LOOPQ: QUANTIZATION FOR LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped language models (LoopLMs) increase effective computational depth by repeatedly applying shared Transformer blocks without increasing the parameter count. However, this reuse makes them particularly sensitive to post-training quantization (PTQ). We systematically study PTQ in LoopLMs and identify three sources of degradation: **distribution shift across roles**, **state reuse across loop transitions**, and **recursive error accumulation**. We propose **LoopQ**, a loop-aware PTQ framework that preserves a shared quantized backbone while introducing lightweight adaptations. LoopQ combines loop-aware activation scaling and selective transformations to accommodate changing hidden-state distributions, cross-loop state alignment to stabilize transitions, and trajectory-aware calibration to limit accumulated error. Under W4A4 quantization, LoopQ achieves a 67.8% average relative improvement in five-task mean accuracy across four backbones and a conservative estimate of 75.2% for the average relative perplexity reduction across eight backbone–dataset pairs, compared with the strongest static PTQ baseline for each comparison.

1 INTRODUCTION

Looped language models (LoopLMs) (Dehghani et al., 2018; Lan et al., 2019; Zhu et al., 2025; Jeddi et al., 2026; Bae et al., 2025; Fan et al., 2024) increase effective computational depth by **recursively reusing shared Transformer blocks** without increasing the parameter count. Their compact parameterization makes them promising for memory- and bandwidth-constrained deployment (Zhou et al., 2022; Laguna et al., 2022; Kwon et al., 2026; Yu et al., 2024). Post-training quantization (PTQ) could further reduce the storage and data movement required for weights and activations (Alizadeh et al., 2020; Shkolnik et al., 2020; Xiao et al., 2023b). Yet LoopLMs have largely been evaluated in full precision, while existing PTQ methods were developed primarily for conventional LLMs (Xiao et al., 2023a; Ashkboos et al., 2024; Lin et al., 2024; Kim et al., 2026; Hu et al., 2025; Sun et al., 2025; Sanjeet et al., 2026). Whether these methods remain effective under recursive block reuse is unclear.

Figure 1 illustrates the quantization challenge in LoopLMs. Quantization reduces storage but can sharply degrade accuracy, while hidden-state errors grow across loops and spike near loop transitions. These observations motivate quantization methods that account for recursive state reuse. The underlying challenge is that the same physical block serves different computational roles across loops and processes changing hidden-state distributions (Zeitoun et al., 2026). We distinguish three related sources of error. **(1) Distribution Shift across Roles:** shared quantization scales and transformations may be poorly suited to the activation magnitudes and geometry of individual loops. **(2) State Reuse across Transitions:** quantized outputs feed subsequent loops, carrying state mismatch across boundaries. **(3) Recursive Error Accumulation:** errors introduced in earlier loops propagate through subsequent computations and can be amplified. Section 3 analyzes how shared quantization incurs mismatch and how loop-wise errors propagate along the recurrent trajectory.

Making every quantization transformation loop-specific could reduce such mismatch, but would increase storage and weaken the parameter-sharing advantage of LoopLMs. We therefore propose **LoopQ**, which retains a shared quantized backbone and introduces loop-dependent adaptation where it is most useful. As shown in the right panel of Figure 1, **Loop-aware Activation Scaling** adjusts activation scales across loops to address magnitude drift. **Selective Loop-aware Transformation** uses a *sharing-sensitivity score* to select transformation groups that benefit from loop-specific treatment,

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.

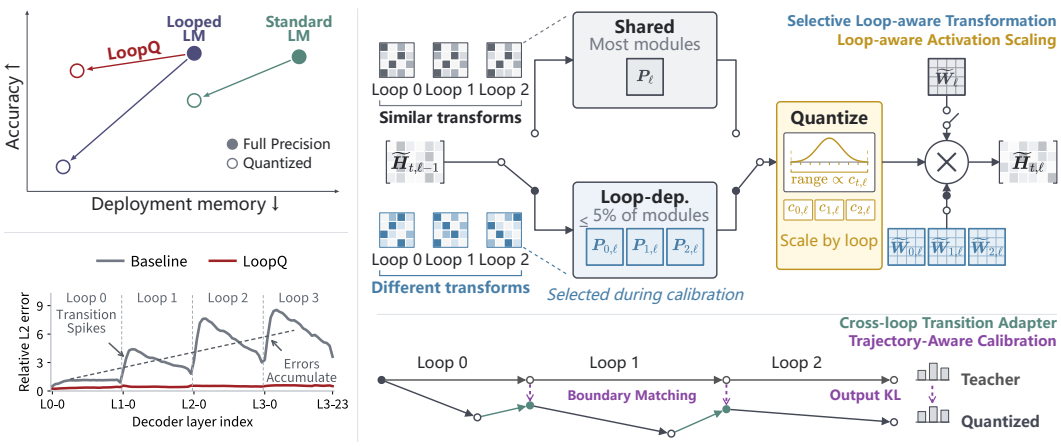


Figure 1: **Left:** A schematic accuracy–storage comparison of looped and standard, non-recursive LMs before and after quantization (top), and relative L_2 error of Ouro 1.4B across four loops for the PTQ baseline and LoopQ (bottom). **Right:** LoopQ uses loop-aware activation scales and restricts loop-dependent transformations to selected modules, with the transforms folded into stored weights (upper). The CTA adjusts hidden states at loop boundaries, while TAC aligns post-adapter states with full-precision boundary states and final predictions through output KL (lower).

addressing residual geometry mismatch while limiting storage overhead. **Cross-loop Transition Adapter** corrects hidden states at loop transitions with a lightweight adapter. Finally, **Trajectory-Aware Calibration** jointly calibrates these components along the recurrent trajectory, balancing alignment with full-precision boundary states against agreement in final predictions.

Our contributions are threefold. **(i)** We analyze quantization under recursive block reuse, characterizing the cost of sharing a quantization configuration across loops and the propagation of state error through subsequent loops. **(ii)** We develop LoopQ, which combines loop-dependent scaling and selective transformations with boundary-state correction and trajectory-aware calibration while retaining a shared quantized backbone. **(iii)** LoopQ achieves a 67.8% average relative gain in five-task mean accuracy, a conservatively estimated average relative perplexity reduction of 75.2%, and up to $2.13\times$ BF16 decode throughput on long prompts.

2 Problem Formulation

We first define the recursive computation in a LoopLM, then describe PTQ with quantization parameters shared across loops.

LoopLM. A LoopLM applies a shared stack of L Transformer layers for T loops. Let $\boldsymbol{x}$ be an input sequence of n tokens, and let $f_\ell(\cdot;\boldsymbol{\theta}_\ell)$ denote layer ℓ. The hidden state at layer ℓ of loop t is $\boldsymbol{H}_{t,\ell} \in \mathbb{R}^{n\times d}$, where d is the hidden dimension. We consider one input sequence and omit the batch dimension. Starting from $\boldsymbol{H}_{0,0} = \mathrm{Embed}(\boldsymbol{x})$, the model computes

$$\boldsymbol{H}_{t,\ell} = f_\ell(\boldsymbol{H}_{t,\ell-1};\boldsymbol{\theta}_\ell), \qquad \boldsymbol{H}_{t+1,0} = \boldsymbol{H}_{t,L}, \tag{1}$$

for $t \in \{0,\dots,T-1\}$ and $\ell \in \{1,\dots,L\}$. Writing $\boldsymbol{H}_t := \boldsymbol{H}_{t,0}$ and $F := f_L \circ \cdots \circ f_1$, we obtain the recurrence $\boldsymbol{H}_{t+1} = F(\boldsymbol{H}_t)$. After T loops, the output logits are $\boldsymbol{Z} = \mathrm{Proj}(\boldsymbol{H}_T) \in \mathbb{R}^{n\times v}$, where v is the vocabulary size.

For the quantization analysis, we represent a layer by a linear projection with weight matrix $\boldsymbol{W}_\ell$, so that $\boldsymbol{H}_{t,\ell} = \boldsymbol{H}_{t,\ell-1}\boldsymbol{W}_\ell^\top$. This abstraction isolates the effect of reusing a quantized mapping; the LoopLM itself consists of full Transformer layers.

