STOUT: STRUCTURED ORDER-AWARE QUANTIZATION TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Layerwise quantization-aware training (QAT) reduces the training cost of large language models by optimizing one layer at a time. However, a fixed input-to-output order assigns adaptation by depth, without considering which layers can best recover from errors introduced by other layers. It can therefore leave layers with limited adaptation ability to handle errors after more capable layers have been frozen. We introduce Oats, Order-aware Adaptation Score, which estimates a layer's potential recovery from other layers' quantization errors. Our local analysis explains this recovery, and experiments show that Oats predicts it more effectively than own-layer sensitivity. Based on Oats, we propose Stout, Structured Order-Aware Quantization Training, a two-stage framework for efficient QAT. The first stage computes Oats through shared measurements and trains layers with the model's output loss in ascending score order, preserving stronger adaptation opportunities for later. The second stage trains scales with fixed quantized weights using ScaleKernel, a fused backward kernel that avoids dense weight-gradient intermediates in global memory. On Qwen3.6-35B-A3B at 2bit, Stout improves LiveCodeBench pass@1 and the BFCL aggregate score by 0.87% and 2.68% over the development-selected ordering baseline. Compared with a matched compiled or fused baseline, ScaleKernel lowers second-stage training time and peak allocated GPU memory by 0.85% and 14.47%, respectively. Our code is available at `https://anonymous.4open.science/status/stout-613D`

1 INTRODUCTION

Large language models (LLMs) have made substantial progress across language tasks, but their growing memory requirements remain a major bottleneck for deployment. Weight quantization addresses this problem by representing model parameters at lower precision. However, quantization involves a tradeoff between accuracy and cost at low bit widths. Post-training quantization (PTQ) incurs relatively little computational cost but can suffer substantial accuracy degradation on complex tasks, such as agentic and coding tasks. Quantization-aware training (QAT) can achieve better performance by allowing the model to adapt to quantization errors through weight updates, but full-model training can demand substantial data, computation, and memory. This tradeoff motivates more efficient quantization procedures that preserve model performance.

Two-stage layerwise QAT reduces the cost of full-model training (Chen et al., 2025). The first stage trains one layer at a time while freezing the rest of the model. A common objective is layerwise reconstruction, which matches quantized layer outputs to their floating-point references on a compact calibration dataset (Shao et al., 2024; Chen et al., 2025). The QAT order typically follows the network from input to output, allowing each layer to adapt to errors in inputs from the quantized prefix. The second stage trains scales with an end-to-end loss while keeping quantized weights frozen (Kim et al., 2023; Chen et al., 2025). Together, these stages recover accuracy while avoiding the gradient and optimizer memory required to train all model weights simultaneously.

However, conventional input-to-output layerwise QAT methods **quantize and train layers in a fixed depth order**, without considering how well each layer can adapt to errors introduced by other quantized layers. Since each layer is frozen after training, this order may leave less adaptable layers to handle later quantization errors while more capable layers have already been frozen. This can limit

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.

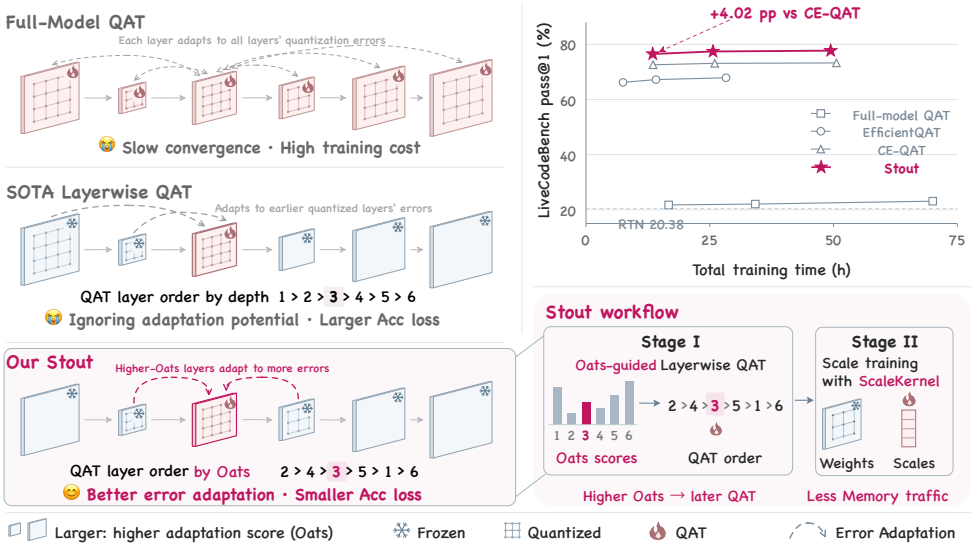

Figure 1: **Stout combines Oats-guided QAT with ScaleKernel.** higher-potential layers train later. ScaleKernel reduces tiles to avoid dense weight-gradient storage.

accuracy recovery, motivating **a layerwise QAT order guided by each layer's ability to adapt to errors from other quantized layers.**

To measure this adaptation ability, we introduce Oats, **O**rder-aware **a**dap**t**ation **s**core, which estimates how much a layer can recover from quantization errors introduced by other layers. Unlike previous sensitivity-based criteria, which assess the damage caused by quantizing a layer (Dong et al., 2019), Oats identifies layers that can provide greater recovery when trained after others. This makes it directly relevant to deciding which layers should be trained later. Our local analysis provides a rationale for Oats-guided training orders by relating the score to recoverable loss. Experiments show that Oats predicts measured recovery better than layer sensitivity, and that the resulting training order improves downstream performance over the tested sensitivity-based and depth-based orders.

Based on this score, we propose **Stout**, a two-stage **St**ructured **o**rder-aware q**u**antization **t**raining framework for layerwise QAT. In the first stage, Stout computes Oats for all layers through shared measurements, then places layers with higher scores later in the training order. Each layer undergoes QAT using the model's output loss, allowing it to adapt to quantization errors beyond its inputs. The second stage trains scales with fixed quantized weights. We introduce ScaleKernel, a fused backward kernel that reduces weight-gradient tiles on chip to avoid dense intermediates in global memory. Together, these components improve accuracy recovery and training efficiency, as illustrated in Fig. 1.

We evaluate Stout on Qwen3.6-35B-A3B and Llama-3.1-8B for code generation and tool use with 2bit weights, and additionally at 4bit on Qwen3.6-35B-A3B, using the same 2K examples randomly sampled from UltraData-SFT-2605 (OpenBMB, 2026) in both training stages, with a maximum sequence length of 64k tokens. At 2bit on Qwen3.6-35B-A3B, Stout improves LiveCodeBench pass@1 by 0.87 percentage points and the BFCL aggregate score by 2.68 percentage points over the ordering baseline selected on development data (Jain et al., 2025; Patil et al., 2025). Controlled recovery measurements support the adaptation ability captured by Oats, and the quality advantage persists against CE and reconstruction baselines at equal total training time. With the same micro-batch size and gradient accumulation, ScaleKernel lowers second-stage training time by 0.85% and peak allocated GPU memory by 14.47%.

2 Preliminaries and Challenges

2.1 Quantization-Aware Training

Quantization-aware training (QAT) simulates low-precision weights in the forward pass and updates floating-point (FP) weights W using a straight-through estimator (STE). For input X, scale $s > 0$,

and bit width b, the forward and backward passes of a linear layer with symmetric quantization are given by

$$Q(W) = \text{clip}\big(\lfloor W/s \rceil, -2^{b-1}, 2^{b-1}-1\big), \quad \hat{W} = sQ(W), \quad Y = X\hat{W}, \quad \frac{\partial \mathcal{L}}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \hat{W}}, \tag{1}$$

where Y is the layer output, $Q(\cdot)$ is an integer quantizer, $\hat{W}$ denotes dequantized weights, $\lfloor \cdot \rceil$ denotes rounding, and $\mathcal{L}$ is the end-to-end loss. The STE approximation applies to unclipped weights. Full-model QAT jointly updates FP weights and scales across all layers, coupling their adaptation and requiring substantial training resources (Chen et al., 2025).

2.2 Layerwise Quantization-Aware Training

Layerwise QAT reduces training cost by optimizing one layer or Transformer block at a time. Two-stage methods combine this approach with end-to-end scale training (Chen et al., 2025). The first stage trains layers from input to output, keeping all but the current layer frozen. Following local reconstruction methods (Shao et al., 2024), it optimizes the current layer's weights W_l and scales s_l using

$$\min_{W_l, s_l} \left\| f_l(\hat{W}_l, \hat{X}_l) - f_l(W_l, X_l) \right\|^2, \tag{2}$$

where $f_l(W, X)$ is the l-th layer mapping with weights W and input X. The inputs $\hat{X}_l$ and X_l come from the quantized prefix and the original FP model, respectively. The reference output on the right is evaluated using the pretrained weights and held fixed during training.

For an L-layer model, let $\mathcal{W}_S$ denote its weights with layers in set S quantized: layer l uses $\hat{W}_l$ if $l \in S$ and W_l otherwise, *i.e.*,

$$[\mathcal{W}_S]_l = \begin{cases} \hat{W}_l, & l \in S, \\ W_l, & l \notin S, \end{cases} \tag{3}$$

where $\mathcal{W}_\emptyset = (W_1, \ldots, W_L)$ denotes the FP model with no quantized layers. In the second stage, scales s across the model are optimized using the end-to-end cross-entropy (CE) loss while the quantized weights $Q(W)$ remain frozen (Kim et al., 2023; Chen et al., 2025), *i.e.,*

$$\min_{s} \ \mathcal{L}(\mathcal{W}_{\{1,\ldots,L\}}), \quad \mathcal{W}_{\{1,\ldots,L\}} = (\hat{W}_1, \ldots, \hat{W}_L), \tag{4}$$

where s denotes the scales across all layers.

2.3 Challenges in Layerwise QAT

Despite their efficiency, previous layerwise QAT methods face three challenges. The first challenge is **objective mismatch**. In the first stage, local reconstruction in Eq. 2 does not directly minimize the model's prediction loss. This motivates training layer l with the end-to-end CE loss:

$$\min_{W_l, s_l} \ \mathcal{L}(\mathcal{W}_{\{1,\ldots,l\}}), \quad \mathcal{W}_{\{1,\ldots,l\}} = (\hat{W}_1, \ldots, \hat{W}_l, W_{l+1}, \ldots, W_L). \tag{5}$$

Under the input-to-output order, layers 1 through $l-1$ are quantized and frozen, while layers $l+1$ through L remain in FP and are also frozen. Gradients pass through these downstream layers to train the current layer. With a different training order, this output-loss gradient also lets the current layer respond to errors in already quantized downstream layers. It provides the training setting in which we study order, while adding suffix-backpropagation cost to each visit.

The second challenge is **quantization order**. During the first stage, each layer undergoes QAT and is then frozen, so its updates can respond only to quantization errors present during its training. Previous layerwise QAT methods typically follow a fixed input-to-output order, leaving later layers to compensate for earlier layers' errors without considering their capacity to do so. If these layers cannot compensate sufficiently, the remaining quantization errors reduce the final model accuracy.

The third challenge is the **memory overhead of scale training**. Although only scales are updated in the second stage, conventional backward implementations, including compiled paths in PyTorch's TorchInductor, can still materialize dense weight gradients before reducing them to scale gradients. Writing and reading these intermediates adds memory traffic and storage overhead, which can limit training efficiency.

3 METHOD

To address fixed quantization order and the memory overhead of scale training, we propose Stout, which introduces Oats and ScaleKernel. Oats guides the training order according to each layer's estimated ability to adapt to other layers' quantization errors, while ScaleKernel fuses scale-gradient reduction to avoid storing dense weight gradients in global memory during scale training.

3.1 FROM TRAINING ORDER TO RECOVERABLE LOSS

Training a layer later allows it to adapt to errors from more previously quantized layers. To determine which layers benefit most from this opportunity, we ask how much additional loss each layer could recover after other layers are quantized. We first isolate the resulting training signal: let S contain all layers except l, and compare $\mathcal{W}_\emptyset$ with $\mathcal{W}_S$ at the same weights, keeping layer l in FP. The gradient difference is

$$\Delta g_l = \nabla_{W_l}\mathcal{L}(\mathcal{W}_S) - \nabla_{W_l}\mathcal{L}(\mathcal{W}_\emptyset), \quad S = \{1, \ldots, l-1, l+1, \ldots, L\}, \tag{6}$$

which removes the gradient already present in the FP model and captures the additional response to other layers' quantization. Gradients are flattened when used in quadratic expressions.

A large gradient response alone does not imply large recovery: the loss curvature determines how far a layer can adapt before further updates stop helping. To translate Δg_l into recoverable loss, we model the two losses locally as quadratics with shared positive-definite curvature H_l with respect to W_l, holding other layers fixed. The following theorem quantifies the recovery available when a layer that has adapted to the reference state is allowed to adapt again after external quantization.

Theorem 1 (Conditional recovery)**.** *For two local quadratic loss models with shared positive-definite curvature H_l and gradient difference Δg_l, adapting layer l from the reference optimum to the new optimum reduces the post-quantization loss by*

$$C_l^* = \tfrac{1}{2}\Delta g_l^\top H_l^{-1}\Delta g_l. \tag{7}$$

At the reference optimum, the new quadratic loss has gradient Δg_l. The update $-H_l^{-1}\Delta g_l$ reaches its new optimum, yielding the recovery C_l^*. Thus, the score accounts for both the adaptation demanded by other layers' errors and the curvature that limits this adaptation. Freezing the layer forfeits this recovery opportunity. The identity also applies to comparisons that keep the target layer quantized in both states.

