Quant-Trim in Practice: Improved Cross-Platform Low-Bit Deployment on Edge NPUs

Rayen Dhahri **Steffen Urban**
Corporate Research and Technology, Carl Zeiss AG
{rayen.dhahri, steffen.urban}@zeiss.com

Abstract

Specialized edge accelerators rely on low-bit quantization, but vendor compilers differ in scaling, clipping, and kernel support, often as black boxes. The same floating-point (FP) checkpoint can therefore yield inconsistent accuracy across backends, forcing practitioners to tweak flags or refactor models to vendor-friendly operator subsets. We introduce *Quant-Trim*, a training-phase method that produces a hardware-neutral checkpoint robust to backend and precision choices. It combines *progressive fake quantization* to align training with the deployed integer grid and *reverse pruning* to tame outlier-driven scale inflation while preserving learnability. Quant-Trim is agnostic to quantization schemes (symmetric/asymmetric, per-tensor/per-channel, INT8/INT4) and requires no vendor-specific graph changes. Across models and tasks, it narrows the FP→low-bit gap, reduces dependence on compiler heuristics/calibration, and avoids per-backend retraining. We report accuracy and edge metrics latency, throughput, energy/inference, and cost under static/dynamic activation scaling and varying operator coverage.

1 Introduction

Deploying neural networks in production typically imposes high demands on latency, power consumption, and cost, especially on the edge. Quantization is a primary lever: mapping floating-point parameters and activations to low-bit integers reduces compute and bandwidth without changing architecture [11, 20, 16]. In practice, however, low-bit accuracy is dominated by *activations*: their wide dynamic range, nonstationary statistics, and outliers make them fragile under clipping and coarse scaling [3, 8, 37, 26, 5].

A common compromise is to keep activations in FP16/BF16 while quantizing weights, which improves stability but retains higher memory traffic and power consumption [29, 17]. By contrast, many NPUs/ASICs enforce INT8 for both weights and activations with *offline* calibration [16, 36, 35, 34]. This heterogeneity means that the same FP checkpoint can produce *widely varying* accuracy across backends due to opaque scaling/fusion choices; and the need to redesign architectures for "quantization-friendliness," distill into restricted operator sets, or retrain per target [11, 8, 3, 30]. Methods that rescale activations into weights [37] or isolate outliers in higher precision [5] help, but typically assume backend support that is not universal on NPUs and remain restricted under vendor-specific quantizers aligning with their compilers.

We introduce *Quant-Trim*, a training-time procedure that produces a single, hardware-agnostic checkpoint robust to vendor compilers and precision regimes. Quant-Trim couples (i) *progressive fake quantization*, which interpolates between FP32 and low-bit scales to align forward statistics smoothly with the deployment grid, and (ii) *reverse pruning*, which pins extreme weights at the quantization boundary to prevent scale inflation while preserving learnability. By aligning train-time

EurIPS 2025 Workshop: Rethinking AI: Efficiency, Frugality, and Sustainability.

numerics with deployment, our recipe reduces sensitivity to backend scaling/clipping, enabling reliable INT8 export without per-backend retraining.

Contributions.

- **Robust Training with Fake Quantization.** We introduce a progressive fake-quantization curriculum that minimizes activation-induced errors across various deployment precisions, preventing optimization collapse.
- **Scale Management through Reverse Pruning.** Our pin-at-boundary method effectively mitigates weight outlier influence while preserving gradient flow and model capacity.
- **Edge Efficiency Evaluation.** We benchmark multiple edge devices (GPUs/SoCs/NPUs) and quantify latency, power, and energy-per-inference, highlighting their promise for efficient and greener AI deployment.

2 Background

Efficient deployment at the edge relies on numeric precision. Floating-point formats (FP32, FP16, BF16) trade accuracy for bandwidth and energy, enabling scalable training and inference [29]. Integer quantization (e.g., INT8/INT4) goes further by mapping tensors to fixed grids; with scale s and zero-point z,

$$Q(x) = \mathrm{clip}\Big(\lfloor x/s \rceil + z,\ q_{\min}, q_{\max}\Big), \qquad \hat{x} = s\,(Q(x) - z),$$

typically using symmetric (z=0) weights and asymmetric activations [16, 20, 10]. Per-channel scales often improve accuracy for conv/linear layers, but support varies by compiler.

Different hardware vendors ship black-box compilation and optimization algorithms tailored for their hardware and hence come with varying constraints, e.g.:

- Per-channel / asymmetric kernels
- Mixed precision (tunable per layer)
- Operator support
- Post-training layer fusion
- Histogram observers

The hardware used in our evaluations is covered under section A.1 and section A.2. Many NPUs require static INT8 for both weights and activations; GPUs allow mixed regimes (e.g., W8/A16) and emerging low-FP formats (FP8) [18]. These differences mean the *same* FP checkpoint can yield divergent low-bit accuracy across backends.

We seek a training-time procedure that produces a *single* checkpoint whose low-bit behavior is stable under heterogeneous compiler choices (scaling, clipping, kernel availability), without backend-specific graph edits. We base our approach on fake quantization (to align train-time forwards with deployed integer numerics) and scale control via reverse pruning by compressing the range.

3 Methodology

In this section we introduce our **Quant-Trim** approach. It combines two key components. First is the *progressive fake quantization* that smoothly interpolates between FP32 and low-bit execution to avoid optimization collapse, and the *reverse pruning* step that pins extreme weights at the quantization boundary to prevent a few values from inflating the scale while retaining representational power. The Quant-Trim training workflow is depicted in fig. 1. In the following subsection we describe each step in detail. Methods and earlier work that motivated our work are described in section 4.

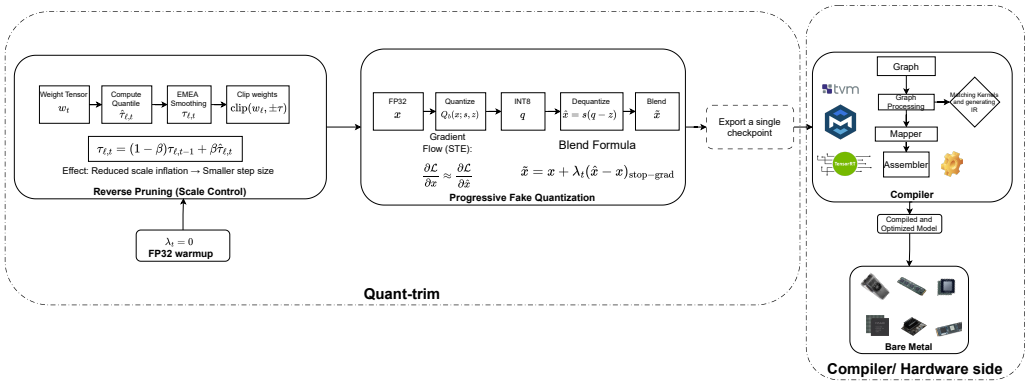


Figure 1: **Quant-Trim training pipeline.** Our method combines two key components: (1) **Reverse Pruning** clips extreme weights at robust quantile thresholds $\tau_{\ell,t}$ to prevent scale inflation while retaining representational power, and (2) **Progressive Fake Quantization** smoothly interpolates between FP32 and INT8 execution via a curriculum schedule λ_t to avoid optimization collapse. The blend coefficient gradually increases from 0 (full FP32 warmup) through a quartic ramp to 1 (full fake quantization), while computing per-tensor/channel scales and zero-points. Gradients flow via straight-through estimator (STE). The final model exports to standard ONNX without custom operators, ensuring compatibility with NPU compilers. The resulting model is then compiled to the target AI accelerator.