Post-training Quantization on LoopLMs. We begin with a conventional PTQ configuration that applies the same quantization parameters to each shared layer on every loop. Direct low-bit quantization can incur substantial error when weights or activations have heavy tails or large differences in scale across channels. Transformation-based PTQ addresses this by applying an invertible transformation $\boldsymbol{P}_\ell$ to the activations and the corresponding inverse transformation to the

weights. The full-precision mapping is unchanged:

$$\boldsymbol{H}_{t,\ell} = (\boldsymbol{H}_{t,\ell-1}\boldsymbol{P}_\ell)(\boldsymbol{W}_\ell \boldsymbol{P}_\ell^{-\top})^\top. \tag{2}$$

With a shared transformation $\boldsymbol{P}_\ell$ and activation quantization step size c_ℓ, quantized inference takes the form

$$\widetilde{\boldsymbol{H}}_{t,\ell} = Q_a(\widetilde{\boldsymbol{H}}_{t,\ell-1}\boldsymbol{P}_\ell; c_\ell)\, Q_w(\boldsymbol{W}_\ell \boldsymbol{P}_\ell^{-\top})^\top, \qquad \widetilde{\boldsymbol{H}}_{t+1,0} = \widetilde{\boldsymbol{H}}_{t,L}. \tag{3}$$

Here, Q_w and Q_a denote weight and activation quantization. For a scalar activation x, symmetric uniform quantization is

$$Q_a(x; c_\ell) = c_\ell \cdot \mathrm{clip}(\lfloor x/c_\ell \rceil, q_{\min}, q_{\max}), \tag{4}$$

where $c_\ell > 0$ sets the quantization step size, $\lfloor \cdot \rceil$ denotes rounding to the nearest integer, and $q_{\min}$ and $q_{\max}$ are the representable integer bounds. We suppress activation-group indices for clarity.

Equation (3) exposes two coupled difficulties specific to recursive reuse. First, each layer uses the same $\boldsymbol{P}_\ell$ and c_ℓ across loops even though its activation distribution changes. Second, the quantized output of one loop becomes the input to the next, allowing local quantization errors to alter subsequent states and accumulate along the trajectory. We analyze these effects in the next section.

3 Where Quantization Error Arises in LoopLMs

We analyze two coupled effects of quantizing a recursively reused Transformer stack: mismatch between shared quantization parameters and loop-dependent activations, and propagation of error through reused hidden states. Proofs are provided in Appendix A.

Distribution Shift across Roles. At a given layer, all loops share the same quantization configuration, but the layer receives different hidden-state distributions as recursion proceeds. A shared scale c_ℓ must balance resolution for small activations against clipping of large ones. Changes in dominant directions and channel-wise energy can also make different loops favor different transformations, leaving a shared $\boldsymbol{P}_\ell$ suboptimal for some loops.

Proposition 1. *For $\boldsymbol{x} \sim \mathcal{D}$, let $\boldsymbol{H}_{t,\ell-1}(\boldsymbol{x})$ be the hidden state on the fixed full-precision trajectory, and define $\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P}) := \boldsymbol{H}_{t,\ell-1}(\boldsymbol{x})\boldsymbol{P}$. With Q_a from Eq. (4) applied elementwise, the expected transformed-activation quantization error is*

$$\varepsilon_{t,\ell}(\boldsymbol{P}, c) := \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\|Q_a(\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P}); c) - \boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P})\|_F^2\right].$$

Assume that all minima below are finite and attained over the same admissible set of invertible $\boldsymbol{P}$ and $c > 0$. Let $\varepsilon^*_{t,\ell} := \min_{\boldsymbol{P},c} \varepsilon_{t,\ell}(\boldsymbol{P}, c)$ and define the cost of sharing one configuration across loops as

$$\Delta_\ell^{\mathrm{share}} := \min_{\boldsymbol{P}_\ell, c_\ell} \sum_{t=0}^{T-1} \varepsilon_{t,\ell}(\boldsymbol{P}_\ell, c_\ell) - \sum_{t=0}^{T-1} \varepsilon^*_{t,\ell}.$$

Then $\Delta_\ell^{\mathrm{share}} \geq 0$, with equality if and only if a single configuration is optimal for every loop. In particular, if any two loops have disjoint sets of optimal configurations, then $\Delta_\ell^{\mathrm{share}} > 0$: every shared configuration incurs excess transformed-activation quantization error in at least one loop.

Proposition 1 compares the best shared configuration with loop-specific optima. The excess activation error vanishes when all loops admit a common optimum and is positive when their optima are incompatible. The accompanying analysis gives a sufficient condition under which magnitude drift requires different scales, and a local quadratic estimate of the cost of sharing transformations preferred by different loops. This distinction motivates loop-dependent activation scaling and selective relaxation of transformation sharing (Sec. 4.2).

Error Propagation across Loops. Quantization also changes the state passed between loops. Recall the full-precision recurrence $\boldsymbol{H}_{t+1} = F(\boldsymbol{H}_t)$ and its quantized counterpart $\widetilde{\boldsymbol{H}}_{t+1} = \widetilde{F}_t(\widetilde{\boldsymbol{H}}_t)$. Their discrepancy reflects both error in the quantized mapping at the current input and the difference between $\widetilde{\boldsymbol{H}}_t$ and $\boldsymbol{H}_t$. Earlier errors can therefore propagate through state reuse, consistent with the loop-boundary behavior in Figure 1.

Proposition 2. *Define the loop error as $\varepsilon_t := \|\widetilde{\boldsymbol{H}}_t - \boldsymbol{H}_t\|_F$. Assume that both recurrences start from the same initial state and, for each loop t, the full-precision function F is γ_t-Lipschitz on the*

segment between $\widetilde{\boldsymbol{H}}_t$ and $\boldsymbol{H}_t$. Then $\varepsilon_{t+1} \le \varepsilon_t^{\text{quant}} + \gamma_t \varepsilon_t$, where $\varepsilon_t^{\text{quant}} := \|\widetilde{F}_t(\widetilde{\boldsymbol{H}}_t) - F(\widetilde{\boldsymbol{H}}_t)\|_F$. Unrolling this recursion yields

$$\varepsilon_T \le \sum_{\tau=0}^{T-1} \left(\prod_{t=\tau+1}^{T-1} \gamma_t \right) \varepsilon_\tau^{\text{quant}}.$$

Thus, each loop-wise quantization error is propagated through downstream loop sensitivities, with earlier errors passing through a longer chain of transformations.

Proposition 2 bounds the final state error by loop-wise mapping errors weighted by downstream sensitivities. Errors introduced earlier pass through more loops and may have a larger effect, motivating correcting states before reuse and calibrating the quantized model along the full recurrent trajectory.

4 LoopQ: Loop-Aware PTQ for Looped Transformers

LoopQ builds on a shared, transformation-based quantized backbone (Sun et al., 2025) and introduces four coordinated adaptations. **(i)** *Loop-aware Activation Scaling* (LAS) uses loop-dependent activation scales to accommodate changes in magnitude. **(ii)** *Selective Loop-aware Transformation* (SLT) uses a sharing-sensitivity score to identify transformation groups that benefit most from loop-dependent configurations, while keeping the remaining groups shared. **(iii)** *Cross-loop Transition Adapter* (CTA) adjusts the quantized hidden state before it enters the next loop, limiting the state mismatch carried forward. **(iv)** *Trajectory-Aware Calibration* (TAC) jointly optimizes these components along the recurrent trajectory, aligning post-adapter boundary states and final predictions with the full-precision model. Implementation details and calibration procedures are provided in Appendix B.

4.1 Loop-aware Activation Scaling

Sharing quantization parameters across loops creates a trade-off when the same layer receives activations with different magnitudes. The scale-drift analysis accompanying Proposition 1 shows that, under such variation, a single activation step size may be suboptimal even if the shared transformation $\boldsymbol{P}_\ell$ is fixed. We first address this source of mismatch through loop-aware activation scaling.

The activation scale c_ℓ controls both quantization resolution and dynamic range. Because a shared module processes loop-dependent hidden states, a scale large enough to accommodate the activations in one loop may be too coarse for smaller activations in another. Conversely, a scale tuned to preserve resolution for small activations may clip larger ones (Xiao et al., 2023a; Sun et al., 2025). The former increases rounding error, while the latter discards information that subsequent loops cannot recover. Since each quantized loop output becomes the input to the next, either type of error may propagate along the recurrent trajectory. LoopQ alleviates this trade-off by replacing the shared scale c_ℓ with loop-dependent scales $c_{t,\ell}$, yielding $Q_a(x; c_{t,\ell})$.