The theorem compares the two local optima; it does not require QAT to follow this update. Appendix B.2 gives the proof and a finite-update interpretation. Appendix B.3 extends the identity to unequal curvatures and bounds the deviation using curvature change and remaining reference adaptation. The controlled experiments in Tab. 12 connect this local recovery criterion to finite-budget AdamW training.

3.2 OATS: ORDER-AWARE ADAPTATION SCORE

However, computing C_l^* directly is expensive for two reasons: (1) measuring Δg_l separately for each layer requires a different mixed-quantization model state; (2) forming and inverting the dense curvature H_l is impractical at LLM scale. Oats makes this recovery criterion tractable through shared gradient measurements and a diagonal curvature estimate, as shown in Fig. 2.

Sharing measurements of external gradient responses. We use round-to-nearest (RTN) quantization to expose initial errors without training the layers whose order we need to select. To measure their effects together, we randomly partition layers into balanced sets A and B. For a layer $l \in A$, we compare the all-quantized model with a model in which only A is quantized:

$$\Delta\tilde{g}_l = \nabla_{W_l}\mathcal{L}(\mathcal{W}_{A\cup B}) - \nabla_{W_l}\mathcal{L}(\mathcal{W}_A), \qquad l \in A. \tag{8}$$

This difference isolates the response to quantizing B. Reversing the sets gives the responses for layers in B, so three shared gradient computations cover all layers. Each evaluation accumulates gradients over the same scoring data with weights and quantizers fixed. Compared with Eq. 6, the target stays quantized in both states and only the opposite set supplies the added errors. The first

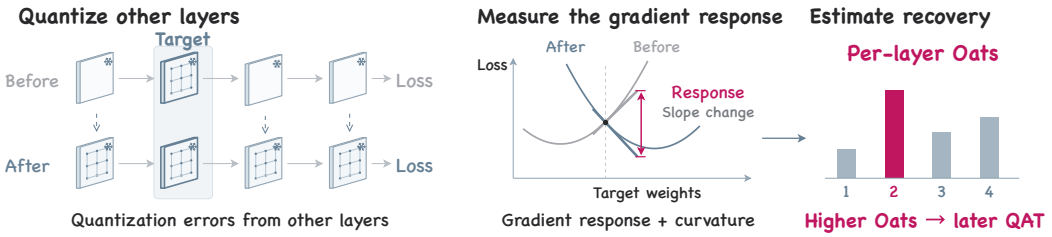

Figure 2: **Oats estimates cross-layer recovery potential.** Quantizing other layers changes the loss slope at fixed target weights. This response, weighted by curvature, motivates the per-layer score. The curves illustrate the response; shared measurements are detailed in the text.

choice matches the target's QAT mode; the second lets layers share model evaluations. These measurements apply the same conditional-recovery principle to a different pair of quantization states.

Efficient approximation of H_l by $\widetilde{H}_l$. We approximate curvature with the model Fisher, which equals the CE curvature of locally linearized logits under the same STE Jacobian. This provides a positive semidefinite curvature model without second derivatives. We estimate its diagonal through ordinary backward passes using labels sampled from the model's predictions. These gradients have zero mean, so cross-token terms vanish in expectation when the labels are sampled independently. The following lemma gives the resulting estimator.

Lemma 1 (Unbiased Fisher-diagonal estimation)**.** *Fix the all-quantized model and a batch with $T > 0$ valid target tokens. Let p_t be its prediction distribution at token t, and independently sample $\tilde{y}_t \sim p_t$ while keeping the input prefixes fixed. For the token-averaged model Fisher $\mathcal{F}_l$ of layer l,*

$$r_l = \frac{1}{\sqrt{T}} \sum_{t=1}^{T} \nabla_{W_l}[-\log p_t(\tilde{y}_t)], \qquad \mathbb{E}[r_l \odot r_l] = \operatorname{diag}(\mathcal{F}_l), \tag{9}$$

where $\odot$ denotes elementwise multiplication and the expectation is over the sampled labels.

A fourth shared computation estimates Fisher curvature, obtaining r_l for every layer in a batch with one backward pass. Across batches, we weight the squared estimates by valid-token counts, average their entries within each quantization group, and add positive damping. This gives $\widetilde{H}_l$, a positive diagonal curvature approximation that retains differences across weight groups while storing only one value per group. Applying its inverse requires division rather than matrix inversion. Appendix B.5 gives the proof and accumulation details.

From the measurements to recovery potential. Combining the shared response and diagonal curvature gives the practical score

$$C_l = \tfrac{1}{2}(\Delta\tilde{g}_l)^\top \widetilde{H}_l^{-1} \Delta\tilde{g}_l. \tag{10}$$

For K random partitions, we average the scores while reusing the all-quantized gradients and curvature estimate.

The score retains the structure of C_l^*: the gradient difference measures adaptation demand, and inverse curvature weights the recovery it permits. The remaining question is what information survives when each layer sees only part of the external quantization error. The following proposition characterizes this under an additive response model. Write $\|v\|^2_{\widetilde{H}_l^{-1}} = v^\top \widetilde{H}_l^{-1} v$.

Proposition 1 (Recovery information retained by shared measurements)**.** *For even $L \geq 2$, suppose quantizing a set S_l of other layers induces $\Delta\tilde{g}_l = \sum_{i \in S_l} v_i$, where each v_i is a fixed response to layer i. Fix $\widetilde{H}_l \succ 0$. If S_l is the set opposite layer l in a uniformly random balanced partition, then*

$$\mathbb{E}[C_l] = \frac{L}{8(L-1)} \left(\sum_{i \neq l} \|v_i\|^2_{\widetilde{H}_l^{-1}} + \Big\| \sum_{i \neq l} v_i \Big\|^2_{\widetilde{H}_l^{-1}} \right). \tag{11}$$

The first term retains the individual effects of other layers; the second retains their combined effect, including reinforcement and cancellation. Thus, shared measurements preserve both kinds of adaptation demand in expectation without constructing a full interaction matrix. Appendix B.4 gives the

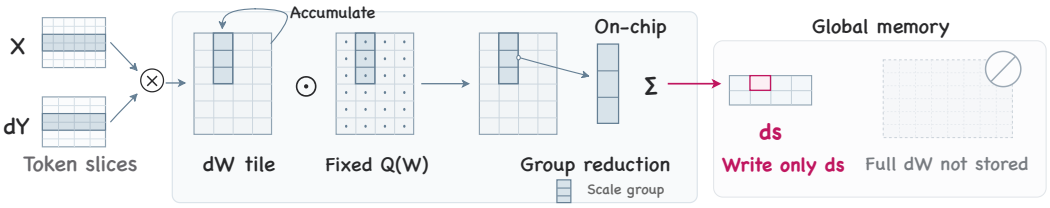

Figure 3: **ScaleKernel reduces gradients before writing them.** Token slices accumulate a dW tile on chip, where fixed quantized weights map it to groupwise scale gradients. Only ds is written to global memory; the full dW buffer is avoided.

proof. Together with Theorem 1, this motivates C_l as a sampled conditional-recovery score, rather than an unbiased estimate of the FP, all-other-layer C_l^*. More partitions reduce sampling variation around this target; they do not remove the difference between the targets. The experiments in Fig. 8 and Tab. 13 show that the shared measurements retain useful ordering information across the tested quantization environments and source sets. Tab. 14 separately evaluates the curvature approximation.

3.3 Stage 1: Layerwise QAT in Score Order

The recovery analysis suggests reserving layers with greater adaptation potential for later, when more quantization errors are present. Stout therefore selects one fixed ascending order of the initial scores C_l in Eq. 10, breaking ties by the original depth. These scores describe recovery in the initial measurement environment, so their ranking can change as layers are quantized and trained. We assess this transfer through environment and drift measurements in Fig. 8 and Tab. 15. The complete-training comparison in Tab. 11 separately demonstrates the benefit of ascending scores over descending and fixed-depth orders. Appendix B.6 explains why the local analysis motivates, but does not guarantee, a globally optimal order.

After measurement, all layers return to FP mode. Each layer then undergoes its full QAT budget with the CE loss $\mathcal{L}(\mathcal{W}_S)$, where S contains the previously trained layers and the current layer. This extends Eq. 5 to the selected training order. Previously trained layers remain quantized and frozen; unvisited layers remain FP and frozen. Only the current layer is updated, with gradients passing through the downstream layers. The network's forward order is unchanged. Algorithm 1 summarizes both training stages; Appendix B.8 gives the execution details.

3.4 Stage 2: Scale Training with ScaleKernel

After layerwise QAT, the second stage trains scales with fixed quantized weights, as in Eq. 4. Its small number of trainable parameters does not automatically make backward memory-efficient: an unfused backward path can materialize a gradient for every weight before reducing these values to the much smaller set of scale gradients. ScaleKernel performs this reduction on chip, avoiding the dense intermediate in global memory. It applies to scale training regardless of the preceding layer order, addressing a separate source of training cost.

Consider the symmetric form $\hat{W} = sQ(W)$ from Section 2, with one scale shared by the weights. For $Y = X\hat{W}$, the chain rule gives

$$\frac{\partial \mathcal{L}}{\partial s} = \left\langle Q(W), X^\top \frac{\partial \mathcal{L}}{\partial Y} \right\rangle, \qquad \frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial Y} \hat{W}^\top. \tag{12}$$

Writing $dY = \partial \mathcal{L}/\partial Y$, let $dW = X^\top dY$ denote the gradient with respect to the dequantized weights $\hat{W}$. The scale gradient is a fixed weighted sum of entries in dW, so each tile can be reduced before it is written to global memory. Fig. 3 contrasts the full dW buffer avoided by ScaleKernel with the compact scale gradients it writes.

ScaleKernel assigns complete quantization groups to each weight tile. For a token tile $\mathcal{T}$, let $X_\mathcal{T}$ and $dY_\mathcal{T}$ be the corresponding input and output-gradient blocks. A tile-local FP32 accumulator D

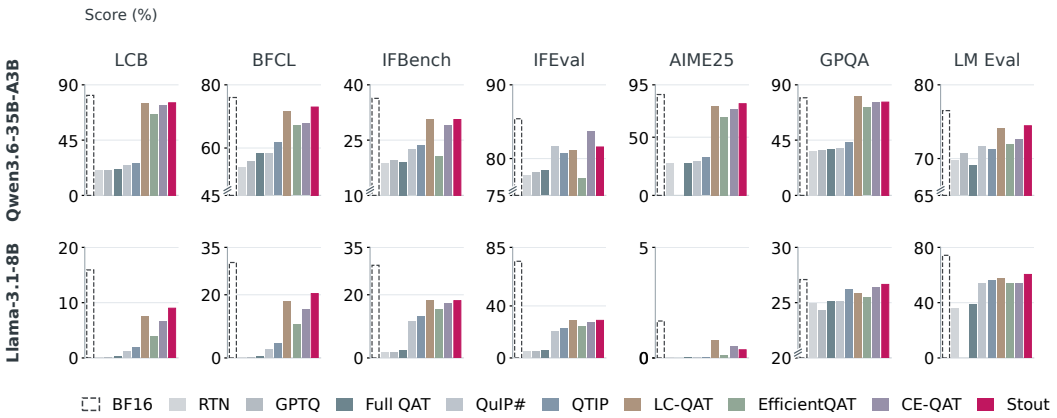

Figure 4: **Performance of 2bit quantization across two models and seven task groups.**

collects their matrix products, then reduces directly to the tile's scale gradients:

$$D = \sum_{\mathcal{T}} X_{\mathcal{T}}^{\top} dY_{\mathcal{T}}, \qquad ds_{\text{tile}} = \text{groupsum}(D \odot Q(W)_{\text{tile}}) \,. \tag{13}$$

Here groupsum sums the input-channel entries sharing each scale, and $Q(W)_{\text{tile}}$ contains the corresponding fixed quantized weights. Each tile spans complete groups, so its output is written directly to the scale-gradient tensor. The full weight-gradient matrix is never assembled in global memory.

For P weights in groups of size G, ScaleKernel writes P/G scale gradients and avoids the dense P-element weight-gradient buffer and its write/read pair. The matrix-multiplication arithmetic and the complete input gradient in Eq. 12 remain, preserving the training signal to earlier layers. Appendix B.9 derives the tiled reduction, gives the loop-level algorithm and optional token splitting, and explains finite-precision rounding. The same-checkpoint comparison in Tab. 10 demonstrates the time and memory savings over the complete second stage. The component ablation in Tab. 2 separates these system gains from the quality benefit of Oats order.

4 Experiment

We evaluate the performance and training efficiency of Stout against low-bit quantization baselines. Further implementation details are provided in Appendix C.

