` and 3 `QGemm` operators, whereas the remaining 17 convolutions, the 3 transposed convolutions, and normalization layers remain in FP32.

B.2 Operator-Level Computational Consistency

We verify the accelerator execution path by comparing operator outputs with a software reference using identical quantized inputs, weights, and shift parameters. The test uses 4-bit input and weight quantization, $T = 8$ timesteps, and random seed 0. Six convolution configurations are evaluated with 400 randomly generated input groups each. For each group, we compute the element-wise Mean-Squared Error (MSE) between the two output tensors and require zero MSE for a pass. Table 4 summarizes the recorded results: all 2,400 groups pass, with zero MSE in every group. These results establish numerical agreement for the tested convolution configurations and inputs; they do not measure full-network deployed mIoU or establish end-to-end checkpoint equivalence.

Table 4: Operator outputs compared with the software reference under 4-bit quantization and eight timesteps. Stride and dilation are given as height-width pairs. Maximum MSE is taken over the test groups for each configuration.

Kernel	Stride	Dilation	Passed/tested	Maximum MSE
3×3	$(1, 1)$	$(1, 1)$	400/400	0.00
3×3	$(2, 2)$	$(1, 1)$	400/400	0.00
3×1	$(1, 1)$	$(1, 1)$	400/400	0.00
3×1	$(1, 1)$	$(2, 1)$	400/400	0.00
1×3	$(1, 1)$	$(1, 1)$	400/400	0.00
1×3	$(1, 1)$	$(1, 2)$	400/400	0.00
Total	-	-	2400/2400	0.00

B.3 Energy-Measurement and Analytical-Estimation Protocol

GPU and CPU deployments use batch size 1. The measurement boundary covers model execution only: preprocessing, engine or session construction, and postprocessing are excluded, and GPU inputs are preloaded into device memory so that host-device transfers are also excluded. Gross power is the total power reported by the platform interface during an active window; idle power is measured over idle intervals in which the model and runtime are loaded but no inference is issued. We report

$$P_{\text{net}} = \bar{P}_{\text{gross}} - \bar{P}_{\text{idle}} , \qquad E_{\text{sys}} = \frac{1}{N} \int_{\mathcal{I}} \left(P(t) - \bar{P}_{\text{idle}} \right) \, dt , \tag{12}$$

where $\mathcal{I}$ contains N inferences. Table 5 lists the platform-specific settings, and Table 6 reports idle power together with gross and net energy per inference.

For the SNN accelerator, energy is measured per command block and extrapolated to the full network; Appendix C.2 gives the procedure.

For comparison with analytical estimates based on operation counts, we adopt the 45-nm, 0.9V per-operation energy reported by Horowitz (2014):

$$\begin{aligned} E_{\text{ANN}}^{\text{ana}} &= \sum_{l} N_l^{\text{MAC}} \left(e_{\text{mul}}(b_l) + e_{\text{add}}(b_l) \right) , \\ E_{\text{SNN}}^{\text{ana}} &= \sum_{l} T r_l N_l^{\text{MAC}} e_{\text{AC}}(b_l) . \end{aligned} \tag{13}$$

Table 5: Energy-measurement settings for the GPU and CPU deployments.

Setting	GPU (TensorRT)	CPU (ONNX Runtime)
Power interface	NVML cumulative board-energy counter	HWiNFO64 CPU package power with trapezoidal integration
Sampling interval	100 ms	1 s
Warm-up	200 inferences	30 s per model
Active window	50,000 inferences over 5,000 validation patches cycled 10 times	180 s over a cycled pool of 100 images
Repetitions	5 alternating FP32/INT8 pairs	5 alternating FP32/INT8 windows
Idle reference	10 s before and after each active window	180 s idle windows, duration-weighted

Table 6: Idle power and gross and net energy per inference for the GPU and CPU deployments. GPU values are medians over the five trials.

Platform	Precision	Idle power (W)	Gross energy	Net energy
Blackwell RTX PRO 6000	FP32	86.88	246.23 mJ	175.58 mJ
Blackwell RTX PRO 6000	INT8	86.41	159.66 mJ	91.06 mJ
VNNI i9-14900K	FP32	27.68	82.74 J	61.24 J
VNNI i9-14900K	INT8	27.68	6.95 J	5.24 J

Here N_l^{MAC} is the dense MAC count, r_l is the firing rate, and $e_{(\cdot)}(b)$ is the per-operation energy at bit-width b. Table 7 lists the 45-nm, 0.9-V arithmetic energy constants and the INT4 extrapolation used in this work. Both estimates cover arithmetic only, excluding memory, state update, and data-movement overhead.

Table 7: Arithmetic energy costs at 45 nm and 0.9 V, in pJ per operation. FP32, FP16, INT32, and INT8 multiplication and addition costs are from Figure 1.1.9 of Horowitz (2014); INT4 costs are estimates used in this work. MAC energy is computed as the sum of one multiplication and one addition.