The scales are calibrated offline and selected according to the physical loop at inference time. Each activation quantizer stores one scale per loop and feature group, a small addition relative to the shared backbone. Loop-dependent scaling adapts activation magnitudes under the shared transformation $\boldsymbol{P}_\ell$. The selective mechanism developed next addresses differences in the transformations preferred by individual loops while limiting the storage required for loop-dependent weight representations.

4.2 Selective Loop-aware Transformation

Loop-dependent scaling reduces magnitude mismatch while keeping $\boldsymbol{P}_\ell$ shared. Proposition 1 shows that sharing increases activation error when loop-specific optima conflict; a fixed-scale quadratic analysis quantifies the cost of tying their preferred transformations. This motivates separate transformations where activation geometries differ and sharing elsewhere to limit storage.

SLT ranks candidate transformation groups using a *sharing-sensitivity score* based on the recurrent calibration loss $\mathcal{L}$ (Sec. 4.4). Ideally, this score would measure the reduction in $\mathcal{L}$ obtainable by allowing loop-dependent transforms $\boldsymbol{P}_{t,\ell}$ while holding the remaining configuration fixed. Unlike the fixed-trajectory activation error in Proposition 1, this criterion evaluates the effect of relaxing sharing

under the calibration objective. A larger score indicates that a shared transform is more costly for that group.[1]

Evaluating this loss reduction exactly would require separately optimizing each candidate. We approximate its local effect on $\mathcal{L}$ using curvature. Writing $\boldsymbol{P}_{t,\ell} = \boldsymbol{P}_\ell + \boldsymbol{\Delta}_{t,\ell}$ as a local perturbation of the shared configuration, a second-order expansion for one transform occurrence gives

$$\mathcal{L}(\boldsymbol{P}_{t,\ell}) \approx \mathcal{L}(\boldsymbol{P}_\ell) + \left\langle \nabla_{\boldsymbol{P}_{t,\ell}} \mathcal{L}, \boldsymbol{\Delta}_{t,\ell} \right\rangle_F + \frac{1}{2} \left\langle \boldsymbol{\Delta}_{t,\ell}, \nabla^2_{\boldsymbol{P}_{t,\ell}} \mathcal{L}[\boldsymbol{\Delta}_{t,\ell}] \right\rangle_F . \tag{5}$$

To estimate the cost of constraining these updates to be shared across loops, we use a blockwise generalized Gauss–Newton (GGN) approximation (Martens & Grosse, 2015) based on predictive KL:

$$\boldsymbol{G}_{\ell j} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{J}_{i,\ell j}^\top \boldsymbol{F}_i \boldsymbol{J}_{i,\ell j}. \tag{6}$$

Here, $\boldsymbol{F}_i = \mathrm{diag}(\boldsymbol{p}_i) - \boldsymbol{p}_i \boldsymbol{p}_i^\top$, with $\boldsymbol{p}_i = \mathrm{softmax}(\widetilde{\boldsymbol{Z}}_i)$, is the Fisher information of the quantized predictive distribution at token i. The Jacobian $\boldsymbol{J}_{i,\ell j}$ relates its logits to rotation coordinate j of transformation group ℓ across all loops, and N counts valid prediction tokens. Each $\boldsymbol{G}_{\ell j}$ is a $T \times T$ block that captures coupling among loop occurrences of one coordinate while omitting cross-coordinate coupling. With positive damping, these blocks yield the sharing-sensitivity score S_ℓ, which estimates the local loss increase from projecting trial loop-dependent coordinates onto a shared configuration. Appendix B.3 details the curvature estimation, score computation, reversible selection procedure, and calibration schedule.

LoopQ uses these scores to select loop-dependent transformation groups during joint calibration. It first permits loop-dependent updates at all candidate sites, then progressively reduces the adaptation budget. At each stage, groups outside the budget are projected to shared coordinates but remain eligible for later selection. This reversible search allows the selected groups to change as other mismatches are corrected. After selection, the retained groups use loop-dependent transforms $\boldsymbol{P}_{t,\ell}$, the remaining groups use shared transforms $\boldsymbol{P}_\ell$, and calibration continues with this structure fixed.

4.3 Cross-loop Transition Adapter

After loop-dependent scaling and selective transformation reduce mismatch within shared layers, the resulting quantized state is still passed directly to the next loop. Proposition 2 shows how an existing state error can propagate through subsequent loop computations. LoopQ therefore introduces a lightweight Cross-loop Transition Adapter (CTA) to adjust the state at each loop boundary without modifying the shared quantized backbone.

For $t < T - 1$, the adapter takes the quantized loop input and output and produces the next loop's input, $\widetilde{\boldsymbol{H}}_{t+1,0} = A_t(\widetilde{\boldsymbol{H}}_{t,0}, \widetilde{\boldsymbol{H}}_{t,L})$, where

$$A_t(\widetilde{\boldsymbol{H}}_{t,0}, \widetilde{\boldsymbol{H}}_{t,L}) = \widetilde{\boldsymbol{H}}_{t,L} + g_t(\widetilde{\boldsymbol{H}}_{t,0}, \Delta\widetilde{\boldsymbol{H}}_t) \odot \Delta\widetilde{\boldsymbol{H}}_t + \Pi^\perp_{\Delta\widetilde{\boldsymbol{H}}_t} \left[\boldsymbol{R}_t(\widetilde{\boldsymbol{H}}_{t,0}, \Delta\widetilde{\boldsymbol{H}}_t) \right]. \tag{7}$$

Here, $\Delta\widetilde{\boldsymbol{H}}_t = \widetilde{\boldsymbol{H}}_{t,L} - \widetilde{\boldsymbol{H}}_{t,0}$ is the update produced by loop t. The gate g_t modulates this update, while $\boldsymbol{R}_t$ uses a low-rank bottleneck to generate an additional correction from the loop input and update. The projection $\Pi^\perp_{\Delta\widetilde{\boldsymbol{H}}_t}$ removes the component of that correction parallel to the update, applied independently to each token and feature group. Thus, the two branches adjust the existing update and supply a complementary directional correction before the state is reused. Their network parameters are shared across loop boundaries, with lightweight boundary-specific offsets providing adaptation. Implementation details are provided in Appendix B.4.

4.4 Trajectory-Aware Calibration

The effects of LAS, SLT, and CTA extend beyond the loop in which each component acts: changing an intermediate state also changes the inputs to subsequent loops. Trajectory-Aware Calibration (TAC) therefore optimizes these components jointly through the full recurrent computation, using the full-precision model as a teacher (Hinton et al., 2015).

[1] Selection is performed over grouped transformation sites rather than entire layers.

For calibration inputs $\boldsymbol{x} \sim \mathcal{D}$, LoopQ minimizes

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\mathrm{KL}\Big(\boldsymbol{Z} \,\Big\|\, \widetilde{\boldsymbol{Z}}\Big) + \sum_{t=0}^{T-2} \mu_t \left\|\widetilde{\boldsymbol{H}}_{t+1,0} - \boldsymbol{H}_{t+1,0}\right\|_F^2\right], \tag{8}$$

where $\mathrm{KL}(\boldsymbol{Z}\|\widetilde{\boldsymbol{Z}})$ denotes the KL divergence between the predictive distributions induced by the full-precision and quantized logits. The state $\widetilde{\boldsymbol{H}}_{t+1,0} = A_t(\widetilde{\boldsymbol{H}}_{t,0}, \widetilde{\boldsymbol{H}}_{t,L})$ is the post-adapter input to loop $t+1$ from Eq. (7).

The predictive KL aligns the final outputs, while the boundary terms match post-adapter states to their full-precision counterparts before those states are reused. The effective coefficient μ_t summarizes token-wise adaptive weighting based on relative boundary-state error and terminal predictive KL. When boundary error is large relative to prediction error, this weighting reduces the influence of state matching so that intermediate-state discrepancies do not dominate calibration. Appendix B.5 gives the exact weighting and its matrix-level formulation.