4.1 Experiment Setup

Models, Data, and Metrics. We use Qwen3.6-35B-A3B and Llama-3.1-8B as our base models (Qwen Team, 2026; Team, 2024). Both training stages use the same 2K examples randomly sampled from UltraData-SFT-2605 (OpenBMB, 2026). We evaluate both models at 2bit and Qwen3.6-35B-A3B additionally at 4bit on LiveCodeBench (LCB), BFCL, IFBench, IFEval, AIME25, and GPQA (Jain et al., 2025; Patil et al., 2025; Pyatkin et al., 2025; Zhou et al., 2023). We also use the LM Evaluation Harness to evaluate five zero-shot tasks: ARC-Challenge, ARC-Easy, HellaSwag, PIQA, and WinoGrande (Biderman et al., 2024). Their average is reported as LM Eval.

Quantization Baselines. We compare Stout with QuIP#, QTIP, and LC-QAT (Tseng et al., 2024a;b; Wang et al., 2026), as well as EfficientQAT, forward CE-QAT, and full-model QAT (Chen et al., 2025). BF16 provides the full-precision reference, while RTN and GPTQ (Frantar et al., 2022) provide training-free references. For a fair comparison, controlled QAT methods share the initial checkpoint, quantizer, and training data. Controlled layerwise methods use the same update budget, while full-model QAT matches their cumulative training tokens. Native-format baselines retain their original quantizers and recipes.

Implementation Details. We use group-wise quantization with group size 128 and a maximum sequence length of 64k. Oats uses 64 training examples, one complementary partition, and one curvature estimate. Hyperparameters and baseline ordering directions are selected on development

	Coding	Agent	Instruction		Reasoning		LM Eval				
Method	LCB	BFCL	IFBench	IFEval	AIME25	GPQA	ARC-C	ARC-E	Hella.	PIQA	Wino.
BF16	81.42	75.99	36.33	85.40	86.67	79.47	62.37	79.55	82.43	83.03	75.14
RTN	77.65	71.65	32.67	82.78	82.72	74.28	60.01	77.89	78.13	79.54	70.51
GPTQ	79.31	73.95	36.00	82.99	83.23	74.67	59.99	75.93	79.43	83.12	72.75
Full-model QAT	77.63	73.95	33.00	83.21	82.76	74.86	61.16	77.18	80.61	83.07	69.60
QuIP#	80.40	76.12	34.40	83.69	83.94	78.95	62.03	80.04	81.91	83.26	71.34
QTIP	81.18	74.46	34.14	81.96	84.64	78.86	62.72	78.41	79.72	79.60	73.23
EfficientQAT	78.56	72.66	33.67	83.09	83.62	77.19	62.12	80.89	79.17	81.56	71.98
CE-QAT	79.21	74.24	36.33	84.01	85.77	79.03	62.12	78.49	79.15	81.83	72.06
Stout	**81.53**	**77.26**	**36.25**	**85.65**	**86.38**	**80.92**	**62.80**	**78.53**	**81.89**	**82.12**	**74.51**

Table 1: **Performance of 4bit quantization on Qwen3.6-35B-A3B.**

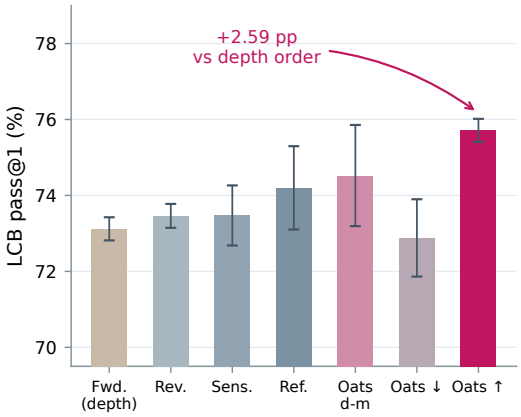

Figure 5: **The score and its direction matter.** Qwen 2bit experiments with different layer orders of QAT.

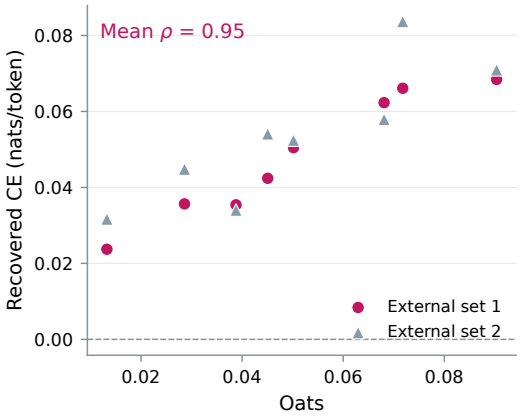

Figure 6: **Recovery increases with Oats.** Each point is a receiver/source-set pair after full-budget adaptation.

data. We report mean and standard deviation when repeated runs are available. Appendix C.1 provides the training and evaluation settings.

4.2 Performance Evaluation

Performance Comparison with Baselines. To evaluate the performance of Stout, we compare it with various quantization baselines across different models and tasks. As shown in Fig. 4 and Tab. 1, Stout leads the quantized baselines on LCB, BFCL, and the LM Eval average across all evaluated settings. On Qwen at 2bit, it improves LCB accuracy by 0.41% over the best SOTA baseline, though Appendix C.10 shows that this particular gap is within training-seed noise, while the BFCL gap over the same baseline is not. At 4bit on Qwen, Stout remains close to the full-precision reference. These gains support the benefit of training more adaptable layers later, allowing them to recover errors introduced by other quantized layers. The advantage is not uniform across individual tasks; Appendix C.3 analyzes where it does not hold. At 2bit, LCB collapses to near zero for RTN and GPTQ on both models, confirming that adaptation is essential for complex generation tasks at this bitwidth. Full-model QAT stays close to RTN: with only 2K training examples, jointly updating every weight provides far too few gradient steps per parameter, whereas layerwise training concentrates the same budget on one block at a time.

Effectiveness of Oats. To evaluate the effectiveness of Oats, we compare different QAT orders for layer on Qwen3.6-35B-A3B model, including forward, reverse, sensitivity-based, reference-potential, and ascending/descending Oats orders under the same CE training settings. As shown in Fig. 5, ascending Oats achieves the best performance. It outperforms the development-selected ordering baseline by 0.87% on LCB and 2.68% on BFCL. Reversing the order increases held-out

Order	Backend	LCB	Stage (h)	GPU
Forward	Compiled	72.78	1.72	47.9
Forward	ScaleKernel	72.50	1.67	41.6
Oats	Compiled	75.96	1.71	49.9
Oats	ScaleKernel	75.95	1.68	37.9

Table 2: **Separate the two contributions.** Qwen 2bit. Each backend pair starts from its order's stage-one checkpoint.

Method	Score (h)	Total (h)	LCB
MSE	0.00	13.66	71.08
CE	0.00	13.66	73.47
Stout	0.86	13.66	75.70

Table 3: **Match total training time.** Qwen 2bit. CE denotes forward CE-QAT; MSE denotes reconstruction. Scoring remains part of the common budget.

CE by 0.049 nats per token. These results support placing layers with greater adaptation potential later in training.

Relation to Recovery Potential. We quantize other layers and measure how much loss a target layer recovers through adaptation. Fig. 6 shows the results under the full training budget. Higher Oats scores correspond to greater recovery, with a mean within-source Kendall correlation of 0.857, compared with -0.071 for layer sensitivity. This supports using recovery potential to guide training order. Since deeper layers tend to have both larger Oats and more headroom, we also measure recovery on receivers stratified by depth, and include a depth-sorted ordering arm in Fig. 5. Ascending Oats improves LCB by 2.59 points over sorting by depth alone, and the association with recovery survives partialling out depth (0.954). Appendix C.9 reports both comparisons; additional budgets and estimator analyses are in Appendix C.4 and Appendix C.5.

4.3 Efficiency Evaluation

End-to-End Training Efficiency. We compare final LCB accuracy against total training time, including scoring, both training stages, compilation, communication, and export. As shown in Fig. 7, the first Stout schedule completes in 11.07 hours, with Oats accounting for 7.42% of the total. At the same training time, Stout outperforms forward CE-QAT by 2.24% on LCB, as reported in Tab. 3. This improvement shows that the ordering benefit offsets its measurement cost.

Scale-Training Efficiency. We compare ScaleKernel with eager autograd, compiled autograd, and an existing fused backend using the same stage-one checkpoint and training settings. Tab. 10 reports step latency, stage time, and peak memory. At 2bit, ScaleKernel reduces peak memory by 14.47% over the fastest compatible fused reference, with a 0.85% reduction in stage time. The memory saving comes from reducing weight-gradient tiles on chip before writing scale gradients. The matrix-multiplication work remains, limiting the runtime improvement.

Separating Order and Kernel Benefits. Tab. 2 compares forward and Oats orders with compiled scale gradients and ScaleKernel. With Oats order fixed, ScaleKernel changes LCB accuracy by only -0.014% while reducing memory. Together with the ordering ablation, this supports attributing the accuracy gain to Oats and the memory saving to ScaleKernel. Further cost analyses are provided in Appendix C.8.

5 Conclusion

We presented Stout, a two-stage framework that improves layerwise QAT through adaptation-guided order selection and efficient scale training. Its Oats score estimates how much a layer can recover from quantization errors introduced by other layers. Shared measurements make this score practical to compute for all layers before training, allowing Stout to preserve stronger adaptation opportunities for later. A local quadratic analysis explains the score, while controlled recovery measurements and complete-model comparisons connect it to improved low-bit code generation and tool use. Comparisons at equal total training time account for both the output-loss objective and order measurement. Independently of order, ScaleKernel lowers the time and peak GPU memory required to train scales with fixed quantized weights. These two contributions improve when layers adapt and how scale gradients are computed.

AI USE STATEMENT

Generative AI tools assisted with literature organization, research brainstorming, mathematical and implementation review, manuscript drafting, and figure preparation. The authors remain responsible for the accuracy of the technical content, attribution of prior work, experimental analysis, and final presentation.

REFERENCES

Juan Amboage, Pablo Monteagudo-Lago, Ian Colbert, Giuseppe Franco, and Nicholas Fraser. Optimal post-training quantization scales and where to find them. *arXiv preprint arXiv:2606.10890*, 2026.

Yamato Arai and Yuma Ichikawa. Quantization error propagation: Revisiting layer-wise post-training quantization. *Advances in Neural Information Processing Systems*, 38:151916–151951, 2026.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *CoRR*, abs/2405.14782, 2024. doi: 10.48550/ARXIV.2405.14782. URL https://doi.org/10.48550/arXiv.2405.14782.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10081–10100, 2025.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 293–302, 2019.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/ARXIV.2210.17323. URL https://doi.org/10.48550/arXiv.2210.17323.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *Proceedings of Machine Learning and Systems*, 2:112–128, 2020.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in neural information processing systems*, 36:36187–36207, 2023.

Jonghoon Kwak, Kyungho Kim, Sang-Seol Lee, Sung-Joon Jang, and Jonghee Park. Quantization aware training with order strategy for cnn. In *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–3. IEEE, 2022.

Gunjun Lee, Sehwan Son, Younjoo Lee, Byungjun Kim, and Jung Ho Ahn. Schurquant: Groupwise discrete optimization for layer-wise llm quantization. *arXiv preprint arXiv:2608.15567*, 2026.

Qijun Luo, Hengxu Yu, and Xiao Li. Badam: A memory efficient full parameter optimization method for large language models. *Advances in Neural Information Processing Systems*, 37: 24926–24958, 2024.

Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtarik. Pv-tuning: Beyond straight-through estimation for extreme llm compression. *Advances in Neural Information Processing Systems*, 37:5074–5121, 2024.

NVIDIA. Custom epilogue fusions for GEMMs. NVIDIA CUTLASS Documentation, 2026. URL https://docs.nvidia.com/cutlass/latest/media/docs/operators/tutorials/001_gemm_with_fused_epilogue.html. Last updated August 31, 2026; accessed September 16, 2026.

OpenBMB. UltraData-SFT-2605. Hugging Face dataset repository, 2026. URL https://huggingface.co/datasets/openbmb/UltraData-SFT-2605.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/46499a0622ecf568b72d17b61e45dbd5-Abstract-Datasets_and_Benchmarks_Track.html.

Qwen Team. Qwen3.6-35B-A3B Model Card. Hugging Face, 2026. URL https://huggingface.co/Qwen/Qwen3.6-35B-A3B. Accessed September 23, 2026.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

John Sweeney. The geometry of sequential learning: Lie-bracket prediction of transfer order. *arXiv preprint arXiv:2606.24993*, 2026.

Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48630–48656. PMLR / OpenReview.net, 2024a. URL https://proceedings.mlr.press/v235/tseng24a.html.

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. QTIP: quantization with trellises and incoherence processing. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/6de2e84b8da47bb2eb5e2ac96c63d2b0-Abstract-Conference.html.

Haoyu Wang, Xingyu Yu, Haiyan Zhao, Fengxiang Wang, and Xu Han. Lc-qat: Data-efficient 2-bit qat for llms via linear-constrained vector quantization. *arXiv preprint arXiv:2606.10531*, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. doi: 10.48550/arXiv.2311.07911. URL https://arxiv.org/abs/2311.07911v1.

A Related Work

Layerwise quantization and task adaptation. OmniQuant learns clipping and equivalent transformations, while EfficientQAT trains layer parameters before end-to-end scale training (Shao et al., 2024; Chen et al., 2025). PEQA further demonstrates task adaptation by training scales with fixed quantized weights (Kim et al., 2023). QEP explicitly propagates and compensates for quantization error across layers (Arai & Ichikawa, 2026). Cross-layer compensation is therefore established; our question is how to schedule each layer's single QAT opportunity to make better use of it. Block-coordinate optimization of the task loss also appears outside quantization in BAdam (Luo et al., 2024). Stout retains layerwise training and scale training, and changes their scheduling and gradient computation.