3.1 Problem definition and notation

3.1.1 Uniform quantizer and STE.

For a tensor x and bit-width $b = 8$, we use a uniform fake quantizer with scale $s > 0$ and zero-point z:

$$Q_b(x; s, z) \;=\; \mathrm{clip}\Big(\big\lfloor \tfrac{x}{s} + z \big\rceil,\; q_{\min},\; q_{\max}\Big), \qquad \hat{x} \;=\; s\big(Q_b(x; s, z) - z\big),$$

where $(q_{\min}, q_{\max}) = (-128, 127)$ for symmetric INT8 (weights) and $(0, 255)$ for asymmetric UINT8 (activations). We use the straight-through estimator (STE) [1, 14, 39]:

$$\frac{\partial \mathcal{L}}{\partial x} \;\approx\; \frac{\partial \mathcal{L}}{\partial \hat{x}},$$

and perform *progressive blending* at each *quantization point* (i.e., every weight tensor and designated activation location) with a strength $\lambda_t \in [0, 1]$:

$$\tilde{x} \;=\; x \;+\; \lambda_t \,(\hat{x} - x)_{\text{stop-grad}},$$

We use a single global blend coefficient λ_t shared across all quantization points at epoch t. Thus, $\lambda_t = 0$ equals full FP32 precision and $\lambda_t = 1$ is full fake-quant in forward. The gradients always follow FP32.

3.1.2 Robust statistics and tensor quantiles

Quantiles For a random variable X with CDF F_X, the p-quantile is

$$Q_X(p) \;:=\; F_X^{-1}(p) \;=\; \inf\{x :\, F_X(x) \geq p\}, \quad p \in (0, 1).$$

Given samples $x_{1:n}$ with order statistics $x_{(1)} \leq \cdots \leq x_{(n)}$, the empirical quantile is $\widehat{Q}_X(p) = x_{(\lceil pn \rceil)}$. For large tensors we compute $\widehat{Q}$ on a random subsample S_t, $|S_t| \leq S_{\max}$ (we use $S_{\max} = 10^5$). We write $p_{\text{hi}}, p_{\text{lo}} \in (0, 1)$ for upper/lower quantiles (e.g., $p_{\text{hi}} = 0.999$, $p_{\text{lo}} = 0.001$), $\mu \in (0, 1)$ for EMA momentum (we use $\mu = 10^{-3}$), and $\varepsilon > 0$ (we use 10^{-6}).

Per-tensor statistics Let $b = 8$. For *weights* (symmetric), with $X = |w|$,

$$m_t \;=\; Q_{|w|}(p_{\text{hi}}) \;\; (\text{or } \widehat{Q}^{(S)}), \qquad \tilde{m}_t \;=\; (1 - \mu)\, \tilde{m}_{t-1} + \mu\, m_t,$$

$$s_t^{(\mathrm{w})} \;=\; \frac{\max(\tilde{m}_t, \varepsilon)}{2^{b-1}-1}, \qquad z^{(\mathrm{w})} = 0.$$

For *activations* (asymmetric),

$$a_t \;=\; Q_x(p_{\mathrm{lo}}), \quad b_t \;=\; Q_x(p_{\mathrm{hi}}), \qquad \tilde{a}_t \;=\; (1-\mu)\,\tilde{a}_{t-1} + \mu\, a_t, \quad \tilde{b}_t \;=\; (1-\mu)\,\tilde{b}_{t-1} + \mu\, b_t,$$

$$s_t^{(\mathrm{a})} \;=\; \frac{\max(\tilde{b}_t - \tilde{a}_t, \varepsilon)}{2^b - 1}, \qquad z_t^{(\mathrm{a})} \;=\; \mathrm{clip}\Big(-\frac{\tilde{a}_t}{s_t^{(\mathrm{a})}},\, q_{\min},\, q_{\max}\Big).$$

The same definitions apply per-output channel c by replacing w with w_c (or x with x_c) and computing $Q_{|w_c|}$ (or Q_{x_c}) along the channel axes; EMAs are then channel-wise.

3.2 Reverse Pruning (Scale Control)

Outlier weights inflate the effective scale. For layer ℓ with weights w_ℓ, let

$$\hat{\tau}_{\ell,t} \;=\; \widehat{Q}^{(S)}_{|w_\ell|}(p_{\mathrm{clip}}) \quad (\text{e.g., } p_{\mathrm{clip}} = 0.95), \qquad \tau_{\ell,t} \;=\; (1-\beta)\,\tau_{\ell,t-1} + \beta\,\hat{\tau}_{\ell,t},$$

with $\beta \in (0, 1]$ an EMA momentum. Every K epochs after warmup we *pin*

$$w_\ell \leftarrow \mathrm{clip}\,\big(w_\ell,\, -\tau_{\ell,t},\, \tau_{\ell,t}\big).$$

For symmetric quantization the post-pruning step size satisfies

$$\Delta' \;=\; \frac{\tau_{\ell,t}}{2^{b-1}-1} \;<\; \Delta \;=\; \frac{\max_i |w_{\ell,i}|}{2^{b-1}-1},$$

allocating more representational levels to the bulk. Empirically, fig. 2 shows compressed weight tails and narrower downstream activation ranges.

3.3 Training Curriculum

We schedule the *global* blend λ_t over epochs t and apply it at every quantization point:

$$\lambda_t \;=\; \begin{cases} 0, & t < E_w \quad \text{(FP32 warmup);} \\ \min\Big(0.5,\; \big(\frac{t-E_w}{E_f-E_w}\big)^4 \cdot 0.5\Big), & E_w \le t < E_f \quad \text{(gentle quartic ramp);} \\ 0.5 + \Big(\min\big(1, \frac{t-E_f}{H}\big)\Big)^2 \cdot 0.5, & t \ge E_f \quad \text{(quadratic to full).} \end{cases}$$

Here E_w is warmup end, E_f is the end of the ramp, and H is the horizon to reach $\lambda_t = 1$. At each quantization point we compute $\hat{x}$ using $Q_b(\cdot)$ with current (s_t, z_t) from the robust statistics above and output $\tilde{x} = x + \lambda_t(\hat{x} - x)_{\text{stop-grad}}$; gradients use STE.

3.4 Training Procedure and Export

Putting it together:

1. **FP32 warmup** ($0{:}E_w$): train with $\lambda_t = 0$.
2. **Reverse pruning onset** ($t = E_w$): start EMA thresholds $\tau_{\ell,t}$; pin every K epochs.
3. **Progressive fake-quant** ($E_w{:}E_f$): enable fake-quant for weights (symmetric) and activations (asymmetric); update (s_t, z_t) via robust EMA quantiles; blend with quartic λ_t.
4. **Advanced/final** ($t \ge E_f$): quadratic ramp to $\lambda_t = 1$; keep STE for stability.

We realize this via per-layer wrappers that (i) temporarily substitute quantized weights in forward while keeping FP32 master weights for updates, and (ii) insert activation fake-quant hooks after common nonlinearities. The final checkpoint is exported to ONNX and compiled with vendor toolchains (TensorRT/TVM/NPU compilers); the computational graph remains standard (no fused rescaling or non-standard formats).