We optimize Eq. (8) through the unrolled recurrence, allowing losses at later loops to update shared transforms $\boldsymbol{P}_\ell$ and earlier loop-dependent components. The latter include activation scales $c_{t,\ell}$, the transforms $\boldsymbol{P}_{t,\ell}$ retained by SLT, and transition adapters A_t; the pretrained backbone weights remain frozen. Joint calibration thus accounts for both the boundary states passed between loops and their effects on the final prediction.

5 Experiments

5.1 Experimental Setup

Benchmarks and Baselines. Following prior quantization work, we evaluate HellaSwag (`HS`) (Zellers et al., 2019), WinoGrande (`WG`) (Sakaguchi et al., 2021), ARC-Challenge (`AC`) (Clark et al., 2018), MMLU (`MM`) (Hendrycks et al., 2020), and OpenAI LAMBADA (`LB`) (Paperno et al., 2016; Radford et al., 2019). We use `acc_norm` where available and report the mean accuracy across these five tasks. We also measure perplexity on WikiText-2 (`WT2`) (Merity et al., 2016) and LAMBADA (`OP`). The PTQ baselines are SmoothQuant (Xiao et al., 2023a), QuaRot (Ashkboos et al., 2024), SpinQuant (Liu et al., 2025), FlatQuant (Sun et al., 2025), and MixQuant (Sanjeet et al., 2026). BF16 is the full-precision reference, while symmetric round-to-nearest (RTN) quantization provides a direct-quantization baseline. Appendix C.2 evaluates LoopQ under alternative quantization settings to examine whether its benefits extend beyond static uniform quantization.

Backbones and Implementation. We evaluate four backbones from three LoopLM architectures: *Ouro* (Zhu et al., 2025), with 24 shared layers and 4 loops in its 1.4B variant and 48 shared layers and 4 loops in its 2.6B variant; shortcut-modulated *LoopFormer* (Jeddi et al., 2026), with 3 shared layers and 8 loops; and partially looped *Parcae* (Prairie et al., 2026), with 4 shared layers repeated over 8 loops and 8 non-looped layers. The main W4A4 and W4A8 comparisons use static activation quantization, symmetric uniform quantizers, and round-to-nearest discretization. All static PTQ baselines collect activation statistics across the complete recursive forward pass. LoopQ uses 1,024 calibration examples and FlatQuant-style Kronecker-decomposed transforms (Sun et al., 2025). Implementation and calibration details appear in Appendix B.

5.2 Quantization Performance Across Tasks and Backbones

Task Accuracy. Table 1 reports W4A4 and W4A8 results across four backbones, with the largest gains under W4A4. On Ouro 1.4B, LoopQ improves five-task mean accuracy by 33.47 percentage points over the strongest static baseline, retaining 92.23% of BF16 accuracy. On Parcae, it improves LAMBADA accuracy by 16.98 points. Across all four backbones under W4A4, the average relative gain in five-task mean accuracy is 67.8%. LAS and SLT reduce mismatch through loop-dependent scales and selective transforms, consistent with Proposition 1.

Perplexity and Error Propagation. We focus on LAMBADA (`OP`) perplexity; WikiText-2 (`WT2`) shows a similar overall trend. On Ouro 1.4B, all static W4A4 baselines exceed 10^3 OP PPL, while LoopQ achieves 8.20. On Parcae, LoopQ reduces OP PPL by 86.0% relative to the strongest static baseline. Across both datasets and all four backbones, a conservative estimate of the average relative

Table 1: Main results under W4A4 and W4A8. PPL values above 10^3 are displayed as $> 10^3$.

		W4A4							W4A8						
		Tasks ↑					Perplexity ↓		Tasks ↑					Perplexity ↓	
Model	**Method**	**HS**	**WG**	**LB**	**AC**	**MM**	**WT2**	**OP**	**HS**	**WG**	**LB**	**AC**	**MM**	**WT2**	**OP**
Ouro 1.4B	*BF16*	*0.7055*	*0.6606*	*0.6146*	*0.5341*	*0.6886*	*11.18*	*6.22*	*0.7055*	*0.6606*	*0.6146*	*0.5341*	*0.6886*	*11.18*	*6.22*
	RTN	0.2777	0.4941	0.0033	0.2389	0.2282	$> 10^3$	$> 10^3$	0.5022	0.5866	0.3220	0.3745	0.3569	43.89	46.72
	SmoothQuant	0.2575	0.5170	0.0000	0.2457	0.2499	$> 10^3$	$> 10^3$	0.6010	0.5722	0.2851	0.4104	0.2910	33.27	60.21
	QuaRot	0.2908	0.4972	0.0293	0.2270	0.2366	451.44	$> 10^3$	0.6817	0.6195	0.5523	0.4983	0.5577	15.27	8.95
	SpinQuant	0.2914	0.4925	0.0078	0.2210	0.2352	653.78	$> 10^3$	0.4838	0.5630	0.3647	0.3745	0.3240	27.88	27.61
	FlatQuant	0.2698	0.4483	0.0000	0.2278	0.2358	$> 10^3$	$> 10^3$	0.6948	**0.6511**	0.5962	0.4983	**0.6336**	13.53	6.92
	MixQuant	0.2719	0.4665	0.0006	0.2329	0.2409	$> 10^3$	$> 10^3$	0.6769	0.6125	0.4956	0.5068	0.4464	13.94	12.67
	LoopQ	**0.6689**	**0.6125**	**0.5812**	**0.4898**	**0.6022**	**13.22**	**8.20**	**0.6954**	0.6456	**0.6255**	**0.5145**	0.6206	**13.29**	**6.09**
Ouro 2.6B	*BF16*	*0.7636*	*0.6867*	*0.7013*	*0.5802*	*0.7380*	*10.51*	*4.22*	*0.7636*	*0.6867*	*0.7013*	*0.5802*	*0.7380*	*10.51*	*4.22*
	RTN	0.2811	0.5043	0.0361	0.2244	0.2311	962.74	$> 10^3$	0.5517	0.5777	0.4093	0.3729	0.3386	17.21	22.07
	SmoothQuant	0.2589	0.5012	0.0000	0.2662	0.2455	$> 10^3$	$> 10^3$	0.3219	0.5099	0.0693	0.2363	0.2297	198.38	$> 10^3$
	QuaRot	0.2967	0.4964	0.0782	0.2329	0.2307	301.26	$> 10^3$	0.5389	0.5659	0.4157	0.3874	0.4419	15.78	21.53
	SpinQuant	0.2805	0.4862	0.0200	0.2193	0.2316	621.14	$> 10^3$	0.5651	0.5754	0.4611	0.4044	0.4432	15.81	17.26
	FlatQuant	0.2560	0.4917	0.0000	0.2560	0.2455	$> 10^3$	$> 10^3$	0.6393	0.6085	0.3895	0.4309	0.4519	25.51	26.07
	MixQuant	0.2900	0.4862	0.0068	0.2406	0.2347	853.32	$> 10^3$	0.7447	0.6740	0.6272	0.5503	**0.7145**	13.92	6.27
	LoopQ	**0.6860**	**0.6227**	**0.5680**	**0.5034**	**0.5265**	**15.89**	**8.17**	**0.7555**	**0.6780**	**0.6837**	**0.5819**	0.7007	**11.38**	**4.47**
LoopFormer 3×8	*BF16*	*0.3218*	*0.5107*	*0.3776*	*0.2457*	*0.2306*	*35.62*	*26.26*	*0.3218*	*0.5107*	*0.3776*	*0.2457*	*0.2306*	*35.62*	*26.26*
	RTN	0.2731	0.4988	0.0016	**0.2457**	0.2391	$> 10^3$	$> 10^3$	0.3156	0.5099	0.3621	**0.2389**	**0.2311**	39.00	29.51
	SmoothQuant	0.2653	0.4972	0.0004	0.2440	**0.2418**	$> 10^3$	$> 10^3$	0.3019	0.5178	0.1296	0.2363	0.2301	113.70	396.20
	QuaRot	0.3087	0.5264	0.0869	0.2389	0.2295	56.05	290.49	**0.3207**	0.5209	0.3705	0.2363	0.2303	37.47	29.87
	SpinQuant	0.2959	0.5051	0.2331	0.2176	0.2301	98.20	88.91	0.3049	0.5146	0.2451	0.2244	0.2291	42.83	92.20
	FlatQuant	0.2782	0.5170	0.0276	0.2184	0.2318	462.39	$> 10^3$	0.3186	0.5193	0.3768	0.2338	0.2300	38.25	**26.85**
	MixQuant	0.3092	**0.5406**	0.2822	0.2261	0.2304	39.73	54.82	0.3188	**0.5280**	0.3578	0.2346	0.2302	41.87	30.79
	LoopQ	**0.3176**	0.5043	**0.3466**	0.2372	0.2301	**37.36**	**34.46**	0.3174	0.5217	**0.3788**	0.2381	0.2301	**37.44**	27.62
Parcae 370M	*BF16*	*0.4409*	*0.5446*	*0.3565*	*0.2910*	*0.2558*	*26.98*	*32.55*	*0.4409*	*0.5446*	*0.3565*	*0.2910*	*0.2558*	*26.98*	*32.55*
	RTN	0.2992	0.5091	0.0097	0.2227	0.2455	797.53	$> 10^3$	0.3983	0.5130	0.1013	0.2688	0.2447	82.00	645.47
	SmoothQuant	0.3252	0.4980	0.0833	0.2270	0.2379	489.94	914.59	0.3511	0.5162	0.1310	0.2312	0.2355	162.19	250.89
	QuaRot	0.3534	0.5146	0.1289	0.2321	0.2349	273.18	454.18	0.4214	0.5272	0.3266	0.2756	0.2334	34.41	45.35
	SpinQuant	0.3057	**0.5185**	0.0305	0.2201	0.2354	837.26	$> 10^3$	0.3765	0.5051	0.1807	0.2568	0.2481	207.42	276.29
	FlatQuant	0.3191	0.4972	0.0695	0.2278	0.2310	219.36	$> 10^3$	0.4319	**0.5422**	0.3278	0.2867	0.2366	28.80	40.05
	MixQuant	0.3261	0.5162	0.0534	0.2526	0.2312	186.12	$> 10^3$	0.4085	0.5178	0.2309	0.2747	0.2297	53.01	101.24
	LoopQ	**0.4088**	**0.5185**	**0.2987**	**0.2782**	**0.2480**	**32.85**	**63.71**	**0.4340**	0.5288	**0.3466**	**0.2892**	**0.2608**	**27.68**	**37.07**