Training order and adaptation. HAWQ combines curvature and quantization displacement to guide mixed precision and fine-tuning order (Dong et al., 2019), and order strategies for QAT have been studied explicitly (Kwak et al., 2022). General sequential-learning predictors also analyze interactions between gradients and curvature (Sweeney, 2026). Oats estimates the adaptation available when a layer remains trainable after other layers are quantized, using matched measurements that keep the receiving layer's quantization fixed. Its local inverse-curvature recovery formula follows the same compensation principle used in Optimal Brain Surgeon (Hassibi & Stork, 1992) and in SchurQuant's elimination of remaining continuous variables (Lee et al., 2026). SchurQuant uses this principle for discrete weight-group decisions within a linear layer's reconstruction problem; Stout uses shared model-level measurements to schedule Transformer layers under the task loss. The classical quadratic identity supplies the interpretation, while the intervention, shared estimator, and training schedule define the proposed method.

Partial quantization and scale optimization. Quant-Noise trains with randomly quantized subsets to improve robustness to compression (Fan et al., 2020). Stout uses complementary quantization masks to measure Oats, with no parameter updates, and scores each receiver only in states where its own quantization is unchanged. All layer scores then produce one complete training order. PiSO studies data-aware scale selection for round-to-nearest quantization, and SchurQuant alternates discrete group optimization with affine-grid optimization (Amboage et al., 2026; Lee et al., 2026). PV-Tuning also optimizes discrete compressed parameters (Malinovskii et al., 2024). Our second stage keeps quantized weights and zero points fixed and trains only scales.

B Derivations and Implementation Details

Algorithm 1 Stout: measure adaptation, select order, and train

Require: Initial model, training data, scoring data, K partitions
Ensure: Quantized model with trained scales

Before training: measure Oats without parameter updates

Initialize quantizers; set $C_l \leftarrow 0$ for every layer
Measure all-quantized gradients and estimate $\widetilde{H}_l$ on the scoring data
for each of the K random balanced partitions **do**
    Measure gradients of the quantized set, with the other set in floating point
    Reverse the sets and repeat ▷ Keep each measured layer quantized
    Compute every $\Delta\tilde{g}_l$ using Eq. 8
    Add $1/K$ of the score in Eq. 10 to each C_l
end for

Stage 1: layerwise QAT

Restore floating-point modes and sort layers by increasing C_l
for each layer in the selected order **do**
    Activate its quantizer and train only this layer with CE on the full data
    Freeze the layer in quantized mode
end for

Stage 2: scale training

Train scales on the full data with fixed quantized weights, using ScaleKernel

B.1 The Measurement Target

For a dataset $\mathcal{D}$, let $\mathcal{V}(x)$ contain the valid target positions after the causal shift and loss masking. For a model weight configuration $\mathcal{W}$, we use the token-mean cross-entropy

$$\mathcal{L}_{\mathcal{D}}(\mathcal{W}) = \frac{1}{T_{\mathcal{D}}} \sum_{x \in \mathcal{D}} \sum_{t \in \mathcal{V}(x)} -\log p_{\mathcal{W}}(y_t \mid x_{<t}), \qquad T_{\mathcal{D}} = \sum_{x \in \mathcal{D}} |\mathcal{V}(x)|. \tag{14}$$

Masked targets remain in the causal context. Let $\mathcal{P}$ denote the data used to measure scores and $\mathcal{D}_{\text{train}}$ the training data. Appendix C.1 specifies their construction, context lengths, and label masks. Independent evaluation data determine recovery and downstream performance, rather than selecting the order on the test set.

For scoring, W_l collects only the latent weights of the target quantized linear modules in layer l. Gradients and updates are vectorized when used in quadratic forms; this does not change their matrix layout in the implementation. A balanced partition divides all target layers into sets A and B. Using the initial round-to-nearest quantizers, $\mathcal{W}_{A \cup B}$ quantizes all target layers, while $\mathcal{W}_A$ and $\mathcal{W}_B$ quantize only the indicated set and leave the other set in floating point. Each layer's reference state keeps its own set quantized, so the layer remains quantized in both states. The token-mean gradients are

$$g_l^Q = \nabla_{W_l} \mathcal{L}_{\mathcal{P}}(\mathcal{W}_{A \cup B}), \qquad g_l^{\text{ref}} = \begin{cases} \nabla_{W_l} \mathcal{L}_{\mathcal{P}}(\mathcal{W}_A), & l \in A, \\ \nabla_{W_l} \mathcal{L}_{\mathcal{P}}(\mathcal{W}_B), & l \in B. \end{cases} \tag{15}$$

For partition r, the response to quantizing the opposite set is

$$\Delta \tilde{g}_l^{(r)} = g_l^Q - g_l^{\text{ref},(r)}. \tag{16}$$

The measured responses $\Delta \tilde{g}_l^{(r)}$ apply the conditional comparison in the receiver's QAT mode, with a sampled external set. They are computed using the prescribed QAT backward without updating any parameter. In the theoretical quadratic model, g_l^{ref} and Δg_l are ordinary gradients of continuous local coordinates. This separation distinguishes the exact surrogate-model statements from their application to real fake-quantized training.

B.2 Conditional Recovery and Curvature Approximation

Proof of Theorem 1. Let u be a continuous update to layer l, measured from the same parameter value in both states. The two local models share positive-definite curvature H_l:

$$\begin{aligned} q_0(u) &= c_0 + (g_l^{\text{ref}})^\top u + \tfrac{1}{2} u^\top H_l u, \\ q_1(u) &= c_1 + (g_l^{\text{ref}} + \Delta g_l)^\top u + \tfrac{1}{2} u^\top H_l u. \end{aligned} \tag{17}$$

Here c_0, c_1 are constants; any shared quadratic regularization is included in H_l. Stationarity gives

$$u_0^* = -H_l^{-1} g_l^{\text{ref}}, \qquad u_1^* = -H_l^{-1}(g_l^{\text{ref}} + \Delta g_l). \tag{18}$$

Positive definiteness makes both minimizers unique. Completing the square in the perturbed model yields

$$q_1(u) = q_1(u_1^*) + \tfrac{1}{2}(u - u_1^*)^\top H_l (u - u_1^*). \tag{19}$$

Substituting $u_0^* - u_1^* = H_l^{-1} \Delta g_l$ gives $q_1(u_0^*) - q_1(u_1^*) = \tfrac{1}{2} \Delta g_l^\top H_l^{-1} \Delta g_l$. □

The identity does not require $g_l^{\text{ref}} = 0$ at the measurement point; it does use the reference optimum u_0^* as the starting point for defining additional recovery. It is an exact result for these quadratics, not a claim that a finite QAT visit reaches either optimum.

A finite-update interpretation. At u_0^*, the perturbed model has gradient Δg_l. For a positive-definite approximation $\widetilde{H}_l$ and the update $v = -\eta \widetilde{H}_l^{-1} \Delta g_l$, its exact quadratic decrease is

$$\begin{aligned} q_1(u_0^*) - q_1(u_0^* + v) = \eta \Delta g_l^\top \widetilde{H}_l^{-1} \Delta g_l \\ - \tfrac{\eta^2}{2} \Delta g_l^\top \widetilde{H}_l^{-1} H_l \widetilde{H}_l^{-1} \Delta g_l. \end{aligned} \tag{20}$$

When $\widetilde{H}_l = H_l$, this is $(2\eta - \eta^2)C_l^*$ with $C_l^* = \frac{1}{2}\Delta g_l^\top H_l^{-1}\Delta g_l$. For $0 < \eta < 2$, the same potential therefore determines the recovery of this preconditioned step. This connection requires the stated update rule and starting point, rather than an arbitrary Adam trajectory.

If $(1-\varepsilon)\widetilde{H}_l \preceq H_l \preceq (1+\varepsilon)\widetilde{H}_l$ with $0 \le \varepsilon < 1$, inversion reverses the positive-definite order. Consequently, for $\widehat{C}_l = \frac{1}{2}\Delta g_l^\top \widetilde{H}_l^{-1}\Delta g_l$,

$$\frac{\widehat{C}_l}{1+\varepsilon} \le C_l^* \le \frac{\widehat{C}_l}{1-\varepsilon}. \tag{21}$$

A group-diagonal Fisher approximation does not automatically satisfy this spectral condition. The inequality states what accuracy would suffice; it does not certify the practical approximation or preserve every layer ranking.

Existing optimization needs. The difference between the two states' total quadratic training gains is

$$\begin{aligned} &\tfrac{1}{2}(g_l^{\mathrm{ref}} + \Delta g_l)^\top H_l^{-1}(g_l^{\mathrm{ref}} + \Delta g_l) - \tfrac{1}{2}(g_l^{\mathrm{ref}})^\top H_l^{-1} g_l^{\mathrm{ref}} \\ &\quad = C_l^* + (g_l^{\mathrm{ref}})^\top H_l^{-1}\Delta g_l. \end{aligned} \tag{22}$$

Using the same measured gradients and $\widetilde{H}_l$, we record the diagnostic

$$\Delta\mathcal{L}_l^{\mathrm{gain}} = C_l + \frac{1}{K}\sum_{r=1}^{K}(g_l^{\mathrm{ref},(r)})^\top \widetilde{H}_l^{-1}\Delta\tilde{g}_l^{(r)}. \tag{23}$$

This quantity can be negative and asks a different question from conditional recovery. It tests whether the layer's unfinished baseline optimization needs materially affect finite-budget training. It requires no extra model evaluation and does not replace the Oats value C_l in the default schedule.

Cross-layer adaptation response. Let $e_i = \mathrm{vec}(\hat{W}_i - W_i)$ be a quantization perturbation and H_{li} a cross-layer Hessian block in a smooth local model, with $H_{ll} = H_l$. For an externally quantized set S,

$$\Delta g_l \simeq \sum_{i\in S} H_{li}e_i, \qquad C_l^* \simeq \tfrac{1}{2}e_S^\top H_{Sl}H_l^{-1}H_{lS}e_S. \tag{24}$$

The cross-layer terms describe how quantizing the external set changes the layer's adaptation opportunity. The own-layer term $\frac{1}{2}e_l^\top H_l e_l$, used in established quantization and fine-tuning priorities (Dong et al., 2019), supplies the sensitivity control in our experiments. Eliminating continuous compensating variables yields the same inverse-curvature structure in Schur-complement approaches (Lee et al., 2026). Oats uses shared cross-layer measurements of this structure to guide the training order.

B.3 Conditional Recovery with Unequal Curvatures

Quantizing other layers can change the target layer's curvature as well as its gradient. The next result identifies the correction to Theorem 1 when the two local quadratic models have different curvatures. This is distinct from approximating one shared curvature by $\widetilde{H}_l$ in Eq. (21).

Proposition 2 (Recovery under curvature change)**.** *For one fixed layer, let the reference and perturbed losses have quadratic models in the same continuous update coordinates u:*

$$\begin{aligned} q_0(u) &= c_0 + g_0^\top u + \tfrac{1}{2}u^\top H_0 u, \\ q_1(u) &= c_1 + (g_0 + \Delta g)^\top u + \tfrac{1}{2}u^\top H_1 u, \end{aligned} \tag{25}$$

where H_0, H_1 are symmetric positive-definite matrices and c_0, c_1 are constants. At the reference optimum $u_0^* = -H_0^{-1}g_0$, the remaining gradient and recoverable loss in the perturbed model are

$$\begin{aligned} r &= \Delta g - (H_1 - H_0)H_0^{-1}g_0, \\ C_{\neq} &\equiv q_1(u_0^*) - \min_u q_1(u) = \tfrac{1}{2}r^\top H_1^{-1} r. \end{aligned} \tag{26}$$

If $\|H_0^{-1/2}(H_1 - H_0)H_0^{-1/2}\|_2 \le \delta < 1$, define $C_0 = \frac{1}{2}\Delta g^\top H_0^{-1}\Delta g$ and $B_0 = \frac{1}{2}g_0^\top H_0^{-1}g_0$. Then

$$\frac{(\sqrt{C_0} - \delta\sqrt{B_0})_+^2}{1+\delta} \le C_{\neq} \le \frac{(\sqrt{C_0} + \delta\sqrt{B_0})^2}{1-\delta}, \tag{27}$$

where $(x)_+ = \max(x, 0)$.

Proof. Substituting u_0^* into $\nabla q_1(u) = g_0 + \Delta g + H_1 u$ gives r in Eq. (26). The perturbed optimum is $u_1^* = -H_1^{-1}(g_0 + \Delta g)$, so $r = H_1(u_0^* - u_1^*)$. Completing the square in q_1 therefore gives $C_{\neq} = \frac{1}{2} r^\top H_1^{-1} r$.