3.5 Mechanism and Intuition

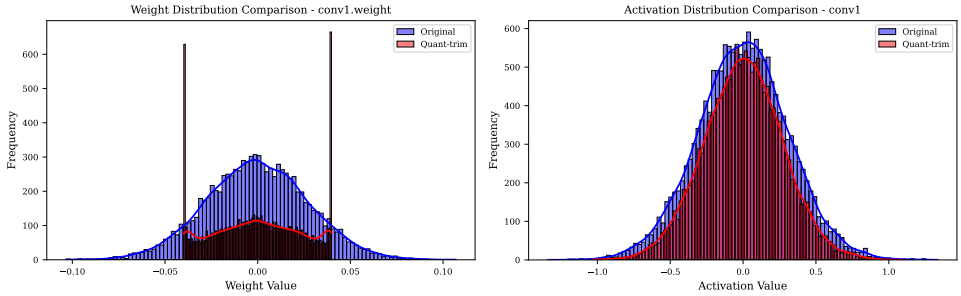


Figure 2: **Distributional effect of Quant-Trim.** Left: reverse pruning compresses weight tails, reducing scale inflation. Right: activations exhibit a narrower dynamic range, making INT8 mapping more stable.

Scale control. Pinning the largest $p\%$ weights reduces the extreme order statistics that set the quantization scale, shrinking Δ and allocating more representational levels to the bulk.

Smooth noise injection. The blend $\tilde{x} = x + \lambda_t(\hat{x} - x)_{\text{stop-grad}}$ gradually aligns train-time forward with deploy-time integer forward while the backward pass remains FP32-stable [1, 8], reducing the distributional gap between FP32 and low-bit, as visualized in fig. 2.

4 Related Work

QAT and learned quantizers. Quantization-aware training enhances low-bit deployment by learning clip ranges and step sizes (PACT, LSQ) [3, 8]. Extensions target transformers and mixed regimes, but accuracy can vary across compilers due to scale/zero-point handling and fusion rules [10].

PTQ and calibration sensitivity. Post-training methods reduce retraining cost but remain sensitive to activation range estimation and small calibration sets (AdaRound, BRECQ) [30, 24]. Weight-only routes help for LMs yet leave activation error unresolved (LLM.int8, AWQ, GPTQ) [5, 26, 9].

Activation outliers and distribution reshaping. A main failure mode is activation heavy tails under A8/W8 and A4/W4. Smoothing or reassigning scale to weights reduces clipping/rounding (SmoothQuant and variants) [37]. Rotation/dual-transform and low-rank absorption further reduce tails for ultra-low bits [25, 28, 23]. Many rely on fused rescaling or side branches that are not universally supported on edge NPUs.

Pruning for scale control. Classical magnitude pruning targets sparsity, not scale robustness. Outlier-aware pruning prioritizes sensitive channels but is backend-specific [22, 38]. Our *reverse pruning* focuses on *tail pinning*: clipping scale-setting weights to contract per-(tensor/channel) ranges before fake-quant—keeping the exported graph standard.

Backend heterogeneity. Edge compilers differ in per-channel/asymmetric support, placement of rescale ops, dynamic vs. static activation scaling, and allowed mixed-precision islands [16, 10]. This leads to cross-backend variance from the same FP checkpoint and motivates training-time robustness rather than backend-specific graph edits.

Regularization links. Bayesian/Laplace shrinkage can suppress heavy tails and improve sparsifiability; it is complementary to our scale-control view [15, 4, 6].

5 Experiments

We present large-scale optimization and experiments across multiple hardware platforms, tasks, and models to quantify NPU efficiency and the impact of varying numerical precision on devices that

support multiple precisions. The experimental pipeline is detailed in section A. We also provide comprehensive hardware descriptions and specifications to contextualize the results under section A.2 and to make explicit the limitations of each platform and conduct an ablation study on the effect of Quant-Trim components in section B. Device capabilities are summarized in table 6.We then show the stability of the training results and the effect of our method on the end quantization results on the accelerators.

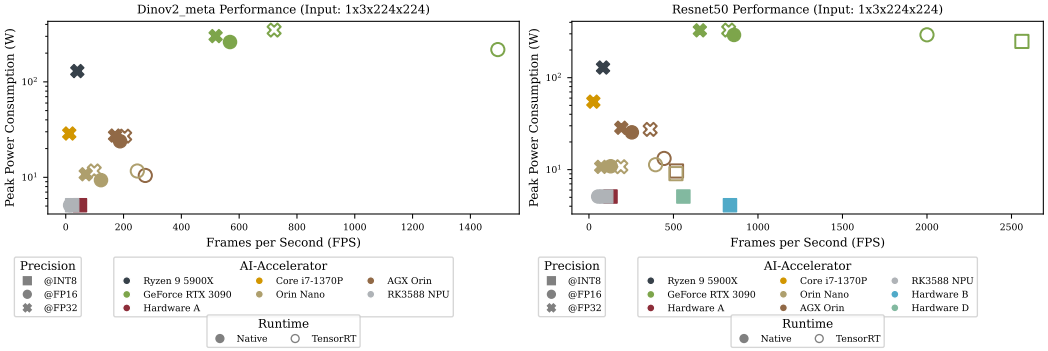
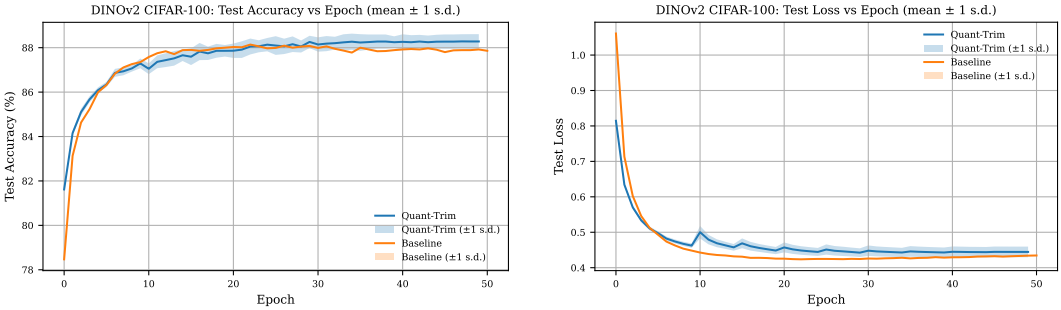


Figure 3: **Power–throughput trade-off for DINOv2 and ResNet-50.** Batch=1, 224×224 input. x-axis: median FPS over 200 timed iters after 20 warm-ups; y-axis: average Peak-power (5 runs; whiskers show 5–95th percentiles). **Encoding:** color = device; marker shape = precision; *filled* markers = platform's default runtime (NPUs: vendor runtime; NVIDIA: CUDA), *unfilled* markers = TensorRT. Left: DINOv2; Right: ResNet-50. Device specs in table 6.



(a) Top-1 accuracy (Quant-Trim vs. MAP).

(b) Test loss (Quant-Trim vs. MAP).

Figure 4: **Training dynamics on CIFAR-100 (DINOv2).** A small accuracy dip during the ramp phase is followed by convergence to MAP-level performance.

5.1 Inference Results and Efficiency

Within the same hardware, lower precision yields faster speeds. For NVIDIA GPUs and embedded Jetson devices, TensorRT gives a significant boost compared to using CUDA kernels. In general, for the shown models, the requirements to perform real-time tasks are slightly beyond 60 FPS to be able to build on top and take into consideration the latency that comes from data processing and sensor systems. We provide technical details about the system and the potential latency added for end deployment under section A.1.