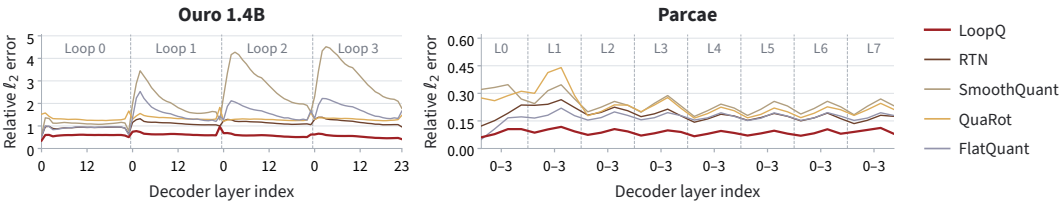


Figure 2: Relative quantization error ℓ_2 across layers and loops on LAMBADA.

PPL reduction is 75.2%, obtained by substituting 10^3 for baseline PPL entries reported as $> 10^3$ before averaging the eight relative reductions. TAC addresses this degradation by jointly calibrating boundary states and final predictions.

Backbone-level Robustness. Recovery varies across architectures. Under W4A4, LoopFormer and Parcae retain 97.0% and 92.8% of BF16 mean accuracy, while both Ouro models show larger gaps despite substantial gains over static baselines. Ouro reuses its full Transformer stack across four loops, versus three shared layers in LoopFormer and four in Parcae. CTA applies corrections at loop boundaries, leaving more quantized layers between corrections on Ouro.

Error along Loop Trajectories. Figure 2 compares relative ℓ_2 errors across layers and loops on LAMBADA. LoopQ maintains lower errors than the static baselines on both Ouro and Parcae, and reduces the pronounced peaks near loop transitions on Ouro. Consistent with Proposition 2, CTA corrects boundary states before reuse.

5.3 Reasoning and Component Ablations

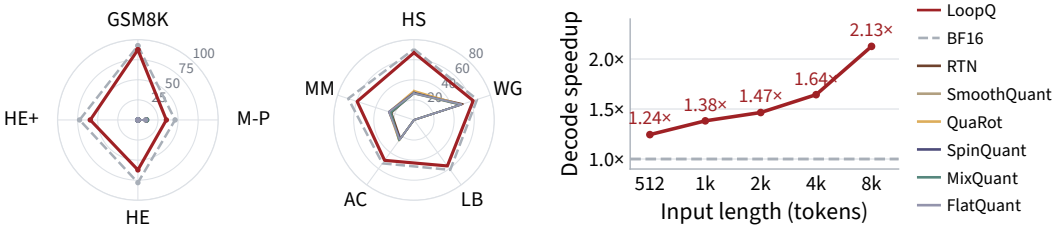
Reasoning and Code Generation. We evaluate reasoning on GSM8K (Cobbe et al., 2021) and MMLU-Pro (`M-P`) (Wang et al., 2024), and code generation on HumanEval (`HE`) (Chen et al., 2021)

Table 2: Ablations on Ouro 1.4B-Thinking under W4A4KV4. *Unrolled FlatQuant* calibrates each of the four unrolled loops independently.

Method	Task accuracy ↑						Perplexity ↓	
	HS	WG	LB	AC	MM	Mean	WT2	OP
BF16	*70.55*	*66.06*	*61.46*	*53.41*	*68.86*	*64.07*	*11.18*	*6.22*
Unrolled FlatQuant	*68.61*	*65.04*	*59.98*	*51.96*	*61.57*	*61.43*	*13.13*	*6.72*
QuaRot (KV16)	29.08	49.72	2.93	22.70	23.66	25.62	451.44	$> 10^3$
LAS Only	52.13	55.09	18.86	38.91	27.73	38.54	37.89	143.87
w/o CTA	66.14	59.91	55.21	48.63	57.41	57.46	14.03	9.59
w/o Boundary Loss	66.43	61.40	**57.48**	47.70	59.03	58.41	13.51	**8.84**
Random SLT	66.46	60.77	53.43	46.50	56.94	56.82	14.02	10.48
Full LoopQ	**66.89**	**62.04**	56.61	**49.74**	**59.51**	**58.96**	**13.41**	8.98

Table 3: Deployment memory and five-task mean accuracy.

	Method	GiB ↓	Mean ↑
BF16	Ouro 1.4B	2.67	64.07
	Qwen3-1.7B	2.99	54.10
	Qwen3-4B	6.98	63.18
W4A4	*Unrolled Ouro*	*2.83*	*61.43*
	Qwen3-1.7B	1.92	49.14
	Qwen3-4B	3.38	60.00
	Ouro FlatQuant	1.03	23.63
	Ouro LoopQ	**1.13**	**58.96**



(a) Reasoning and code (b) KV4 task accuracy (c) Decode speedup

Figure 3: Additional evaluations on Ouro 1.4B. (a) Accuracy on reasoning and code-generation tasks (%). (b) Task accuracy (%) under W4A4KV4. (c) Packed W4A4KV4 decode speedup over BF16.

and HumanEval+ (HE+) (Liu et al., 2023). Figure 3a shows that most PTQ baselines struggle on these tasks, whereas LoopQ retains substantial capability. LoopQ achieves 87.3% on GSM8K and 62.2%/59.3% on HE/HE+, compared with 92.3% and 78.0%/72.6% for BF16. These tasks often require long autoregressive outputs, during which deviations in earlier predictions can alter the context for later tokens. This effect extends beyond the within-forward-pass error propagation characterized by Proposition 2.

Ablation Study. Table 2 shows how LoopQ's components improve Ouro 1.4B-Thinking. LAS alone achieves 38.54% five-task mean accuracy, compared with 25.62% for the strongest listed static baseline. Adding SLT and trajectory-aware calibration, but not CTA, raises the mean to 57.46% and reduces LAMBADA PPL from 143.87 to 9.59. Scale adaptation therefore helps but does not account for the full recovery. Adding CTA further improves all five task accuracies, bringing the mean to 58.96%. Selection and boundary alignment also matter: replacing GGN-based selection with six random groups lowers the mean by 2.14 points, while removing the boundary loss lowers it by 0.55 points despite improving some individual metrics. Appendix C.1 shows the groups selected by SLT.