Format	Multiply	Add/AC	Total/MAC
FP32	3.70	0.90	4.60
FP16	1.10	0.40	1.50
INT32	3.10	0.10	3.20
INT8	0.20	0.03	0.23
INT4 (estimated)	0.05	0.03	0.08

For INT4, we scale the INT8 multiplication cost quadratically with operand bit width: $e_{\mathrm{mul}}(4) = 0.20(4/8)^2 = 0.05$ pJ. We retain the INT8 addition cost, $e_{\mathrm{AC}}(4) = 0.03$ pJ, giving $e_{\mathrm{MAC}}(4) = 0.08$ pJ.

C SNN Accelerator

C.1 Physical Implementation and Layer-wise Workload Mapping

The accelerator is implemented in a 28-nm process and organized into eight tiles, each containing eight compute cores. Figure 4 shows the physical layout of one accelerator tile.

Figure 5 shows how the host executes the W4A1T8 workload of Appendix B.1 on the accelerator. The host invokes the accelerator one layer at a time. While a layer executes, it occupies all compute resources of the accelerator, and the host schedules the next layer only after all results of the current layer have been returned. Each hardware execution interval therefore corresponds to exactly one layer, which allows runtime and energy to be attributed to that layer.

For layer l, the host loads the input activations X_l, the weights W_l, and a command descriptor specifying the operator type, tensor dimensions, computation mode, and number of cores to invoke. It packs the data with the descriptor and splits the package into N_l mutually independent command blocks, each carrying a portion of the data and the corresponding execution-control fields. On the

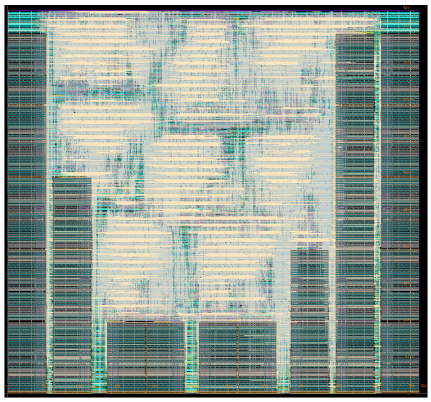

Figure 4: Physical layout of one tile in the jointly developed 28-nm SNN accelerator.

accelerator, a scheduler allocates the blocks across the eight tiles, each containing eight compute cores, according to these fields. For each block, the assigned cores read the activations and weights, perform the computation, and write the results. After all N_l blocks have completed, the accelerator returns the output Y_l to the host as the input to the next layer. The forward pass is thus mapped to an ordered sequence of hardware invocations, with parallel execution within each layer.

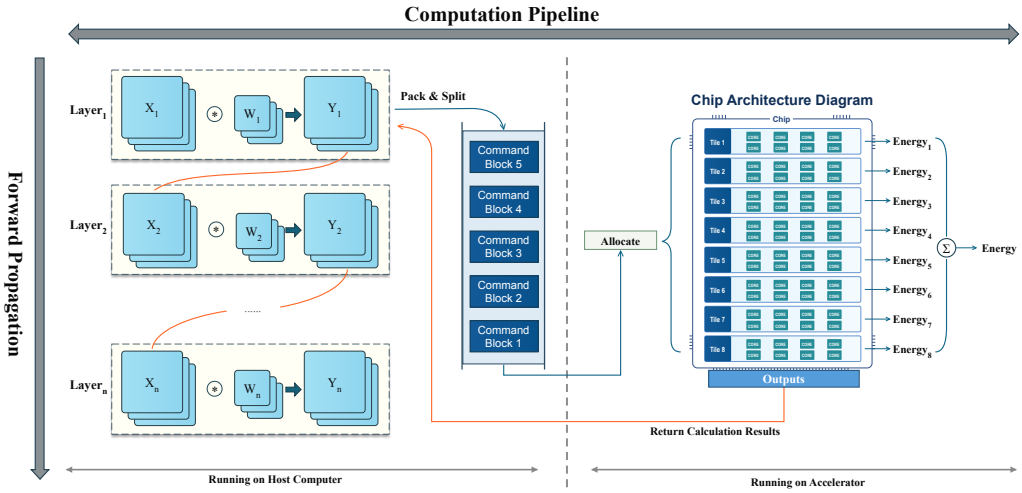

Figure 5: Layer-wise execution and energy measurement on the SNN accelerator. Host side (left): for each layer, the input activations X_l, the weights W_l, and the command descriptor are packed and split into command blocks. Accelerator side (right): the blocks are allocated across eight tiles of eight compute cores each, and the layer output is returned to the host as the input to the next layer. Per-tile energies are summed to give the block energy in Eq. (14).

C.2 Energy-Measurement Protocol

For command block n in layer l, let $\Delta t_{l,n}$ be its runtime and $\bar{P}_{i,l,n}$ the average power of tile i during its execution. The block, layer, and network energies are

$$E_{l,n} = \Delta t_{l,n} \sum_{i=1}^{8} \bar{P}_{i,l,n} \,, \qquad E_l = \sum_{n=1}^{N_l} E_{l,n} \,, \qquad E_{\text{acc}} = \sum_{l=1}^{L} E_l \,. \tag{14}$$

In practice, tile power is measured for 10 randomly sampled command blocks per layer, and their mean block energy is extrapolated to the layer total.