Power Consumption vs Precision NPUs' energy consumption is very low in comparison to NVIDIA GPUs, which can pull up to 200 W, while NPUs do not exceed 10 W. Hardware that supports different precision like the Jetson family and GPU are 2 to 3 times faster at lower precision compared to FP32 for both ResNet-50 and DINOv2. The Power Consumption is measured on chip.

Runtime Provider Within the same hardware, we can see that for NVIDIA hardware (GPU and Jetson), TensorRT provides a large speedup for FP16, nearly tripling the speed of DINOv2 from 600

FPS to over 1500 FPS as shown in fig. 3.Additional results for U-Net [33] and MobileNetV3-Small [13] are shown in fig. 11.

5.2 Training dynamics

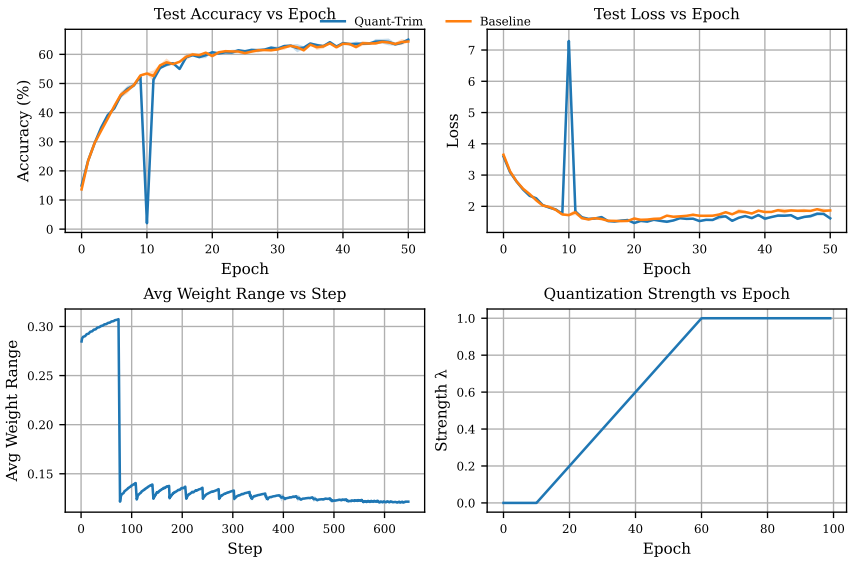


Figure 5: Quant-Trim exhibits a brief dip when fake-quantization ramps in, then recovers to near-baseline accuracy and loss by the end of training for ResNet on CIFAR-100.

Training dynamics & convergence. Similar to MAP training, Quant-Trim leads to full convergence at the end of training and similar predictive performance. Training stabilizes, ensuring progressive quantization with a slight drop once we start the fake-quantization process, but regains near complete accuracy by the end of training for ResNet-50 and DINOv2 as shown respectively in fig. 5 and fig. 4. A similar trend is seen for ResNet-18 on Coco as shown in fig. 10.

Figure 6: **NanoSAM2 distillation with Quant-Trim.** For two inputs (left in each subfigure), the student's deepest FPN feature matches the teacher's saliency structure while exhibiting fewer saturated patches after reverse pruning.

Feature alignment (NanoSAM2). We distill a compact NanoSAM2 image encoder from *SAM-2.1 Hiera* inspired by the `nanosam` recipe[1], adapted to SAM-2.1 and our Quant-Trim curriculum (section 3). Distillation minimizes a three-scale FPN loss (Huber; weights $[1, \frac{1}{4}, \frac{1}{8}]$) between teacher and student features, while Quant-Trim runs on the student to align training numerics with INT deployment. NanoSam2 reaches an **mIOU** of **0.5889** on the Coco 20217 validation set [27]. Fig. 6 shows representative cases: the student reproduces the teacher's saliency structure (object contours, part boundaries) without high-frequency artifacts, and reverse pruning suppresses rare saturated responses that otherwise inflate activation ranges. Qualitatively, the feature maps remain sharp at object edges and smooth in background regions, indicating that Quant-Trim preserves the inductive content required for mask decoding while preparing the checkpoint for NPU compilers. As shown in

[1] `https://github.com/NVIDIA-AI-IOT/nanosam`

Method	Top-1	Top-5	MSE↓	Brier↓	ECE↓
Quant-Trim	65.10 (64.60)	86.90 (86.80)	0.01601	0.54295 (0.55070)	0.19791 (0.19966)
MAP	62.70 (62.30)	85.70 (85.80)	0.02110	0.56826 (0.57182)	0.22860 (0.23226)

Table 2: ResNet-50 on CIFAR-100, Hardware D. Entries are **On-Device**; ONNX (FP32) references are in parentheses. MSE is computed as the mean squared difference between device and ONNX *logits* (pre-softmax). **Average FPS**: 571, **IP execution time**: 1.5 ms

fig. 7, NanoSAM2–ResNet-18 reaches real-time latencies on specialized NPUs at single-digit watts, surpassing a desktop FP16 GPU baseline in our setting. *Implementation note:* the image encoder (dominant compute) runs on the NPU, while the lightweight prompt decoder runs on the host CPU.

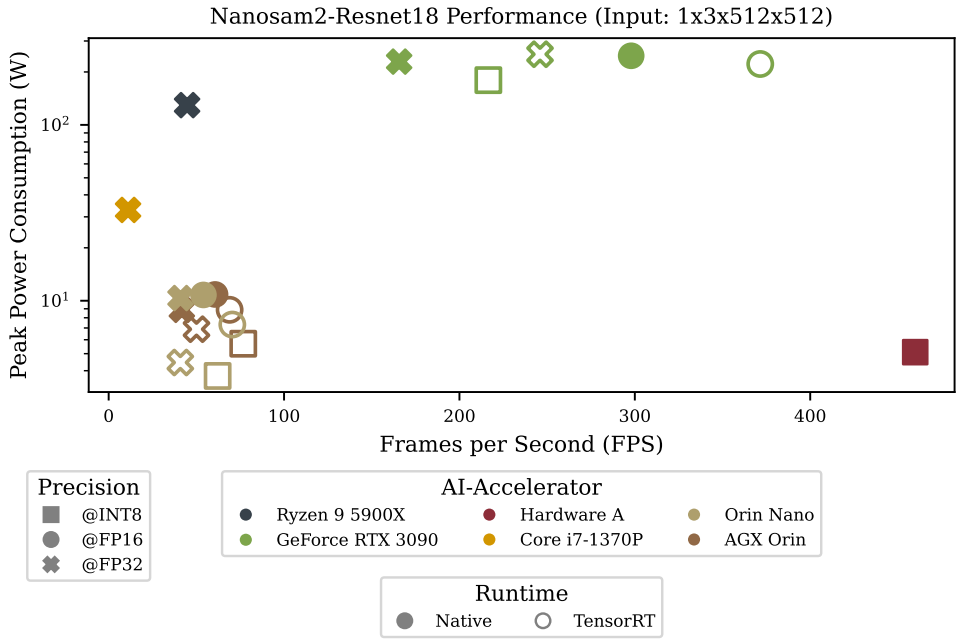


Figure 7: **End-to-end inference for NanoSAM2–ResNet-18 across accelerators.** Single-image, batch=1, 512×512 input; tiled inference with 50% overlap when required. Results are averaged across runs and warmup runs are used for each hardware. Runtimes use TensorRT (FP16) on GPU/Jetson and vendor runtimes on NPUs (INT8 or BF16/INT as supported). *Hardware A*, which uses A8W8, is 6 times faster than the Jetson family hardware and faster than the desktop GPU at an FP16 baseline while operating at single-digit watts (about 5 W).