Dynamic Quantization Settings. Table 4 compares INT4 and NVFP4 under dynamic PTG quantization, with activation ranges estimated online in both settings. Under INT4, LoopQ exceeds FlatQuant by 3.81 percentage points in mean accuracy. Under NVFP4, it achieves 0.6143 mean accuracy, outperforming MixQuant by 0.28 points and FlatQuant by 1.38 points, with the lowest WikiText-2 word perplexity. MR-GPTQ (Egiazarian et al., 2026) is designed specifically for NVFP4; its INT4 entries are therefore N/A. Full results appear in Appendix C.2.

5.4 Deployment Efficiency

Model-State Storage. Table 3 shows that LoopQ adds 0.10 GiB over Ouro FlatQuant while improving five-task mean accuracy by 35.33 percentage points. Unrolled FlatQuant gains another 2.47 points but requires 2.83 GiB because it stores separate backbone copies. LoopQ achieves nearly the same accuracy at 1.13 GiB, 57.68% less than Ouro BF16. It also reports higher accuracy and a smaller footprint than quantized Qwen3-1.7B, although Qwen3 memory is measured after loading and Ouro memory is estimated. Appendix C.4 details these memory-footprint accounting conventions.

Table 4: Alternative quantization results on Ouro 1.4B-Thinking.

Setting	Metric	RTN	MR-GPTQ	SmoothQuant	QuaRot	MixQuant	FlatQuant	LoopQ
INT4	Mean ↑	0.2555	N/A	0.2538	0.3722	0.2775	0.5461	**0.5842**
	WT2 PPL ↓	$> 10^3$		$> 10^3$	62.89	225.54	18.84	**16.63**
NVFP4	Mean ↑	0.5762	0.5962	0.5870	0.6053	0.6116	0.6005	**0.6143**
	WT2 PPL ↓	16.35	14.97	15.50	15.38	15.16	15.21	**14.89**

KV-Cache Quantization. Under W4A4KV4 on Ouro 1.4B-Thinking, static baselines achieve at most 0.02% LAMBADA accuracy, whereas LoopQ reaches 56.61% and achieves 58.96% five-task mean accuracy (Figure 3b). Its WikiText-2 and LAMBADA PPL remain at 13.41 and 8.98, respectively, showing that LoopQ's recovery extends to quantized caches.

Decode Throughput. Figure 3c reports packed W4A4KV4 inference on an RTX 5090 with $1{,}024$ generated tokens per request. LoopQ achieves $1.24\times$ BF16 decode throughput at 512 prompt tokens and $2.13\times$ at $8{,}192$, where both runtimes slow down but the relative advantage grows. These are achieved-throughput comparisons under the stated configuration; Appendix C.4 reports the measurement protocol and component-level overhead.

6 Related Work

PTQ for LLMs. Post-training quantization (PTQ) reduces model storage and inference data movement without end-to-end retraining (Alizadeh et al., 2020; Shkolnik et al., 2020; Xiao et al., 2023b; Sun et al., 2025; Liu et al., 2025; Egashira et al., 2024; Lin et al., 2024; Frantar et al., 2022). Weight-only methods limit quantization error using approximate second-order information (Frantar et al., 2022) or protect weights associated with salient activations (Lin et al., 2024), but leave activations at higher precision. Joint weight-activation methods address activation outliers by retaining selected dimensions at higher precision (Dettmers et al., 2022), allocating precision by salience (Huang et al., 2024), or preserving high-variance subspaces (Saxena et al., 2025). Transformation-based methods reshape activations through channel-wise scaling (Xiao et al., 2023a), rotations (Ashkboos et al., 2024; Liu et al., 2025), or more flexible transformations (Hu et al., 2025; Sun et al., 2025; Sanjeet et al., 2026). LoopQ retains a shared quantized backbone while adapting selected scales and transformations across loops and calibrating along the recurrent trajectory.

Looped Language Models. Transformer compression has used pruning, merging, and layer reorganization (Imfeld et al., 2023; Liu et al., 2024; Zhou et al., 2023; Verma et al., 2025). LoopLMs instead increase computational depth by repeatedly applying shared layers. Universal Transformer introduced recurrent Transformer computation (Dehghani et al., 2018), and ALBERT demonstrated extensive cross-layer parameter sharing (Lan et al., 2019). Recent architectures explore varied forms of recursion (Bae et al., 2025; Zhu et al., 2025; Zeitoun et al., 2026; Prairie et al., 2026; Jeddi et al., 2026), while inference systems exploit weight reuse under memory and bandwidth constraints (Zhou et al., 2022; Laguna et al., 2022; Kwon et al., 2026; Gupta et al., 2026; Riera et al., 2022). These studies primarily address architecture, parameter efficiency, and full-precision behavior; low-bit quantization of recurrent hidden-state trajectories remains less explored.

7 Conclusion

In this paper, we present **LoopQ**, a loop-aware PTQ framework for LoopedLMs. We show that LoopLMs face unique quantization challenges caused by loop-dependent distribution shifts, transition-state reuse, and recursive error accumulation. LoopQ addresses these challenges by preserving a shared quantized backbone while adding lightweight loop-dependent adaptations, including activation scaling, selective transformation, cross-loop state alignment, and trajectory-aware calibration. Experiments across multiple LoopLM backbones and benchmarks demonstrate that LoopQ consistently improves low-bit robustness over static PTQ baselines, especially under W4A4 quantization. LoopQ also preserves early-exit capability and achieves up to $2.13\times$ BF16 decode throughput in packed deployment.

AI USE STATEMENT

We used generative AI tools to assist with implementing portions of the proposed method and with editing the manuscript to improve readability and language clarity. All AI-assisted code was reviewed, verified, and tested by the authors, and all AI-assisted edits to the manuscript were reviewed for correctness. We take full responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies post-training quantization of existing language models using public calibration and evaluation datasets; it does not involve new human-subject data collection. Quantization can change model outputs, including behavior relevant to bias, harmful content, and privacy. Our accuracy and efficiency results do not establish that such behavior is preserved. Deployments of quantized checkpoints should therefore evaluate these risks separately.

REPRODUCIBILITY STATEMENT

We specify LoopQ's quantization and calibration objectives in the main text and provide implementation details, including group selection and the transition adapter, in Appendices B.1–B.5. Proofs appear in Appendix A. The experimental sections and appendices document the calibration data, quantization settings, task-level results, and protocols for measuring deployment memory and inference throughput. Source code is available at https://anonymous.4open.science/r/LoopQ/.

REFERENCES

Milad Alizadeh, Arash Behboodi, Mart van Baalen, Christos Louizos, Tijmen Blankevoort, and Max Welling. Gradient l1 regularization for quantization robustness. In *International Conference on Learning Representations*, 2020.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting llm quantization. *Advances in Neural Information Processing Systems*, 37:41709–41732, 2024.

Vage Egiazarian, Roberto Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the gap between promise and performance for microscaling fp4 quantization. In *International Conference on Learning Representations*, volume 2026, pp. 113529–113563, 2026.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Neelesh Gupta, Peter Wang, Rajgopal Kannan, and Viktor K Prasanna. A persistent-state dataflow accelerator for memory-bound linear attention decode on fpga. *arXiv preprint arXiv:2603.05931*, 2026.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems*, volume 5, pp. 164–171, 1992.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Xing Hu, Yuan Cheng, Dawei Yang, Zukang Xu, Zhihang Yuan, Jiangyong Yu, Chen Xu, Zhe Jiang, and Sifan Zhou. Ostquant: Refining large language model quantization with orthogonal and scaling transformations for better distribution fitting. *arXiv preprint arXiv:2501.13987*, 2025.

Wei Huang, Haotong Qin, Yangdong Liu, Yawei Li, Qinshuo Liu, Xianglong Liu, Luca Benini, Michele Magno, Shiming Zhang, and Xiaojuan Qi. Slim-llm: Salience-driven mixed-precision quantization for large language models. *arXiv preprint arXiv:2405.14917*, 2024.