For the bound, write $E = H_0^{-1/2}(H_1 - H_0)H_0^{-1/2}$, $v = H_0^{-1/2}\Delta g$, and $h = H_0^{-1/2} g_0$. The recovery becomes

$$C_{\neq} = \tfrac{1}{2}(v - Eh)^\top (I + E)^{-1}(v - Eh). \tag{28}$$

Here I is the identity matrix. Since E is symmetric with $\|E\|_2 \leq \delta < 1$, the eigenvalues of $(I + E)^{-1}$ lie between $(1 + \delta)^{-1}$ and $(1 - \delta)^{-1}$. The triangle and reverse triangle inequalities give

$$(\|v\|_2 - \delta\|h\|_2)_+ \leq \|v - Eh\|_2 \leq \|v\|_2 + \delta\|h\|_2. \tag{29}$$

Combining these inequalities with $\|v\|_2^2 = 2C_0$ and $\|h\|_2^2 = 2B_0$ proves Eq. (27). □

The correction couples curvature change with unfinished reference adaptation: $B_0 = q_0(0) - q_0(u_0^*)$ is the reference loss still recoverable at the measurement point. If that point is already reference-optimal, $g_0 = 0$ and the correction vanishes, giving $C_{\neq} = \frac{1}{2}\Delta g^\top H_1^{-1}\Delta g$ even when the curvatures differ. If the curvatures agree, the original recovery identity is recovered. Thus simply replacing H_l by H_1 in the original formula is generally insufficient away from reference optimality.

These are exact statements about positive-definite quadratics. Their use as local approximations also requires the relevant updates to stay within a region where the models are accurate. Neither the curvature-change condition above nor the Fisher-approximation condition in Eq. (21) is guaranteed for nonconvex QAT with STE and AdamW. The diagnostic in Tab. 14 separates these two sources of approximation; it does not add curvature estimation or trial training to the default algorithm.

B.4 What Complementary Partitions Estimate

For an even number L of layers, draw a uniformly random permutation and place its first $L/2$ layers in A, the remainder in B. Draw a new independent permutation for every partition. Conditional on receiver l, its opposite set S_l is a uniformly chosen subset of $m = L/2$ among $n = L - 1$ other layers. The nonlinear score targets the expectation of the finite external intervention under this distribution. Averaging K partitions estimates that expectation conditional on the fixed model, data, and curvature estimate, rather than recovering every pair's interaction.

Proposition 1 describes a common additive approximation to these responses. All source vectors and the curvature matrix are fixed before drawing a partition. A smooth local model gives $v_i = H_{li} e_i$ by linearizing at one shared quantized reference; this need not equal the nonlinear response across every mixed state.

Proof of Proposition 1. For $L > 2$, let I_i indicate membership in S_l. Uniform balanced sampling gives

$$\mathbb{E} I_i = p = \frac{m}{n}, \qquad \mathbb{E}[I_i I_k] = q = \frac{m(m-1)}{n(n-1)} \quad (i \neq k). \tag{30}$$

For $n > 1$, expanding the squared norm gives

$$\mathbb{E}\left[\tfrac{1}{2}\Big\| \sum_{i \in S_l} v_i \Big\|^2_{\widetilde{H}_l^{-1}}\right] = \tfrac{1}{2}\left[(p - q)\sum_i \|v_i\|^2_{\widetilde{H}_l^{-1}} + q\Big\| \sum_i v_i \Big\|^2_{\widetilde{H}_l^{-1}}\right]. \tag{31}$$

For even $L > 2$, $p - q = q = L/[4(L-1)]$, yielding Eq. (11). For $L = 2$, the sole other layer is always external; both sides equal $\frac{1}{2}\|v_i\|^2_{\widetilde{H}_l^{-1}}$. □

For $L = 48$, individual interaction energy and the energy of the combined response each have coefficient $6/47$. For odd L, use groups of sizes $\lfloor L/2 \rfloor$ and $\lceil L/2 \rceil$; conditional on the receiver's group, Eq. (31) applies with the corresponding opposite-set size m.

A single partition excludes all same-set sources for its receivers. For 48 layers, a specified pair stays in the same set with probability $23/47$ per partition, or $(23/47)^K$ across independent partitions. More data within one partition cannot remove this omission. We therefore distinguish data-subset stability from partition stability. Additional partitions reduce this source of Monte Carlo variation; they do not make a finite K score a complete interaction matrix.

B.5 Model Fisher from Streamed Pseudo-Label Gradients

Proof of Lemma 1. At fixed all-RTN weights and fixed causal prefixes, let p_t be the vocabulary distribution and J_t the logit Jacobian with respect to the scored weight coordinates. The model Fisher for the token-mean objective is

$$\mathcal{F} = \frac{1}{T} \sum_t J_t^\top \left[\operatorname{diag}(p_t) - p_t p_t^\top \right] J_t. \tag{32}$$

For softmax cross-entropy it is also the generalized Gauss–Newton matrix of the logits under the same Jacobian. It need not equal the hard-label loss Hessian of a nonlinear network. With fake quantization, J_t is the prescribed STE Jacobian and the construction is the corresponding local surrogate curvature.

Independently sample $\tilde{y}_t \sim p_t$ without changing the supplied prefixes or differentiating the sampling distribution. Writing $e_{\tilde{y}_t}$ for the sampled one-hot vector, the score vector is $s_t = J_t^\top (p_t - e_{\tilde{y}_t})$. Conditional on the fixed prefixes,

$$\mathbb{E} s_t = 0, \qquad \mathbb{E}[s_t s_t^\top] = J_t^\top \left[\operatorname{diag}(p_t) - p_t p_t^\top \right] J_t. \tag{33}$$

For microbatch h with T_h valid targets, the implementation computes

$$r_h = \nabla_W \left[-\frac{1}{\sqrt{T_h}} \sum_{t \in h} \log p_t(\tilde{y}_t) \right], \qquad \widehat{f}_k = \sum_h \frac{T_h}{T} \, r_{h,k}^2, \tag{34}$$

where W contains all scored weights, k indexes a weight coordinate, and $T = \sum_h T_h$. We average R_F independent repetitions when more than one is used. The draws at different positions are independent, even though their fixed prefixes overlap. Thus $r_h = T_h^{-1/2} \sum_{t \in h} s_t$ satisfies

$$\mathbb{E}[r_h r_h^\top] = \frac{1}{T_h} \sum_{t \in h} \mathbb{E}[s_t s_t^\top], \qquad \mathbb{E}\left[\sum_h \frac{T_h}{T} r_{h,k}^2 \right] = \mathcal{F}_{kk}. \tag{35}$$

Restricting the first identity to layer l gives Eq. (9) for that microbatch. The second identity shows that token-weighted accumulation gives the dataset-mean Fisher diagonal. Averaging independent repetitions preserves this equality. □

Squaring each microbatch gradient before averaging removes unnecessary cross-microbatch random products. In contrast, squaring an ordinary averaged pseudo-label gradient gives the wrong normalization. Squaring real-label CE gradients estimates a different object.

In distributed execution, square each independent local microbatch gradient before any cross-rank gradient averaging. Reduce the token-weighted group statistics and token counts afterward. Microbatches with no valid targets contribute zero and do not divide by $\sqrt{T_h}$. Label masks, causal shifts, model modes, and initial parameters remain identical across curvature and real-label samples. Samples are drawn independently across repetitions and ranks with recorded seeds.

Accumulate group means directly, rather than storing a dense Fisher matrix. For the weight coordinates in quantization group $\mathcal{G}$, set $d_{\mathcal{G}} = |\mathcal{G}|^{-1} \sum_{k \in \mathcal{G}} \widehat{f}_k$. Let $\bar{d}_m$ be the mean of the group values in linear module m. The damped diagonal approximation is

$$\widetilde{H}_{kk} = d_{\mathcal{G}} + \lambda_m \quad (k \in \mathcal{G}), \qquad \lambda_m = \max(\rho \bar{d}_m, \epsilon_0). \tag{36}$$

The restriction to layer l is $\widetilde{H}_l$; only the group values need storage. The default relative damping is $\rho = 10^{-3}$ with floor $\epsilon_0 = 10^{-12}$; both are recorded configuration values. FP32 accumulation is used for gradient and curvature buffers, and final scalar energies are reduced in FP64. A group mean can still have appreciable variance when coordinates are correlated. Unbiased diagonal estimates do not imply an unbiased inverse curvature, score, or ranking.

B.6 Scope of the Global Scheduling Rule

The quadratic recovery identity isolates a receiver's opportunity to adapt to external quantization, but it does not account for every consequence of choosing that receiver's position. A layer can

also leave quantization errors that later layers must adapt to. Moreover, the initial all-RTN anchor differs from the evolving mixture of trained quantized layers and unvisited floating-point layers. The default order therefore uses a fixed, shared measurement and is assessed against forward, reverse, random, sensitivity-based, and reversed-score schedules under matched formal training budgets.

A scalar model explains one condition supporting the scheduling intuition. Suppose a training visit maps the existing error E to $T_i(E) = \rho_i E + r_i$, where $0 \le \rho_i < 1$ is its residual fraction and $r_i \ge 0$ is its own remaining error. Assume these parameters do not depend on the order. Then

$$T_l(T_i(E)) - T_i(T_l(E)) = (1-\rho_i)r_l - (1-\rho_l)r_i. \tag{37}$$

Sorting $r_i/(1-\rho_i)$ in descending order is optimal in this scalar model: every adjacent inversion can be removed without increasing the error, and any subsequent affine visits multiply the exchange difference by a nonnegative factor. If all r_i are equal, weaker adaptation first is consistent with this rule. To infer increasing C_i, the scores would additionally have to rank the fractions $1-\rho_i$ under a common error burden. Neither condition follows from Theorem 1, so this example provides intuition without guaranteeing the order selected from Oats.

The main empirical questions remain distinct: whether C_l predicts conditional receiver recovery, and whether its ascending order improves actual finite-budget QAT. A useful static schedule need not preserve identical score rankings at every training state, while rank stability alone cannot establish the best scheduling direction. We therefore use environment and drift measurements as evidence about transfer, alongside the ascending-versus-descending training comparison. The cross-term diagnostic in Eq. (23) examines unfinished baseline adaptation. Partition, damping, and sample-subset studies examine estimation stability. None of these diagnostics adds trial training or dynamic reordering to the default algorithm.

B.7 Quantizer Coordinates and Gradient Semantics

The main text presents the method using symmetric quantization. Here we specify the groupwise affine backend used by the reference implementation. The main text uses $Y = X\hat{W}$. The implementation stores weights in output-major order: $\bar{W} = W^\top \in \mathbb{R}^{O\times I}$ has O output rows and I input channels, and its dequantized counterpart is $\overline{\hat{W}} = \hat{W}^\top$. Split each row into groups of size G, assume G divides I, and define $J = I/G$ and $g(j) = \lfloor j/G \rfloor$ for zero-based column indices. The raw scale and zero-point tensors have shape $O \times J$. For each group, initialize

$$s_0 = \operatorname{clip}\left(\frac{\bar{W}_{\max} - \bar{W}_{\min}}{2^b - 1}, 10^{-4}, 10^4\right), \qquad z_0 = \operatorname{round}\left[-\operatorname{clip}\left(\frac{\bar{W}_{\min}}{s_0}, -10^4, 10^4\right)\right]. \tag{38}$$

The forward quantizer uses

$$\begin{aligned} s^{\text{eff}} &= \operatorname{clip}(s, 10^{-4}, 10^4), & z^{\text{eff}} &= \operatorname{clip}(\operatorname{round}(z), 0, 2^b - 1), \\ q_{oj} &= \operatorname{clip}\left(\operatorname{round}\left(\frac{\bar{W}_{oj}}{s^{\text{eff}}_{o,g(j)}}\right) + z^{\text{eff}}_{o,g(j)}, 0, 2^b - 1\right). \end{aligned} \tag{39}$$

The stored dequantized weights are $\overline{\hat{W}}_{oj} = (q_{oj} - z^{\text{eff}}_{o,g(j)}) s^{\text{eff}}_{o,g(j)}$. All target quantizers are initialized once from the initial model and reused unchanged during scoring. Each layer starts formal QAT from its corresponding initialized quantizer when it is visited.

Formal stage-one training uses latent weights, raw s, z, and the configured normalization weights. The scoring registry is narrower: it contains only the latent weights of the target quantized linear modules. Scales, zero points, biases, and normalization parameters are frozen during every sample, and each shared tensor is registered once. Scales and zero points share the quantization learning rate; latent and normalization weights use the weight learning rate. Biases and zero-rate groups remain frozen. Rounding and effective-parameter clamps use identity STEs; the final integer-range clamp keeps its ordinary derivative m. Writing $u = \bar{W}/s^{\text{eff}}$ and applying m at $\operatorname{round}(u) + z^{\text{eff}}$ gives

$$\frac{\partial \overline{\hat{W}}}{\partial \bar{W}} = m, \qquad \frac{\partial \overline{\hat{W}}}{\partial s} = (q - z^{\text{eff}}) - m\frac{\bar{W}}{s^{\text{eff}}}, \qquad \frac{\partial \overline{\hat{W}}}{\partial z} = s^{\text{eff}}(m-1). \tag{40}$$

Scale and zero-point contributions sum over each group. Unsaturated zero-point contributions cancel; saturation can create nonzero gradients.

For scoring, the weight backward is the fixed diagonal surrogate $\partial\overline{\hat{W}}/\partial\bar{W} = m$. It is the same for a receiver in both of its measurement states, because its own weights and quantizer do not change. The exact derivative of rounding is zero almost everywhere, whereas the STE supplies this surrogate derivative. Theorem 1 concerns the common local quadratic models, and the Fisher identity above concerns the prescribed surrogate Jacobian; neither asserts that the discrete quantizer has these true derivatives. Formal QAT can cross quantization boundaries and train scales and normalization parameters, so its attainable finite-budget recovery need not equal the weight-subspace prediction.