5.3 On-Device Deployment

Method	Top-1	Top-5	MSE↓	Brier↓	ECE↓
Quant-Trim	64.60 (64.60)	86.80 (86.80)	0.01344	0.55235 (0.55070)	0.19821 (0.19966)
MAP	64.30 (62.30)	86.60 (85.80)	0.03987	0.57446 (0.57182)	0.22984 (0.23226)

Table 1: ResNet-50 on CIFAR-100, Hardware B(INT8 weights, BF16 activations). Over 500 frames **average FPS**: 834.18 and average System Latency: **5.34 ms**. Entries are **On-Device**; FP32 values are in parentheses. Quant-Trim improves Top-1 by +2.3 pts and cuts MSE by ∼66% vs. MAP, with near FP32–device parity across metrics.

Beyond accuracy (MSE and calibration). We quantify backend drift using the mean squared error (MSE) between *logits* (pre-softmax scores) produced on-device and by the ONNX FP32 reference: $\mathrm{MSE} = \frac{1}{N}\sum_i \|\text{device\_logits}_i - \text{onnx\_logits}_i\|_2^2$. On Hardware B, Quant-Trim cuts logit MSE from 0.03987 (MAP) to 0.01344 ($\downarrow\sim 66\%$), with improved calibration (Brier $0.57446 \to 0.55235$, $\downarrow\sim 3.8\%$; ECE $0.22984 \to 0.19821$, $\downarrow\sim 13.7\%$) and a small Top-1 gain ($+0.3$ pts), see table 1. On Hardware D, Quant-Trim similarly reduces logit MSE from 0.021 to 0.016 ($\downarrow\sim 24.2\%$) and improves calibration (Brier $0.56826 \to 0.54295$, $\downarrow\sim 4.5\%$; ECE $0.22860 \to 0.19791$, $\downarrow\sim 13.4\%$) while adding $+2.4$ Top-1 pts; see table 2. In both cases, lower MSE indicates closer numerical agreement to the FP32 reference at the *logit* level (before probabilities are formed), and lower Brier/ECE reflect better probability calibration. For ResNets used as a *feature backbone*, the lower MSE also signals higher backbone *signal fidelity*, which helps keep the encoder usable for downstream heads and distillation targets—especially relevant on the edge where ResNet backbones commonly distil or interface with Vision Transformer (ViT) backbones in architectures like DETR, SAM and CLIP [7, 2, 19, 32].

Signal-to-Noise Ratio (SNR) We show in table 3 that Quant-Trim demonstrates superior signal fidelity during deployment on Hardware A, achieving an SNR of 43.12 on the output layer, even with only calibration enabled. In contrast, models trained using MAP, augmented with hardware-based optimization techniques like Equalization and Adaround (which require additional overhead), achieve a lower SNR of 34.3. This highlights Quant-Trim's ability to maintain higher signal quality without the need for extensive post-training adjustments, ensuring robust deployment across hardware platforms

Training Method	**SNR (Output Layer)**	**Training Details**
Quant-Trim (Calibration Only)	43.12	no additional fine-tuning
Baseline (Equalization + Adaround [30])	34.30	320 epochs, 256 images, all layers quantized

Table 3: Comparison of Signal-to-Noise Ratio (SNR) on the output layer of ResNet-50 for Hardware A (A8W8 INT). Quant-Trim achieves significantly higher SNR with only calibration enabled compared to the baseline trained with Equalization + Adaround over 320 epochs.

6 Conclusion

We presented *Quant-Trim*, a training-time procedure that couples progressive fake quantization with reverse pruning to produce a single, hardware-agnostic checkpoint. Across CNNs and transformers, Quant-Trim narrows the FP→INT gap, stabilizes training despite activation outliers, and reduces sensitivity to backend scaling, clipping, and operator coverage. On edge accelerators, the same checkpoint compiles reliably and delivers favorable latency/energy trade-offs without per-backend retraining or concrete graph edits. We further demonstrated practicality with NanoSAM2, showing on-device throughput gains while retaining accuracy. Future work will extend the evaluation to larger datasets and models.

Limitations This work highlights a promising approach to optimizing neural networks for post-training quantization from the compiler's perspective. It is important to note that this is a work in progress. One challenge we are addressing separately is the deployability of these networks on specific hardware, as some architectures may not be suitable yet for deployment. The vendors and hardware that we tested are initially designed for particular architectures. Vendors are actively working to improve support for transformer architectures. Some vendors have either released custom hardware primarily suited for generative AI workloads, others are adjusting their compilers for enhanced compatibility, showing the promise of Edge Deployment. Our goal is not to compare one device to another but rather improve the deployment results for each hardware as selecting a specific NPU for an application highly depends on the task at hand, operation support and the size of the graph. We are committed to advancing efficient green AI solutions. While our approach is primarily designed to meet stringent real-world application requirements, it also scales up to GPU levels. With further tweaking and optimization, we can reduce the footprint of larger GPUs. Future evaluations on larger datasets will enhance this work, as we are currently addressing both training and inference; the experiment load is massive, thus we will include a broader evaluation in a future version.

References

[1] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons. *arXiv preprint arXiv:1305.2982*, 2013.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*, pages 213–229. Springer, 2020.

[3] J. Choi, Z. Wang, S. Venkataramani, P. Chuang, V. Srinivasan, and K. Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. In *International Conference on Learning Representations (ICLR) Workshop*, 2018.

[4] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux: Effortless bayesian deep learning. In *NeurIPS*, 2021.

[5] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[6] R. Dhahri, A. Immer, B. Charpentier, S. Günnemann, and V. Fortuin. Shaving weights with occam's razor: Bayesian sparsification for neural networks using the marginal likelihood. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[7] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

[8] S. K. Esser, J. L. McKinstry, V. Bablani, R. Appuswamy, and D. S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020.

[9] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[10] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2022.

[11] A. Gholami, S. Kim, Z. Dong, Z. D. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. In *JMLR*, 2017.

[15] A. Immer, M. Bauer, V. Fortuin, and G. Rätsch. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, 2021.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

[17] D. Kalamkar, D. Mudigere, J. Wang, H. Yuen, P. van Beek, S. Sridharan, M. Smelyanskiy, and D. Das. A study of bfloat16 for deep learning training. In *arXiv preprint arXiv:1905.12322*, 2019.

[18] Y. Kim, P. Micikevicius, D. Masters, B. Ginsburg, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2023.

[19] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything, 2023.

[20] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. In *arXiv preprint arXiv:1806.08342*, 2018.

[21] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[22] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park. OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 12, pages 13355–13364, 2024.

[23] M. Li, Y. Lin, Z. Zhang, T. Cai, X. Li, J. Guo, E. Xie, C. Meng, J.-Y. Zhu, and S. Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024.

[24] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction, 2021.

[25] H. Lin, H. Xu, Y. Wu, J. Cui, Y. Zhang, L. Mou, L. Song, Z. Sun, and Y. Wei. Duquant: Distributing outliers via dual transformation makes stronger quantized llms, 2024.

[26] S. Lin, E. Frantar, M. Kurtz, P. Stock, and D. Alistarh. Awq: Activation-aware weight quantization for llms. *arXiv preprint arXiv:2306.00978*, 2023.