Moritz Imfeld, Jacopo Graldi, Marco Giordano, Thomas Hofmann, Sotiris Anagnostidis, and Sidak Pal Singh. Transformer fusion with optimal transport. *arXiv preprint arXiv:2310.05719*, 2023.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. *arXiv preprint arXiv:2602.11451*, 2026.

Junhan Kim, Yeo Jeong Park, Seungwoo Son, Chungman Lee, Ho-young Kim, Joonyoung Kim, and Yongkweon Jeon. Turboboa: Faster and exact attention-aware quantization without backpropagation. *arXiv preprint arXiv:2602.04929*, 2026.

Hyucksung Kwon, Kyungmo Koo, Janghyeon Kim, Woongkyu Lee, Minjae Lee, Gyeonggeun Jung, Hyungdeok Lee, Yousub Jung, Jaehan Park, Yosub Song, et al. Pimphony: Overcoming bandwidth and capacity inefficiency in pim-based long-context llm inference system. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–21. IEEE, 2026.

Ann Franchesca Laguna, Mohammed Mehdi Sharifi, Arman Kazemi, Xunzhao Yin, Michael Niemier, and X Sharon Hu. Hardware-software co-design of an in-memory transformer network accelerator. *Frontiers in Electronics*, 3:847069, 2022.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

Deyuan Liu, Zhanyue Qin, Hairu Wang, Zhao Yang, Zecheng Wang, Fangying Rong, Qingbin Liu, Yanchao Hao, Bo Li, Xi Chen, et al. Pruning via merging: Compressing llms via manifold alignment based layer merging. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17817–17829, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*, 2025.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Marc Riera, José María Arnau, and Antonio González. Crew: Computation reuse and efficient weight storage for hardware-accelerated mlps and rnns. *Journal of Systems Architecture*, 129:102604, 2022.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sai Sanjeet, Ian Colbert, Pablo Monteagudo-Lago, Giuseppe Franco, Yaman Umuroglu, and Nicholas J Fraser. Mixquant: Pushing the limits of block rotations in post-training quantization. *arXiv preprint arXiv:2601.22347*, 2026.

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. Resq: Mixed-precision quantization of large language models with low-rank residuals. In *International Conference on Machine Learning*, pp. 53095–53114. PMLR, 2025.

Moran Shkolnik, Brian Chmiel, Ron Banner, Gil Shomron, Yury Nahshan, Alex Bronstein, and Uri Weiser. Robust quantization: one model to rule them all. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 5308–5317, 2020.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, et al. Flatquant: Flatness matters for llm quantization. In *International Conference on Machine Learning*, pp. 57587–57613. PMLR, 2025.

Neha Verma, Kenton Murray, and Kevin Duh. Merging feed-forward sublayers for compressed transformers. *arXiv preprint arXiv:2501.06126*, 2025.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023a.

Yisong Xiao, Aishan Liu, Tianyuan Zhang, Haotong Qin, Jinyang Guo, and Xianglong Liu. Robustmq: benchmarking robustness of quantized models. *Visual Intelligence*, 1(1):30, 2023b.

Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1071–1085. IEEE, 2022.

Wangchunshu Zhou, Ronan Le Bras, and Yejin Choi. Modular transformers: Compressing transformers into modularized layers for flexible efficient inference. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 10452–10465, 2023.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

Table 5: Summary of notation.

Notation	Meaning
$\boldsymbol{x}, \mathcal{D}$	Input token sequence and calibration set.
T, L, n, d, v	Number of loops, shared layers, tokens, hidden features, and vocabulary entries.
t, ℓ, i, j	Loop, layer (or selectable group), token, and coordinate indices.
$f_\ell(\cdot; \boldsymbol{\theta}_\ell)$	Full-precision layer and its parameter vector.
$\boldsymbol{H}_{t,\ell}, \widetilde{\boldsymbol{H}}_{t,\ell}$	Teacher and quantized hidden-state matrices.
$\boldsymbol{H}_t = \boldsymbol{H}_{t,0}$	Input-state matrix of loop t.
$\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P})$	Transformed teacher activation $\boldsymbol{H}_{t,\ell-1}(\boldsymbol{x})\boldsymbol{P}$.
$(\boldsymbol{H}_{t,\ell})_{i,:}$	Token row i of $\boldsymbol{H}_{t,\ell}$.
$F, \widetilde{F}_t$	Teacher loop map and quantized loop map.
$\boldsymbol{Z}, \widetilde{\boldsymbol{Z}}$	Teacher and student terminal logit matrices.
$\boldsymbol{W}_\ell, \boldsymbol{P}_\ell$	Frozen backbone weight and shared invertible transform.
Q_a, Q_w	Activation and weight quantization–dequantization maps.
$c_\ell, c_{t,\ell}$	Shared and loop-dependent static activation quantization steps.
$q_{\min}, q_{\max}$	Minimum and maximum integer quantization levels.
$\varepsilon_t, \varepsilon_t^{\text{quant}}$	Propagated state error and local quantization error.
γ_t	Local Lipschitz factor of the teacher loop map.
$\boldsymbol{P}_{t,\ell}, \boldsymbol{\Delta}_{t,\ell}$	Loop-dependent transform and its perturbation from a shared transform.
$\boldsymbol{\phi}_{\ell j}$	Occurrences of rotation coordinate j of group ℓ across loops.
S_ℓ	Sharing-sensitivity score for selecting loop-dependent transforms.
A_t	Adapter between loops t and $t+1$.
$\Delta\widetilde{\boldsymbol{H}}_t$	Current loop update.
$\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{b}_t$	Shared low-rank factors of sizes $d \times r$ and $2d \times r$, and loop-dependent latent offset.
$\mathcal{L}$	Unified calibration objective.
$\mathrm{KL}(\boldsymbol{Z} \Vert \widetilde{\boldsymbol{Z}})$	Mean predictive KL associated with the terminal logits.
$k_i, e_{t,i}$	Token-level terminal KL and calibrated relative boundary error.
μ_t	Effective adaptive coefficient for the matrix-level boundary loss.
$\boldsymbol{G}_{\ell j}, \boldsymbol{C}_{\ell j}$	Cross-loop GGN block and its damped estimate.
$\boldsymbol{p}_i, \boldsymbol{F}_i$	Quantized predictive distribution and Fisher information matrix at token i.
$g_t, \boldsymbol{R}_t, \Pi^\perp$	Input-dependent gate, scaled bottleneck correction, and projection removing the component along the loop update.

A Notation and Theoretical Analysis

A.1 Notation Table

A.2 Proof of Proposition 1

Proposition 1. *For $\boldsymbol{x} \sim \mathcal{D}$, let $\boldsymbol{H}_{t,\ell-1}(\boldsymbol{x})$ be the hidden state on the fixed full-precision trajectory, and define $\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P}) := \boldsymbol{H}_{t,\ell-1}(\boldsymbol{x})\boldsymbol{P}$. With Q_a from Eq. (4) applied elementwise, the expected transformed-activation quantization error is*

$$\varepsilon_{t,\ell}(\boldsymbol{P}, c) := \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\|Q_a(\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P}); c) - \boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P})\|_F^2\right].$$

Assume that all minima below are finite and attained over the same admissible set of invertible $\boldsymbol{P}$ and $c > 0$. Let $\varepsilon^*_{t,\ell} := \min_{\boldsymbol{P},c} \varepsilon_{t,\ell}(\boldsymbol{P}, c)$ and define the cost of sharing one configuration across loops as

$$\Delta_\ell^{\text{share}} := \min_{\boldsymbol{P}_\ell, c_\ell} \sum_{t=0}^{T-1} \varepsilon_{t,\ell}(\boldsymbol{P}_\ell, c_\ell) - \sum_{t=0}^{T-1} \varepsilon^*_{t,\ell}.$$

Then $\Delta_\ell^{\text{share}} \geq 0$, with equality if and only if a single configuration is optimal for every loop. In particular, if any two loops have disjoint sets of optimal configurations, then $\Delta_\ell^{\text{share}} > 0$: every shared configuration incurs excess transformed-activation quantization error in at least one loop.

Proof. We prove the claim by contradiction.