B.8 Full-Suffix Differentiation and State-Safe Execution

No optimizer is constructed during scoring. Parameters, initialized quantizers, observer buffers, and model modes are fixed within each evaluation, including activation-checkpoint recomputation. Disable dropout and automatic observer updates. Complete backward before changing a quantization mask. Measurement switches only the selected forward modes; no weight update must be undone. After scoring, restore every unvisited layer to its initial floating-point mode before formal training begins.

In $\mathcal{W}_{A\cup B}$, request all scored weight gradients. In $\mathcal{W}_A$, request only receiver gradients in A, and in $\mathcal{W}_B$, only those in B. Freezing source parameters saves their parameter-gradient computation but does not detach their input-gradient paths. The suffix after any requested receiver must remain differentiable. A prefix can be detached only if it lies before every requested receiver; the sample generally requests receivers across the model. For a formal visit, only the current layer's full training registry is enabled, with previous visits quantized and future visits floating point.

For summed real-label microbatch losses $\mathcal{L}_h^{\mathrm{sum}}$ and target counts T_h, accumulate each state's gradient as

$$\bar{g} = \frac{\sum_a \sum_{h\in\mathcal{P}^{(a)}} \nabla\mathcal{L}_h^{\mathrm{sum}}}{\sum_a \sum_{h\in\mathcal{P}^{(a)}} T_h}, \tag{41}$$

where a indexes ranks. Globally sum gradients and valid-token counts before taking differences and quadratic energies. Averaging independently normalized rank gradients or per-microbatch energies changes the score. Ranks with no valid targets participate with zero contributions. This order of operations differs deliberately from the Fisher accumulation in Eq. (35).

The reference implementation retains g^Q in a CPU buffer and processes one mixed state at a time, using a separate receiver buffer for its token-mean gradients. Once a state is complete, compute its receivers' score contributions and optional cross-term diagnostics, then release that working buffer. For $K > 1$, reuse the unchanged g^Q for every partition. For $K = 1$ with the cross-term diagnostic disabled, a lower-storage variant subtracts token-normalized mixed-state gradients directly from the corresponding entries of the g^Q buffer. After both complementary states, this buffer contains every $\Delta\tilde{g}_l$. It cannot subsequently supply the lost reference vector for additional partitions or the optional cross-term diagnostic without other storage or recomputation.

Gradient hooks may stream completed parameter gradients to CPU, but they must preserve the token normalization and account for shared tensors and checkpoint recomputation. Gradient compression or row sampling is not part of the default score. No Adam moments are needed for measurement; CPU offload does not remove activation memory, communication, or checkpoint costs. The reference uses BF16 compute with FP32 master weights and accumulation, unscales any loss-scaled gradients before scoring, and checks all scores for finiteness. Stable sorting uses raw FP64 scores in ascending order and original depth for exact ties, without a tunable tie threshold.

Formal stage-one training uses a fresh optimizer per layer and the same per-layer data stream, parameter registry, and budget across order baselines. A frozen language-model head still transmits hidden-state gradients. In stage two, normalization and communication precede clipping of the complete scale-gradient vector and the optimizer update. Fused cross-entropy preserves the exact label masks and reduction; approximate gradient filtering is disabled for the reference computation.

B.9 ScaleKernel: Derivation and Tiling of the Scale Gradient

Write each dequantized weight as a fixed coefficient c_{oj} times its group scale. The symmetric formulation in the main text uses $c = Q(\bar{W})$; the affine backend in Appendix B.7 uses $c_{oj} =$

Algorithm 2 ScaleKernel: direct scale-gradient reduction with fixed quantized weights

Require: Compute tensors X, dY, stored dequantized weights $\overline{\hat{W}}$, quantized weights q, zero points z, group size G, token partitions $\{\mathcal{T}_a\}_{a=1}^{R}$

for each output-row tile $\mathcal{O}$ and complete group tile $\mathcal{G}$ **do**
  $\mathcal{J} \leftarrow \{j : g(j) \in \mathcal{G}\}$
  $C \leftarrow \mathrm{unpack}(q, z; \mathcal{O}, \mathcal{J})$
  for $a = 1, \ldots, R$ **do**
    $A \leftarrow 0$ in FP32 accumulator storage
    for token tile $\mathcal{T} \subseteq \mathcal{T}_a$ **do**
      $A \leftarrow A + dY_{\mathcal{T},\mathcal{O}}^{\top} X_{\mathcal{T},\mathcal{J}}$
    end for
    $P_{a,\mathcal{O},\mathcal{G}} \leftarrow \mathrm{groupsum}(A \odot C)$
  end for
  $ds_{\mathcal{O},\mathcal{G}} \leftarrow \sum_{a=1}^{R} P_{a,\mathcal{O},\mathcal{G}}$
end for
Compute the complete input gradient $dX = dY\overline{\hat{W}}$ if requested

$q_{oj} - z_{o,g(j)}$. With all coefficients and non-scale parameters fixed, stage two solves

$$\min_{\{s_l\}} \mathcal{L}_{\mathcal{D}_{\text{train}}}\left(\{\hat{W}_l(s_l)\}\right), \qquad \overline{\hat{W}}_{oj}(s) = c_{oj} s_{o,g(j)}. \tag{42}$$

For one linear layer $Y = X\hat{W} = X\overline{\hat{W}}^{\top}$, write $dY = \partial\mathcal{L}/\partial Y$ for the arriving output gradient. The differential is

$$\begin{aligned} \mathrm{d}\mathcal{L} &= \sum_{t,o,j} dY_{to}\left(\overline{\hat{W}}_{oj}\,\mathrm{d}X_{tj} + X_{tj} c_{oj}\,\mathrm{d}s_{o,g(j)}\right) \\ &= \sum_{t,j} (dY\overline{\hat{W}})_{tj}\,\mathrm{d}X_{tj} + \sum_{o,g}\left[\sum_{j:g(j)=g} c_{oj} \sum_{t} dY_{to} X_{tj}\right]\mathrm{d}s_{og}. \end{aligned} \tag{43}$$

The coefficient of $\mathrm{d}s_{og}$ is the scale gradient for row o and group g. It is the inner product of the fixed values c_{oj} and the corresponding entries of $dY^{\top}X$. In the symmetric, single-group notation of the main text, $c = Q(\bar{W})$; transposing the storage layout gives $ds = \langle Q(W), X^{\top}dY\rangle$ and $dX = dY\hat{W}^{\top}$, as in Eq. (12). The arriving gradient dY already includes loss masks and normalization, so backward does not divide by token count again.

For output rows $\mathcal{O}$ and columns $\mathcal{J}$ spanning complete groups, accumulate

$$A_{\mathcal{O},\mathcal{J}} = \sum_{\mathcal{T}} dY_{\mathcal{T},\mathcal{O}}^{\top} X_{\mathcal{T},\mathcal{J}}. \tag{44}$$

Multiply by the corresponding c and reduce within groups before writing to global memory. Algorithm 2 implements this for the affine backend: unpack loads the selected quantized weights and subtracts their zero points using signed arithmetic; $\mathrm{groupsum}$ adds columns within each quantization group.

For $P = OI$ weights, $R = 1$ writes ds directly. Splitting tokens into $R > 1$ partitions uses RP/G FP32 partial values, followed by a fixed-order reduction. The dense P-element intermediate is absent, but its matrix-multiplication arithmetic remains. Tiles cover complete groups so that every weight contribution reaches exactly one scale accumulator; partial token and channel tiles require validity masks. Atomic accumulation is an alternative with a different summation order. Scale updates occur after backward, invalidating any cached dequantized weights.

ScaleKernel uses the same compute inputs and forward weight values as the reference. Input casts are part of this contract: the source tensor's storage type alone does not determine its matrix-multiplication precision. FP32 group reduction can differ from an eager BF16 path that rounds each completed $d\overline{\hat{W}}$ entry before scale reduction. Matching that rounding requires combining all token

partials before rounding each entry; reducing groups first generally changes the result. We therefore distinguish real-valued derivative equality from training-precision agreement, using a high-precision reference for the former and the specified eager training path for the latter.

B.10 Quantized-Weight Extraction and Packing

At each block endpoint, extract $q, s^{\text{eff}}, z^{\text{eff}}$ using Eq. (39) and the final forward precision. Stage two freezes q, z and all non-scale parameters, initializes scales from s^{eff}, and optimizes them directly without first-stage clamps or latent-weight re-rounding. Canonical shapes are $q \in \{0, \dots, 2^b - 1\}^{O \times I}$, $z \in \{0, \dots, 2^b - 1\}^{O \times J}$, and $s \in \mathbb{R}^{O \times J}$, with $J = I/G$. Compute $q - z$ after signed promotion.

The adopted dequantizer accepts finite zero or negative scales; this is a backend property, not a condition imposed on every INT implementation. Scale-layout views share one parameter. Storage conversion must keep scales and reconstructed weights finite.

For INT2/INT4, let $p = 32/b$. Packed weights have shape $\lceil I/p \rceil \times O$, zero points $J \times \lceil O/p \rceil$, and scales $J \times O$. With zero-initialized unsigned 32-bit storage,

$$\begin{aligned} \text{qweight}_{\lfloor j/p \rfloor, o} \mid &= \text{uint32}(q_{oj}) \ll b(j \bmod p), \\ \text{qzeros}_{g, \lfloor o/p \rfloor} \mid &= \text{uint32}(z_{og}) \ll b(o \bmod p). \end{aligned} \tag{45}$$

Here $\mid =$ is bitwise OR assignment and $\ll$ is left shift. Store zero points directly, leave padding bits zero, and mask signed storage when decoding. Export preserves q, z exactly with the final scales; scale-storage precision is checked separately against its corresponding reference.

B.11 Accounting for Scoring and Scale-Training Work

Let K be the number of complementary partitions, R_F the number of Fisher repetitions, and M_p the number of microbatches in one traversal of the scoring data. Excluding activation-checkpoint recomputation, the reference implementation performs

$$N_{\text{pass}} = 1 + 2K + R_F, \qquad N_{\text{forward}} = N_{\text{backward}} = N_{\text{pass}} M_p. \tag{46}$$

The first term is the common real-label gradient, the second contains the mixed states, and the third estimates curvature. These counts are shared across all L layers. Sorting the resulting scalar scores costs $O(L \log L)$ CPU work and requires no further model evaluation. The initial quantizer setup and mode restoration involve no optimizer updates.

The configured dataset and microbatch counts are given in Appendix C.1; Appendix C.8 applies this accounting to the experimental setup. Repeated traversals do not create additional independent training data, and valid target counts can be smaller than input-token counts because of label masking.

A retained computation graph can optionally share one all-RTN forward between the real-label loss and one Fisher sample, reducing the evaluation count to three forwards and four backwards for the default configuration. This is not the baseline memory plan, and checkpoint recomputation still contributes actual work. The quoted four-pass baseline uses independent complete evaluations and avoids this graph-lifetime requirement.

Each scoring backward may request many layers' parameter gradients, whereas a formal QAT visit requests only one layer's parameters and its suffix input gradients. Therefore neither the number of passes nor the ratio of token visits alone determines FLOPs or wall time. Report scoring time and peak GPU/CPU memory separately, and include scoring, checkpoint recomputation, communication, gradient caches, and both formal stages in total cost. Increasing K, R_F, or the amount of scoring data changes these measured totals.

If fusion avoids one write and read of a P-element gradient, its traffic-to-GEMM time ratio is approximately

$$\frac{t_{\text{traffic}}}{t_{\text{GEMM}}} \approx \frac{2\beta P / B_{\text{eff}}}{2 n_{\text{tok}} P / F_{\text{eff}}} = \frac{\beta F_{\text{eff}}}{n_{\text{tok}} B_{\text{eff}}}, \tag{47}$$

where n_{tok} is the number of tokens processed by the weight-gradient GEMM, β is bytes per element, and $B_{\text{eff}}, F_{\text{eff}}$ are effective bandwidth and throughput. This conditional estimate is not a

universal speedup bound: fusion changes launches, occupancy, and reductions. Peak memory depends on overlapping allocations. End-to-end accounting includes forward computation, checkpoint recomputation, communication, caches, and workspaces.

C Supplementary Experiment Details

C.1 Models, Data, and Evaluation

We use `Qwen/Qwen3.6-35B-A3B` for text generation and the base `meta-llama/Llama-3.1-8B` checkpoint. One decoder block, including its experts for Qwen, is one ordering unit. Quantization covers attention and feed-forward projections, including routed/shared expert weights. Routers, normalization, embeddings, and the output head remain floating point; the vision branch is unused. Routing follows the current model, so the source-state comparison includes any induced routing response. Baseline module support and expert-token coverage are recorded for each implementation.