[27] T.-Y. Lin et al. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[28] L. Liu, H. Wu, Y. Wang, H. Xu, G. Lin, Y. Wei, and Z. Sun. Rotated runtime smooth: Training-free activation smoothing for accurate int4 inference. *arXiv preprint arXiv:2409.20361*, 2024.

[29] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, S. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

[30] M. Nagel, M. Van Baalen, T. Blankevoort, and M. Welling. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, pages 7197–7206, 2020.

[31] M. Oquab et al. Dinov2: Learning robust visual features without supervision. *arXiv:2304.07193*, 2023.

[32] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[33] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[34] I. O. Team. Openvino post-training optimization tool, 2023. `https://docs.openvino.ai/latest/pot_introduction.html`.

[35] N. T. Team. Nvidia tensorrt: Int8 calibration, 2023. `https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#int8-calibration`.

[36] T. L. Team. Tensorflow lite: Post-training quantization, 2023. `https://www.tensorflow.org/lite/performance/post_training_quantization`.

[37] W. Xiao, E. Frantar, P. Stock, E. Elsen, M. Kurtz, and D. Alistarh. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.

[38] L. Yin, Y. Wu, Z. Zhang, C.-Y. Hsieh, Y. Wang, Y. Jia, M. Pechenizkiy, Y. Liang, M. Bendersky, Z. Wang, and S. Liu. OWL: Outlier weighed layerwise sparsity — a missing secret sauce for pruning llms to high sparsity. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 57101–57115, 2024.

[39] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin. Understanding straight-through estimator in training activation quantized neural nets, 2019.

A Experimental Setup

A.1 Hardware Platforms

The performance and accuracy of a quantized model are fundamentally tied to the capabilities of the target hardware, including its supported precisions, compiler toolchain, and supported operations. To evaluate the robustness of our method, we benchmark across two distinct classes of edge devices: versatile embedded GPUs and specialized AI accelerators (NPUs).

- **Embedded GPUs:** We use NVIDIA's Jetson Nano and Jetson AGX Orin. These platforms offer general-purpose CUDA and Tensor Core acceleration with broad support for FP32, FP16, and INT8. Their compilers (e.g., TensorRT) provide significant flexibility, including support for **dynamic quantization**, where activation scales are computed on-the-fly. This adapts to varying input data distributions but can introduce runtime overhead.
- **Specialized NPUs:** We benchmark on three representative edge AI accelerators anonymized as Hardware A, B, and C and provide their specifications. These ASICs are optimized for high-throughput, low-power inference, primarily supporting aggressive INT8 (even INT4) quantization, while it is possible to set some layers to INT16; that might risk not fitting into the memory. Their toolchains are often more restrictive and typically rely on **static quantization**, where activation ranges are pre-calibrated offline using a representative dataset. This minimizes runtime computation but makes model performance highly sensitive to the calibration data and potential distribution shifts during deployment.

Device	Type	W/A path	Act. scaling @ inference	PTQ calib. (INT)
Jetson Nano	SoC (GPU)	W8/A8 or W8/A16	STATIC (INT) or QAT	Cond.
Jetson AGX Orin	SoC (GPU)	W8/A8, W8/A16	STATIC (INT) or QAT	Cond.
Hardware A	M.2 NPU	W8/A8 (INT8)	STATIC (no runtime dyn)	Yes
Hardware D	M.2/PCIe NPU	W8/A8 (INT8) or BF16	STATIC (compiler-provided)	No
RK3588 (RKNN)	SoC (NPU)	W8/A8 or FP16	STATIC (INT8 only)	Cond.
Hardware B	M.2 NPU	W8/ABF16 (hybrid)	N/A (A=BF16)	No

Table 4: **Quantization behavior (corrected).** *W/A path* denotes weight/activation precisions at inference. *Act. scaling*: STATIC = fixed/static activation ranges used by the backend (may come from compiler defaults or embedded QAT scales); QAT = scales learned during QAT and embedded in the graph. *PTQ calib. (INT)* indicates whether a representative dataset is required *when an INT mode is selected*. For RK3588, calibration is Cond.: required for INT8, not for FP16 or BF16.

A.2 Hardware Form Factors and Practical Advantages

Edge accelerators appear in two dominant forms:

- **Add-in NPUs** (M.2 / PCIe, USB). Examples include Hardware A(M.2 B/M-key), Hardware B(M.2), and Hardware D(M.2/PCIe variants). These modules attach to a host (x86/ARM SBC, laptop, or embedded carrier) over PCIe (or occasionally USB). The host handles I/O, pre/post-processing, and scheduling; the NPU executes the neural network graph. Advantages: *drop-in* acceleration for existing systems, low incremental power (single-digit watts), and flexible host software. Caveats: host–device transfers (DMA), graph partitioning, and op coverage can introduce tail latency if unsupported ops fall back to the host.
- **System-on-Chip** (SoC) platforms. Examples include NVIDIA Jetson (Orin family) and Rockchip RK3588 boards. These integrate CPU, GPU/NPU, DRAM, and peripherals on one module; the full application stack (capture, pre/post, model, control) runs on-device. Advantages: single box, deterministic I/O, simpler memory topology, and easy CPU fallback for unsupported ops. Caveats: total power headroom is fixed; thermal throttling and shared memory bandwidth must be managed.

Form factor	Typical link	Strengths	Watch-outs
M.2 / PCIe NPU	PCIe Gen3/4 x2–x4	Drop-in accel; low watts; scalable	PCIe copies; op coverage; host fallbacks
USB NPU stick	USB 3.x	Quick prototyping; portable	Higher copy overhead; limited bandwidth
SoC (Jetson)	On-package MMU/DRAM	Unified memory; full stack on device	Thermal/power budgets; shared BW
SoC (RK3588)	On-package MMU/DRAM	Low cost; rich I/O	Compiler maturity; INT-centric kernels

Table 5: Form factors, data links, and common trade-offs at the edge.

Accelerator	Form factor / Interface	Peak perf.	Typical power
Hardware A[†]	M.2 2280 (B/M); PCIe Gen3 x2	26 TOPS (INT8)	∼2.5 W
Hardware B[‡]	Chip (PCIe Gen3 x4 / USB3); M.2 module	6 TOPS/chip (INT8)	0.5–2 W/chip
Hardware D[§]	Low-profile PCIe; Gen3 x8	60 TOPS (INT8)	8–10 W

Table 6: **Edge NPU summary (TOPS & typical power).** Vendor-quoted peak performance; INT8 unless noted. Symbols: [†] Hardware Auses on-chip SRAM only (no external DRAM); [‡] Hardware Bnumbers are per chip (M.2 module aggregates *4* chips, up to ∼24 TOPS); [§] Hardware Dalso reports ∼30 TFLOPS (BF16), typical 8–10 W.

Operator support and fallbacks. Add-in NPUs excel when the compiled graph maps *entirely* to accelerator kernels. If a layer is unsupported (or precision constraints force a dequant–requant island), the compiler will in some cases route that subgraph to the host CPU/GPU; it is also possible to crop and run only a subgraph on the NPU if the issue is only in the pre- or post-processing layers. This is functionally correct but may introduce large latency spikes due to extra memory traffic and synchronization. SoCs tolerate such cases better (same memory space, no PCIe), though overall throughput may be lower.

Pre/post-processing placement. Lightweight steps (resize, normalization, tiling) should run where they minimize copies: on SoCs, prefer GPU/NPU-side ops with zero-copy buffers; on M.2 NPUs, batch/pack on the host, transfer once, and fuse post-ops into the compiled graph where supported. In both cases, avoid alternating host–device–host hops in the critical path.

Memory and bandwidth. Edge NPUs rely on on-chip SRAM and tiling to maintain high reuse; DRAM is accessed in bursts. Sustained performance is determined as much by *dataflow* (blocking/tiling, operator fusion, quantized layout) as by peak TOPS. SoCs share DRAM across CPU/GPU/NPU, so contention and cache policy can affect tail latency; pinning cores and using contiguous DMA buffers often helps.

A.3 Appendix: Hyperparameters and Metrics

Reverse pruning: every K epochs, clip weights to the p_{clip} percentile (per-tensor or per-channel). Running ranges use EMA with momentum μ.

Measurement protocol. Inference: *20 warmup + 200 timed* iterations; medians over **5 runs**. Training: medians over **3** random seeds.

Table 8: Architecture-specific tweaks.

	ResNet (CNN)	DINOv2 (Transformer)
LR / warmup	higher LR; shorter E_w (10–30)	lower LR; longer E_w (30–50)
Ramp length E_f	30–50	60–100
p_{clip} / K	0.90–0.95 / 5	0.96–0.98 / 15
EMA μ	10^{-3}–10^{-2}	10^{-4}–10^{-3}
Attention handling	n/a	Q/K/V and outputs fake-quant; keep scores in FP
Final blend cap	$\alpha_{\max}$=1.0	$\alpha_{\max}$≈0.8

Table 7: Minimal QAT defaults by task/dataset.

	CIFAR-100	**Segm. (COCO)**
Epochs / Batch	100 / 128	100 / 32
Optimizer / LR	AdamW / 3×10^{-4}	AdamW / 5×10^{-4}
Weight decay	0.01	1×10^{-4}
LR schedule	Cosine	Cosine
E_w, E_f, H	10, 50, 20	15, 30, 20
p_{clip} / K	0.90 / 5	0.95 / 5
EMA μ	10^{-2}	10^{-3}
Target precision	INT8 (W/A)	INT8 (W/A)

Metrics (reported). **Classification**: Top-1/Top-5. **Segmentation**: mean Intersection over Union (mIoU), with $\text{mIoU} = \frac{1}{N}\sum_{i=1}^{N} \text{IoU}_i = \frac{1}{N}\sum_{i=1}^{N} \frac{|A_i \cap B_i|}{|A_i \cup B_i|}$ **Calibration/robustness**: ECE, MSE (logits). **Efficiency**: FPS, average power (W).

A.4 Models

To ensure robustness across architectures, we evaluate:

- **ResNet-50 and ResNet-18** [12]: canonical CNN backbone with residual connections.
- **NanoSAM2**: SAM variant with a ResNet-18 backbone; we use knowledge distillation to train the model on COCO.
- **DINOv2** [31]: self-supervised vision transformer trained on large-scale data, challenging to quantize.

A.5 Datasets

We evaluate on:

- **CIFAR-100** [21]: 100-class dataset of tiny natural images (32×32). Serves as a proxy for classification robustness under quantization.
- **MS-COCO** [27]: large-scale benchmark for detection and segmentation. Stress-tests activation quantization due to varied input scales and long-tail distribution.
- **CIFAR-10** [21]: Used for ablations due to its lightweight compute footprint, enabling rapid sweeps over clipping percentiles and fake-quant schedules.

B Ablation Study: Quantization Components and Clipping Sensitivity

We conduct a systematic ablation study on ResNet-18 with CIFAR-10 to isolate the contributions of different quantization components in our framework. The study evaluates five experimental configurations to understand the individual and combined effects of fake quantization training and reverse pruning (outlier weight removal). The experimental setup is described in table 9. The five configurations isolate the role of our fake-quantization and reverse pruning ; all share identical optimizer and schedule.

Our experiment reveals that on the training level, all ablation configurations converge to similar validation accuracy (≈81%) despite varying quantization strategies and clipping thresholds (90%, 95%, 99%) as seen in fig. 8. The convergence of both the FP32 baseline and pruning-only variant to comparable performance levels demonstrates that our progressive quantization strength schedule λ effectively maintains model capacity across diverse compression approaches. This robustness to hyperparameter choices validates the stability of the quantization-aware training framework and suggests that the specific clipping threshold has minimal impact on final model performance. In

Config	Fake-Quant (INT8)	Reverse-Pruning	p_{clip}	Epochs	Notes
(1) FP32 Baseline	✗	✗	–	50	Standard full-precision training
(2) QAT Only	✓	✗	–	50	INT8 fake-quantization; RP disabled
(3) Reverse Pruning Only	✗	✓	95%	50	FP32 training with percentile clipping
(4) QAT + 90% Clipping	✓	✓	90%	50	Aggressive outlier removal
(5) QAT + 99% Clipping	✓	✓	99%	50	Conservative outlier removal

Table 9: **Ablation configurations** for ResNet-18 on CIFAR-10. Shared hyperparameters across all runs: SGD optimizer, learning rate 10^{-3}, weight decay 5×10^{-4}, 50 epochs, and n=3 seeds. Only quantization settings differ between rows.

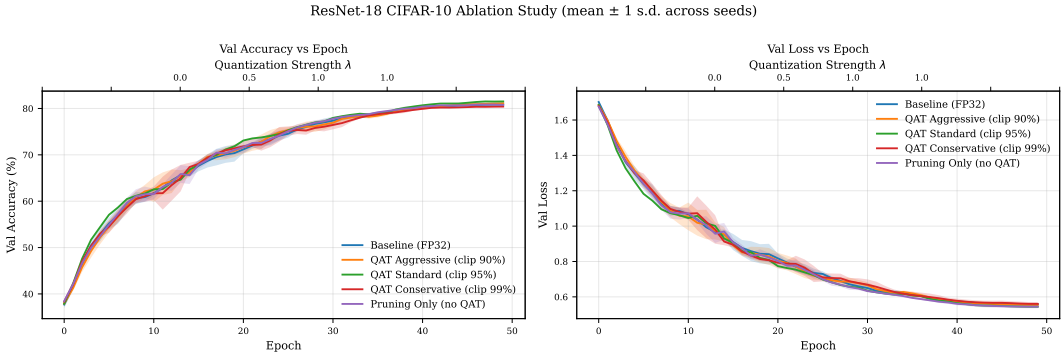


Figure 8: **Ablation study on ResNet-18 CIFAR-10 shows convergence to similar accuracy across configurations. (Left)** Validation accuracy versus training epoch. Despite varying quantization strategies (aggressive clip 90%, standard clip 95%, conservative clip 99%) and the baseline FP32 model, all configurations converge to approximately 81% validation accuracy by epoch 50, demonstrating that the quantization strength schedule λ (top axis) does not significantly impact final model performance. **(Right)** Validation loss follows a consistent downward trend across all methods, further confirming stable convergence behavior. The pruning-only baseline (no QAT) achieves comparable performance, suggesting that the learned representations are robust to different compression approaches. Shaded regions indicate ±1 standard deviation across 3 random seeds per configuration.

addition in fig. 9, We can see based on the activation distribution that for our QAT approach paired with reverse pruning at 95% results in a smoother activation distribution that is easier to quantize and with less outliers.