We randomly sample 2K examples from `openbmb/UltraData-SFT-2605` (OpenBMB, 2026) and reuse this same subset for every stage-one layer visit and stage two. The maximum sequence length is 64k tokens; this is a training limit rather than the native length of every sampled example. One optimizer step covers microbatch size 1, gradient accumulation 8, and 64k tokens of context, so a full pass over the sampled pool is 250 steps. Each stage-one layer visit takes 44 steps and stage two takes one full pass, giving a cumulative update-token count that full-model QAT matches: 1.053819B for Qwen and 0.869270B for Llama. Native-format methods retain their own budgets. A larger full-model budget is a separate cost–quality point, not substituted for the matched-budget endpoint. Actual unique and processed token counts are determined after tokenization, excluding padding; the sample count and length limit alone do not establish a 0.13B-token corpus, and the per-step token count is a padded capacity rather than a count of real tokens in every sequence. The default scoring subset contains 64 of the sampled examples. We sample without replacement using seed 42. The preprocessing manifest stores the resolved dataset revision, the 2,000 selected row identifiers, tokenizer revisions, and valid-token counts. Examples longer than 65,536 tokens are truncated, while shorter examples retain their original length; padding is excluded from token counts. Dataset and checkpoint hashes refer to the resolved inputs recorded at execution time.

Within each model, all methods use the same tokenizer, task definitions, truncation, and output limits. Qwen uses its pinned chat template and reasoning configuration; Llama base uses a completion prompt. AIME25 and GPQA use the recorded task protocols; the GPQA subset is Diamond. LM Eval averages exactly ARC-C, ARC-E, HellaSwag, PIQA, and WinoGrande, each on the 0–100 scale. Scores are not normalized to BF16 or averaged with unrelated tasks.

C.2 Baseline Scope and Complete Quality

RTN and GPTQ round weights without gradient training; GPTQ additionally uses calibration data for its error-compensating update. QuIP#, QTIP, and LC-QAT retain their native quantizers and recipes. Model support, excluded modules, effective storage, and adaptation data are checked per model and bitwidth; unsupported results are not replaced with zero. Controlled reconstruction and CE comparisons share the affine quantizer and update scope. Full-model QAT is reported separately, including the cost of all trainable weights. The 2bit main plot includes full-model QAT; the 4bit table in Tab. 1 includes the complete baseline set supported at that precision on Qwen3.6-35B-A3B. Llama-3.1-8B is reported at 2bit only, since its 4bit runs were not completed. These tables retain all baseline rows. Where a task was not run for a row, the entry is a dash rather than a zero, and the LM Eval average is withheld for any row missing one of its five tasks. Source data contain no training-seed Std for the main quality endpoints, so no uncertainty is invented.

The Qwen LC-QAT 2bit ARC-C and ARC-E values were originally recorded with the column assignments reversed: the logged ARC-C score (77.15) exceeded the logged ARC-E score (58.11), reversing the ordering seen for every other row. The source logs confirmed a labelling error; the values have been corrected to ARC-C 58.11 and ARC-E 77.15. Both corrected values land on their respective 1/N attainable grids and satisfy ARC-C $<$ ARC-E. Their sum and the LM Eval average are unaffected by the interchange.

Setting	Qwen3.6	Llama 3.1
Parameters	35B / 3B active	8B
Decoder blocks	40	32
Architecture	Hybrid MoE	Dense
Variant	Post-trained	Base
Max. sequence length	64k	64k
Weight bits	2 / 4	2
Group size	128	128
Scale dtype	BF16	BF16

Table 4: Model scope and quantization settings. Qwen parameters distinguish total and activated size.

Model	Data (B)	Train tokens (B)
Qwen	0.131	1.054
Llama	0.131	0.869

Table 5: Unique data and cumulative update tokens, in billions, for controlled QAT. RTN uses no gradient-training tokens; scoring is separate.

Method	Bits	LCB	BFCL	LM avg.
BF16	16	81.42	75.99	76.50
RTN	4	77.65	71.65	73.22
Full QAT	4	77.63	73.95	74.32
QuIP#	4	80.40	76.12	75.72
QTIP	4	81.18	74.46	74.74
Eff. QAT	4	78.56	72.66	75.14
CE-QAT	4	79.21	74.24	74.73
Stout	4	81.53	77.26	75.97
GPTQ	4	79.31	73.95	74.24
RTN	2	20.38	53.91	69.73
GPTQ	2	20.38	55.87	70.67
Full QAT	2	21.50	58.38	69.15
QuIP#	2	24.54	58.26	71.65
QTIP	2	26.45	61.77	71.31
LC-QAT	2	74.98	71.66	74.17
Eff. QAT	2	65.96	67.15	71.98
CE-QAT	2	73.46	67.79	72.66
Stout	2	75.39	72.97	74.45

Table 6: Qwen3.6-35B-A3B, with LM Eval averaged over five tasks.

Method	Bits	LCB	BFCL	LM avg.
BF16	16	15.92	30.21	74.26
RTN	2	0.00	0.00	35.71
GPTQ	2	0.00	0.00	—
Full QAT	2	0.20	0.35	38.46
QuIP#	2	1.16	2.60	53.71
QTIP	2	1.96	4.63	55.72
LC-QAT	2	7.59	18.04	57.47
Eff. QAT	2	3.95	10.59	54.05
CE-QAT	2	6.66	15.20	54.26
Stout	2	9.00	20.38	60.42

Table 7: Llama-3.1-8B under the same reporting convention.

C.3 Where the Ordering Advantage Does Not Hold

Stout leads on LCB, BFCL, and the LM Eval average for both models at their evaluated bitwidths, but not on every individual task. Across the three model/bitwidth blocks (Qwen 2bit, Qwen 4bit, Llama 2bit) it is ahead on 22 of 33 task columns and behind on 11. We list the exceptions rather than aggregate them away, because their distribution is informative about when ordering matters.

The exceptions concentrate on tasks where quantization leaves little residual error for a training order to redistribute. The five LM Eval tasks (ARC-C, ARC-E, HellaSwag, PIQA, WinoGrande) and instruction-following metrics (IFBench, IFEval) together account for most of the eleven losses: at 4bit, every method stays within a few points of the full-precision reference, and at 2bit these particular tasks show smaller degradation than generation benchmarks. AIME25 on Llama-3.1-8B at 2bit is the opposite extreme—every method scores under one point, so the ordering is measuring noise rather than recovery. The largest single gap is GPQA on Qwen3.6-35B-A3B at 2bit, where Stout trails LC-QAT; none of the losses fall on the tasks the paper makes quantitative claims about (LCB, BFCL, LM Eval average).

Where Stout is behind, the leading method is always another trained or native-format baseline—CE-QAT, LC-QAT, QuIP#, QTIP, or EfficientQAT—never a training-free method. This is consistent with the proposed mechanism: training order can only help when there is enough quantization error

2bit / BF16	ARC-C	ARC-E	Hella.	PIQA	Wino.
BF16	62.4	79.5	82.4	83.0	75.1
RTN	49.2	70.3	79.4	79.9	69.8
GPTQ	52.8	71.5	79.2	80.2	69.8
Full QAT	50.4	70.3	78.2	79.9	66.8
QuIP#	56.7	71.4	77.2	80.2	72.8
QTIP	51.7	71.7	79.4	81.1	72.7
LC-QAT	58.1	77.2	79.2	81.5	74.9
Eff. QAT	56.3	72.5	79.1	78.5	73.6
CE-QAT	54.6	75.9	78.8	79.8	74.2
Stout	58.9	77.1	80.0	82.0	74.2

Table 8: Qwen 2bit and BF16: individual LM Eval scores.

2bit / BF16	ARC-C	ARC-E	Hella.	PIQA	Wino.
BF16	55.9	81.1	79.5	81.3	73.5
RTN	26.0	26.9	25.8	50.0	49.9
GPTQ	24.4	—	26.4	51.0	50.0
Full QAT	30.6	25.9	27.1	53.5	55.1
QuIP#	37.0	52.2	51.2	68.7	59.4
QTIP	38.3	56.4	54.3	66.5	63.2
LC-QAT	39.2	52.8	60.7	72.4	62.4
Eff. QAT	35.8	52.8	51.8	70.0	60.0
CE-QAT	26.8	53.0	56.0	72.2	63.2
Stout	39.1	61.1	64.0	74.5	63.4

Table 9: Llama 2bit and BF16: individual LM Eval scores.

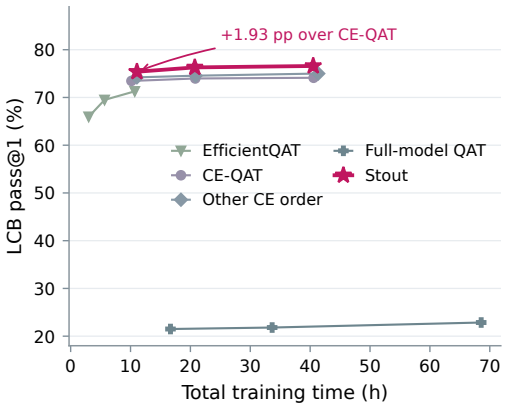

Figure 7: **Quality versus total cost.** Qwen 2bit; every point completes training. The annotation compares Stout with CE-QAT at the first shared update-token budget. The horizontal axis is total pipeline time for each schedule.

Backend	p50 (s)	p95 (s)	Stage (h)	Mem.
Eager	34.67	42.21	2.51	63.0
Compiled	24.04	26.79	1.75	42.2
Fused ref.	23.39	24.84	1.69	42.2
ScaleKernel	23.20	24.49	1.67	36.1

Table 10: **Scale training at 2bit.** Qwen, 64k context. Step latency includes p50 and p95; stage time includes compilation. Mem. is peak allocated device memory in GiB.

for adaptation to redistribute. A method that wins every column at every setting would be a less credible outcome than one with localised, explainable losses.

C.4 Ordering, Recovery, and Uncertainty

Ordering comparisons keep CE, layer budgets, quantizer, and scale backend fixed. Sensitivity and other score directions are chosen on development data. We compare forward, reverse, fixed random permutations, own-layer sensitivity, raw external response, reference gradient potential, potential difference, and ascending/descending Oats. The displayed random permutations are distinct from training-seed repetitions. The *development-selected ordering baseline* referenced in the abstract and §4.2 is the raw external response ordering, selected by minimising mean held-out CE across the competing arms; it is distinct from the reference-potential ordering shown in Tab. 11. The mean stage-one LCB scores are 72.97 for forward order and 74.92 for Oats order, measured before scale training. The corresponding final scores quantify how much of the ordering benefit remains after stage two.

Recovery uses eight preselected receivers, two external sets, and separate weight-only and full stage-one trainables. A receiver first adapts in the reference environment. After external quantization, frozen and adapting branches share parameters and fresh optimizer states; a no-extra-quantization branch measures ordinary continued training. Recovered CE is frozen minus adapted CE. We retain negative gains and never subtract an unrelated control gain to redefine theoretical recovery. Cor-

Order	Score (h)	CE	BFCL
Forward (= depth ↑)	0.00	1.735	67.85 ± 0.12
Reverse	0.00	1.724	68.56 ± 0.60
Sensitivity	0.81	1.725	69.23 ± 0.50
Reference potential	0.84	1.714	70.67 ± 0.05
Oats (depth-matched)	0.88	1.701	72.39 ± 0.13
Oats ↓	0.85	1.736	68.15 ± 0.33
Oats ↑	0.84	1.687	73.06 ± 0.11

Table 11: Qwen 2bit order statistics. Forward order (input-to-output) is identical to ascending depth and serves as the zero-cost depth baseline. BFCL reports mean ± training-seed Std.

Score	τ_S	τ_M	τ_F
Oats	0.89	0.71	0.86
Conditioned	0.71	0.54	0.68
Sensitivity	-0.11	0.00	-0.07
Raw response	0.68	0.64	0.71
Reference	0.82	0.64	0.79
Potential diff.	0.75	0.57	0.71

Table 12: Mean within-source Kendall τ_b at short, medium, and full adaptation budgets.

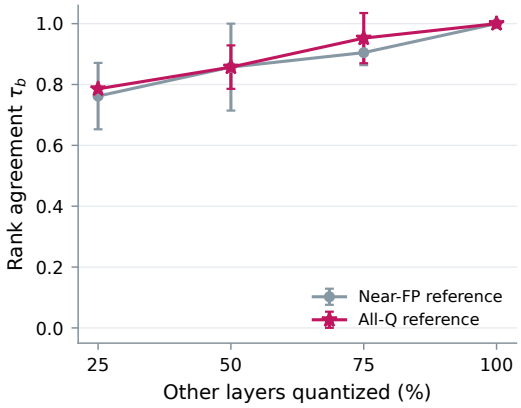

Figure 8: Matched external sets under two quantization backgrounds. Zero-source ranks are undefined.

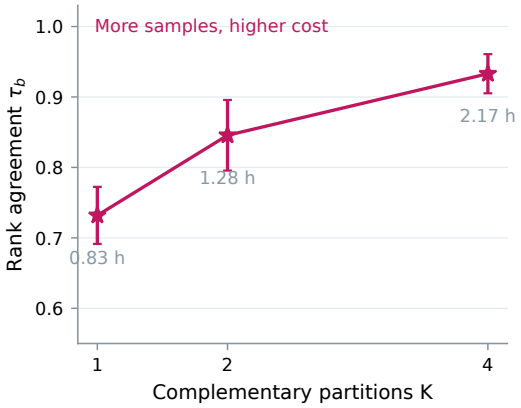

Figure 9: Partition count trades measurement cost against rank stability. Labels give scoring hours.

relations are computed within each source set and budget, then summarized, rather than pooling repeated observations across budgets. The concrete short/medium/full fractions in the current result records are 0.20, 0.55, and 1.00.