C Pseudo code

Algorithm 1 Quant-Trim Training

Initialize weights $w^{(0)}$ in FP32
for epoch $t = 1, \ldots, T$ **do**
  if $t = E_w$ **then**
    Reverse prune (all ℓ): $w_\ell \leftarrow \text{clip}(w_\ell, -\tau_{\ell,t}, \tau_{\ell,t})$ with $\tau_{\ell,t} = (1-\beta)\tau_{\ell,t-1} + \beta\, \widehat{Q}^{(S)}_{|w_\ell|}(p_{\text{clip}})$
  end if
  Update robust stats $(s_t^{(\text{w})}, z^{(\text{w})}{=}0)$ and $(s_t^{(\text{a})}, z_t^{(\text{a})})$ via EMA percentiles
  Set λ_t by the schedule above
  Fake-quant forward: $\hat{w} = Q_b(w; s_t^{(\text{w})}, 0)$, $\hat{x} = Q_b(x; s_t^{(\text{a})}, z_t^{(\text{a})})$, output $\tilde{x} = x + \lambda_t(\hat{x} - x)_{\text{stop-grad}}$
  Backprop with STE on FP32 master weights
end for
Export checkpoint → ONNX (compile with TensorRT/TVM/NPU compilers)

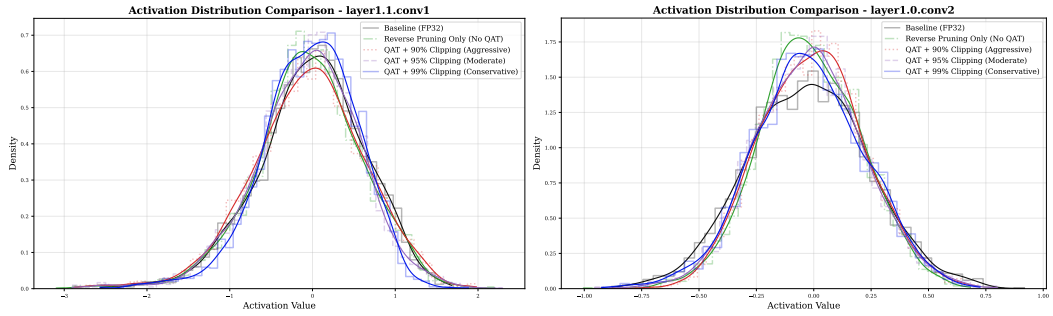


Figure 9: Weight distribution comparison across ablation study configurations for ResNet-18 on CIFAR-10. The baseline FP32 model (black) exhibits the widest weight distribution, while quantization-aware training introduces characteristic distribution narrowing. Reverse pruning alone (green dash-dot) provides regularization effects visible as distribution tightening around zero. The combination of QAT with different clipping percentiles reveals the trade-off between outlier removal and weight preservation: aggressive 90% clipping (red dotted) creates the most **constrained** distribution, while conservative 99% clipping (blue) maintains broader weight ranges and **95%** being the sweet spot (purple) where it is observed in a **low MSE of 0.00023** between compiled model and FP32 predictions on Hardware B.

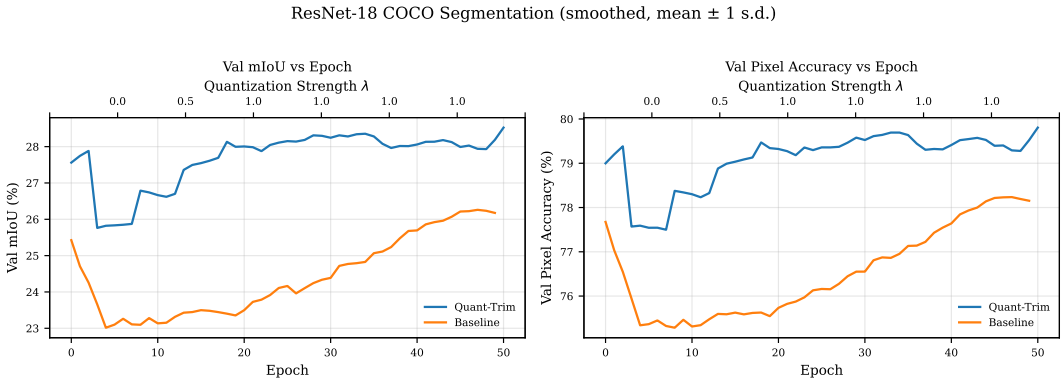


Figure 10: ResNet-18 COCO Segmentation: Val mIoU and Pixel Accuracy vs Epoch

D Additional Results

Hardware	Type	Price	Peak Power (W)	Runtime env (ACC)	Runtime (s)	Price per Watt (€)
RTX 3090 (comparison)	GPU	1500€	190	TensorRT (FP16)	0.12	0.127
Jetson Orin Nano 8 GB	SOM	250€	10	TensorRT (FP16)	0.66	0.040
Hardware A	M.2 Mod.	150€	5	(INT8)	0.10	0.033
Hardware B	M.2 Mod.	125€	5	(BF16)	0.60	0.040
Hardware C	Full SoC	250€	8	(INT/FP16)	3.50	0.032
Hardware D	M.2 Mod.	350€	8	(INT/FP16)	0.76	0.023

Table 10: NanoSAM2 backbone runtime for one 2k×2k image (50 tiles). We report backbone latency only; the lightweight decoder runs on CPU. Images larger than 1024 px are processed by tiled inference (512×512 tiles with 50% overlap).

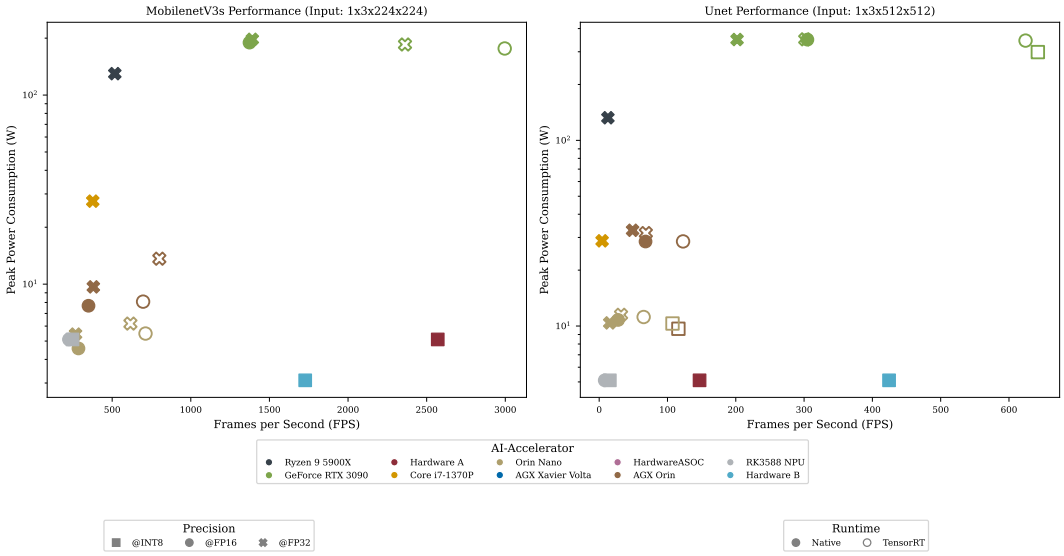


Figure 11: Performance and power consumption comparison across different AI accelerators for MobileNetV3s and U-Net models.