The no-extra-quantization control branch does not quantize any external layers, so its trajectory is independent of which external environment is active. Every control checkpoint therefore reports the same value across the two external sets, at every budget, by construction: the branch is a single run whose identity does not change with the source-set assignment. The adapting and frozen branches do differ across external sets, since those branches train under a specific quantized complement.

C.5 Environment and Estimator Stability

For the same receiver and source set, the bottom comparison adds quantization to a near-FP reference, while the top comparison removes the same source set from the all-Q model. The receiver remains quantized. Source fractions use $m = \mathrm{round}(f(L-1))$, with the actual fraction recorded. At zero sources the response is zero and rank correlation is undefined. At all sources the two comparisons agree. The eight-receiver audit uses the same data and metric throughout; it is not full-model rank agreement.

$$d_l^{\mathrm{bottom}}(S) = g_l(\{l\} \cup S) - g_l(\{l\}), \qquad d_l^{\mathrm{top}}(S) = g_l(\mathcal{U}) - g_l(\mathcal{U} \setminus S). \tag{48}$$

Here $\mathcal{U}$ contains every layer and $g_l(T)$ denotes the receiver gradient with quantization set T.

Partition, data, curvature-sample, and damping sweeps change one factor at a time. Only configurations with complete training endpoints receive task scores. Additional curvature samples reduce

K	Cosine	Rel. error	Nonadd.
1	0.897	0.626	0.104
4	0.940	0.540	0.115

Table 13: Source-set approximation. Mean response cosine, relative error, and nonadditivity.

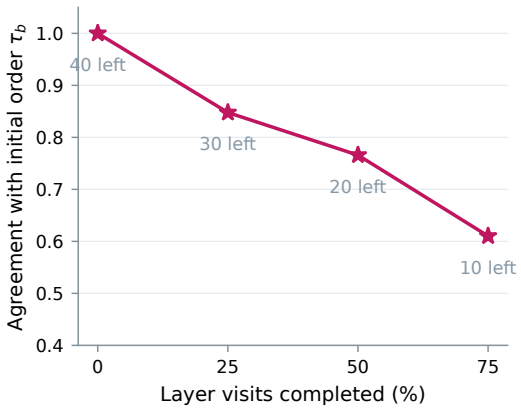

Figure 10: Initial-order agreement among the same remaining layers. Labels give set size.

Quantity	Mean
Fisher change	0.174
Group / diagonal	1.032
Score change	-0.065

Table 14: State-specific Fisher and score changes, averaged over receivers.

Order	Score (h)	LCB	BFCL
Static	0.84	76.03 ± 0.24	73.06 ± 0.25
Refresh	3.25	76.09 ± 0.28	73.08 ± 0.32

Table 15: Static versus refreshed order at equal total time. Scores report mean $\pm$ training-seed Std.

sampling variability without making the half-set score an unbiased all-other-layer score. The rank comparisons and source masks are fixed before measuring performance.

C.6 Approximation Diagnostics

The direct own-quantized reference compares all external layers against none; the deployed-state comparison adds a half-set to an already quantized complement. We therefore compare response direction, score, and receiver rank without assuming equal targets. Complement additivity is checked against an identical reference using $d(S \cup T) - d(S) - d(T)$, retaining both absolute and relative residuals. A separate FP-receiver control isolates the change of receiver mode.

Curvature diagnostics use state-specific model-Fisher diagonals, not the exact Hessian of the deployed quantizer. Score changes under a shared versus state-matched metric are compared with actual finite-budget recovery. Small projected smooth surrogates can test local Taylor errors; they do not establish global QAT convergence. Tab. 14 summarizes the available Fisher and score discrepancies.

C.7 Static Ranking and Refresh

At each checkpoint, initial and updated ranks are restricted to the same unvisited layers. Source masks retain their original pool, and curvature is reestimated on matched data. Limited refresh reorders only unvisited layers and charges every extra pass. Its equal-time comparison uses the fixed 14.50-hour budget recorded for this experiment; this is distinct from the objective-comparison budget. A small mean quality difference alone does not establish equivalence.

C.8 Training Cost and ScaleKernel

The same-update comparison includes reconstruction-target construction or full CE suffix backpropagation. Equal-time comparisons instead allocate remaining time by a development-fixed rule. Every plotted layerwise schedule finishes all layer visits and both stages. Token counts distinguish unique training data, repeated training visits, and scoring. They are not substituted for measured

Backend	p50 (s)	Stage (h)	GPU (GiB)
Eager	34.67	2.51	63.0
Compiled	24.04	1.75	42.2
Fused ref.	23.39	1.69	42.2
ScaleKernel	23.20	1.67	36.1

Table 16: Qwen 2bit scale training: step latency, stage time, and peak allocated GPU memory.

Backend	Host	Reserved	Compile (s)
Eager	22.9	67.7	0
Compiled	22.6	45.0	135
Fused ref.	21.4	44.1	80
ScaleKernel	20.5	37.9	104

Table 17: Qwen 2bit host and reserved GPU memory in GiB; cold compilation is in seconds.

Method	Score (h)	Total (h)	LCB
MSE	0.00	2.94	66.12
CE	0.00	10.45	72.50
Stout	0.81	10.93	75.95

Table 18: Qwen 2bit at matched updates. Total cost includes objective-specific work.

Backend	ds rel. error	dX rel. error
Eager	0.00000	0.00000
Compiled	0.00013	0.00008
Fused ref.	0.00019	0.00011
ScaleKernel	0.00018	0.00010

Table 19: Norm-relative gradient errors at 2bit against the prescribed numerical reference.

runtime. The main cost curve shows controlled QAT; the separate native-baseline role in the source is not relabeled as the strongest baseline without development-selection evidence.

ScaleKernel comparisons share the stage-one checkpoint, microbatch size, accumulation, attention, checkpointing, precision, and offload policy. All system measurements are reported at 2bit, the setting the efficiency claims target; Tab. 17 complements the main-text comparison with host and reserved memory. We do not report peak memory at 4bit, because the change in peak allocation there combines the packed-weight difference with allocator and workspace effects and so cannot be attributed to the kernel. Host memory is process-tree RSS; GPU allocated and reserved peaks are distinct. Latency is reported as p50 because the workbook does not yet contain steady-state means and their repetitions. No p50 is renamed as a mean. The context sweep uses 2,048, 8,192, 32,768, and 65,536 tokens on both models. Each setting uses three profiling repetitions, with 10 warmup steps followed by 100 timed optimizer steps. Matched backends use microbatch size 1 and gradient accumulation 8. On Qwen, expert dispatch and communication remain in full-stage timing; dense-GEMM speed alone is insufficient.

C.9 Separating Oats from Layer Depth

Because higher-index layers both sit deeper and tend to carry more adaptation headroom, Oats and layer depth are correlated, and the eight-receiver measurement in Appendix C.4 is too small to tell them apart on its own. This subsection reports the two comparisons that separate them: a recovery measurement stratified by depth, and a depth-sorted arm in the ordering ablation.

Depth-stratified recovery. We repeat the recovery measurement on 24 receivers instead of eight, drawn in equal numbers from a shallow, a middle, and a deep band of the network, so that within each band the layers differ in Oats while spanning almost no depth. Tab. 22 reports rank correlations with measured recovery, before and after partialling out depth.

With depth spread deliberately reduced, Oats correlates with recovery at 0.963 while depth alone reaches only 0.484. Partialling out depth leaves 0.954, so the association is not a restatement of layer index. Across these receivers Oats and depth correlate at 0.546, well below the 0.976 seen on the original eight, confirming that the earlier figure reflected how those receivers were chosen rather than a property of the score. Own-layer sensitivity remains uninformative here, as it was before.

Forward order is the depth baseline. One decoder block is one ordering unit, so training the network from input to output is exactly ascending depth. Forward order in Tab. 11 is therefore already the zero-cost depth baseline: it requires no measurement, and any comparison against it is a comparison against sorting by depth. Ascending Oats improves final LCB by 2.59 points over it.

Variant	Value	τ_b	Hours
Sequences	16	0.648	0.21
Sequences	32	0.778	0.42
Sequences	64	0.983	0.81
Sequences	128	0.972	1.73
Curvature samples	1	0.980	0.86
Curvature samples	2	0.916	1.08
Curvature samples	4	0.924	1.51
Damping	0.0001	0.826	0.85
Damping	0.001	0.973	0.85
Damping	0.01	0.777	0.85

Table 20: One-factor estimator variants. Ranks use the configured same-data reference.

K	Traversals	Layer ordering
1	4	$O(L \log L)$
2	6	$O(L \log L)$
4	10	$O(L \log L)$

Table 21: Logical scoring cost with one curvature sample. Traversals are shared across all layers; wall time also includes recomputation and communication.

Score	ρ	$\rho \mid$ depth
Oats	0.963	0.954
Layer depth	0.484	—
Sensitivity	-0.015	0.206

Table 22: Rank correlation with measured recovery on 24 depth-stratified receivers, before and after controlling for depth.

Method	LCB	BFCL	LM Eval
CE-QAT	73.47 ± 0.20	67.82 ± 0.03	72.65 ± 0.03
LC-QAT	75.04 ± 0.05	71.71 ± 0.10	74.14 ± 0.06
Stout	75.25 ± 0.16	72.92 ± 0.14	74.39 ± 0.06

Table 23: Qwen 2bit over three training seeds, mean $\pm$ Std. The LCB gap to LC-QAT is within seed noise; the BFCL gap is not.

To separate the part of that gain that comes from moving layers to different depths from the part that does not, we add a depth-matched Oats arm. We split the 40 blocks into eight consecutive bands of five, and permute layers only within a band, using the Oats order restricted to that band. Every layer therefore stays within five positions of where forward order would place it, and the schedule's depth profile is held fixed at the band level. This arm recovers 1.40 of the 2.59 point gain, so roughly half of it survives when the schedule cannot change a layer's depth band at all. The remainder is what Oats buys by moving layers across bands.

Reading the two measurements together. Depth is a useful and free first approximation to adaptation headroom, and part of what Oats provides can be had without measuring anything. What depth cannot do is distinguish layers at the same depth, and both comparisons above isolate exactly that residual.

C.10 Seed Variation and Quantizer Transfer

Training-seed repeats. The main tables report single runs. To check which of the headline gaps survive training noise, we repeat the Qwen 2bit pipeline for CE-QAT, LC-QAT, and Stout under three training seeds (Tab. 23). Against CE-QAT the gaps are 1.78 LCB and 5.10 BFCL, both far outside the seed spread ($p < 0.001$, Welch). Against LC-QAT the BFCL gap of 1.21 is likewise well separated ($p < 0.001$), but the LCB gap of 0.22 is not: with three seeds it does not reach significance ($p = 0.135$). We therefore do not claim an LCB advantage over LC-QAT; the comparison that the data supports is on BFCL and on the aggregate, and against the controlled CE baseline on both.

The ordering ablation uses the same three seed labels (0/1/2) across all policies. LiveCodeBench uses a shared evaluation RNG for sampling answers, so the same seed yields the same per-problem draws regardless of which training policy produced the checkpoint. This shared evaluation source is the correct explanation for the synchronized seed pattern visible in the ordering ablation: a within-policy permutation test across policies is testing evaluation sampling, not training behaviour.

Transfer across quantizers. Stout was developed with a group-wise affine quantizer, while the strongest published baselines use their own. An ordering gain measured only on our quantizer could

Quantizer	Forward	Oats	ΔLCB	ΔBFCL
Group-wise affine	73.62	76.08	2.46	3.98
LC-QAT codebook	75.03	76.70	1.66	2.85
QTIP trellis	26.49	28.82	2.33	2.93

Table 24: Final Qwen 2bit LCB under forward and Oats orders inside three quantizers, each compared against its own forward baseline.

be an artifact of it, so we also apply the Oats order inside two other quantizers, keeping each one's own forward-order run as its reference (Tab. 24).

The ordering gain appears in all three: it is largest on the group-wise affine quantizer, and still worth 1.66 LCB points inside the LC-QAT codebook. The gain is therefore a property of the schedule rather than of the quantizer it was developed with, although it is smaller for the stronger quantizers, which leave less residual error for a training order to redistribute.

C.11 LIMITATIONS

Depth explains part of the ordering signal. Oats correlates with layer depth, and Appendix C.9 is the measurement that separates the two. The separation holds on 24 depth-stratified receivers and in a depth-sorted ordering arm, but both are single-model, single-bitwidth measurements, and the residual association after controlling for depth rests on 24 receivers rather than on a large sample. A depth-sorted order is free and already recovers part of the benefit; we report it as a baseline rather than treating the full ordering gain as attributable to Oats.

Other limitations. Most main-table endpoints are single runs. Appendix C.10 adds three-seed repeats for the three methods the headline comparison rests on, and shows that the LCB gap to LC-QAT does not reach significance at that sample size; the remaining table cells carry no uncertainty estimate. System measurements cover Qwen3.6-35B-A3B at 2bit only, so the memory and latency claims are not established at other bitwidths or on the dense model, and Llama-3.1-8B is evaluated at 2bit only. The local analysis in Sec. 3 motivates an ascending order but does not guarantee global optimality, as Appendix B.6 discusses.