For any shared configuration $(\boldsymbol{P}_\ell, c_\ell)$, the loop-wise activation quantization error satisfies

$$\varepsilon_{t,\ell}(\boldsymbol{P}_\ell, c_\ell) \geq \varepsilon^*_{t,\ell}.$$

Summing over loops and minimizing over the shared configuration gives $\Delta^{\text{share}}_\ell \geq 0$.

Suppose that shared calibration incurs zero total excess error. Let $(\boldsymbol{P}^*_\ell, c^*_\ell)$ attain the shared minimum. Then

$$\sum_{t=0}^{T-1} \left[\varepsilon_{t,\ell}(\boldsymbol{P}^*_\ell, c^*_\ell) - \varepsilon^*_{t,\ell}\right] = 0.$$

Since every summand is nonnegative, each must vanish. Thus, $(\boldsymbol{P}^*_\ell, c^*_\ell)$ is simultaneously optimal for all loops. In particular, if two loops have disjoint sets of optimal configurations, this contradicts their incompatibility. Conversely, a common optimal configuration attains every loop-wise minimum and therefore gives $\Delta^{\text{share}}_\ell = 0$.

Consequently, when no common optimum exists, every shared configuration satisfies

$$\sum_{t=0}^{T-1} \left[\varepsilon_{t,\ell}(\boldsymbol{P}_\ell, c_\ell) - \varepsilon^*_{t,\ell}\right] \geq \Delta^{\text{share}}_\ell > 0.$$

At least one loop must therefore incur strictly positive excess error. The proposition follows. □

Implications for loop-dependent calibration. To relate this excess activation error to loop-dependent activation distributions, we first consider scale drift. Fix a shared transformation $\boldsymbol{P}_\ell$ and suppose that

$$\boldsymbol{X}_{t,\ell}(\boldsymbol{x}; \boldsymbol{P}_\ell) \overset{d}{=} s_t \mathbf{R}, \qquad s_t > 0,$$

where $\boldsymbol{x} \sim \mathcal{D}$ and $\mathbf{R}$ has a common normalized distribution across loops with a finite second moment. For a symmetric uniform quantizer with fixed integer bounds,

$$Q_a(s\boldsymbol{X}; c) = sQ_a(\boldsymbol{X}; c/s), \qquad s > 0.$$

Therefore,

$$\varepsilon_{t,\ell}(\boldsymbol{P}_\ell, c) = s_t^2 \mathbb{E}\Big[\|Q_a(\mathbf{R}; c/s_t) - \mathbf{R}\|_F^2\Big] = s_t^2 \varepsilon_{\text{ref}}(c/s_t),$$

where $\varepsilon_{\text{ref}}(\alpha) := \mathbb{E}[\|Q_a(\mathbf{R}; \alpha) - \mathbf{R}\|_F^2]$. Thus, optimizing c for loop t is equivalent to optimizing the normalized step size $\alpha = c/s_t$.

Assume that ε_{ref} has a unique minimizer $\alpha^* > 0$. The optimal step size for loop t is then

$$c^*_{t,\ell} = s_t \alpha^*.$$

If a shared step size c_ℓ were simultaneously optimal for two loops $t \neq t'$, it would have to satisfy

$$c_\ell = s_t \alpha^* = s_{t'} \alpha^*.$$

Since $\alpha^* > 0$, this equality implies $s_t = s_{t'}$, contradicting the scale-drift condition. Hence, under these assumptions, distinct activation scales yield incompatible optimal step sizes. Applying the proposition to calibration with $\boldsymbol{P}_\ell$ fixed gives strictly positive excess activation error whenever the shared minimum is attained.

We next consider directional mismatch. Hold the activation scales fixed and let q_ℓ be a positive-definite quadratic surrogate for the recurrent calibration objective over loop-dependent transforms. Denote its unrestricted minimizer by $\{\boldsymbol{P}^*_{t,\ell}\}$. Write $\boldsymbol{D}_t := \boldsymbol{P}_{t,\ell} - \boldsymbol{P}^*_{t,\ell}$. Completing the square gives

$$q_\ell(\{\boldsymbol{P}_{t,\ell}\}) = q_\ell(\{\boldsymbol{P}^*_{t,\ell}\}) + \frac{1}{2} \sum_{t,t'=0}^{T-1} \left\langle \boldsymbol{D}_t, \mathcal{C}_\ell^{t,t'}[\boldsymbol{D}_{t'}] \right\rangle_F,$$

where the blocks $\mathcal{C}_\ell^{t,t'}$ form a symmetric positive-definite curvature operator. The off-diagonal blocks account for interactions between loops. Under the sharing constraint $\boldsymbol{P}_{t,\ell} = \boldsymbol{P}_\ell$ for every t, write $\boldsymbol{D}_t(\boldsymbol{P}) := \boldsymbol{P} - \boldsymbol{P}^*_{t,\ell}$. The excess loss becomes

$$\Delta^{\text{quad}}_\ell = \min_{\boldsymbol{P}_\ell} \frac{1}{2} \sum_{t,t'=0}^{T-1} \left\langle \boldsymbol{D}_t(\boldsymbol{P}_\ell), \mathcal{C}_\ell^{t,t'}[\boldsymbol{D}_{t'}(\boldsymbol{P}_\ell)] \right\rangle_F.$$

Positive definiteness implies that this minimum is zero if and only if a single $\boldsymbol{P}_\ell$ equals every $\boldsymbol{P}^*_{t,\ell}$. If two loops favor different transforms, no shared transform can satisfy this condition, and $\Delta^{\text{quad}}_\ell > 0$.

Thus, scale drift can produce incompatible optimal step sizes, while conflicting transformation updates incur a local cost when constrained to be shared. The latter cost provides the basis for SLT's sharing-sensitivity score, computed in rotation coordinates using a damped GGN approximation (Appendix B.3). The loss increase is exact for the quadratic surrogate; its use for the recurrent calibration loss is a local approximation.

A.3 Proof of Proposition 2

Proposition 2. *Define the loop error as $\varepsilon_t := \|\widetilde{\boldsymbol{H}}_t - \boldsymbol{H}_t\|_F$. Assume that both recurrences start from the same initial state and, for each loop t, the full-precision function F is γ_t-Lipschitz on the segment between $\widetilde{\boldsymbol{H}}_t$ and $\boldsymbol{H}_t$. Then $\varepsilon_{t+1} \le \varepsilon_t^{\text{quant}} + \gamma_t \varepsilon_t$, where $\varepsilon_t^{\text{quant}} := \|\widetilde{F}_t(\widetilde{\boldsymbol{H}}_t) - F(\widetilde{\boldsymbol{H}}_t)\|_F$. Unrolling this recursion yields*

$$\varepsilon_T \le \sum_{\tau=0}^{T-1} \left(\prod_{t=\tau+1}^{T-1} \gamma_t \right) \varepsilon_\tau^{\text{quant}}.$$

Thus, each loop-wise quantization error is propagated through downstream loop sensitivities, with earlier errors passing through a longer chain of transformations.

Proof. By definition,

$$\varepsilon_{t+1} = \|\widetilde{\boldsymbol{H}}_{t+1} - \boldsymbol{H}_{t+1}\|_F = \|\widetilde{F}_t(\widetilde{\boldsymbol{H}}_t) - F(\boldsymbol{H}_t)\|_F.$$

Applying the triangle inequality gives

$$\varepsilon_{t+1} \le \|\widetilde{F}_t(\widetilde{\boldsymbol{H}}_t) - F(\widetilde{\boldsymbol{H}}_t)\|_F + \|F(\widetilde{\boldsymbol{H}}_t) - F(\boldsymbol{H}_t)\|_F.$$

By the definition of the loop-wise quantization error,

$$\|\widetilde{F}_t(\widetilde{\boldsymbol{H}}_t) - F(\widetilde{\boldsymbol{H}}_t)\|_F = \varepsilon_t^{\text{quant}}.$$

Since F is γ_t-Lipschitz on the segment between $\widetilde{\boldsymbol{H}}_t$ and $\boldsymbol{H}_t$, we have

$$\|F(\widetilde{\boldsymbol{H}}_t) - F(\boldsymbol{H}_t)\|_F \le \gamma_t \|\widetilde{\boldsymbol{H}}_t - \boldsymbol{H}_t\|_F = \gamma_t \varepsilon_t.$$

Combining the two bounds yields

$$\varepsilon_{t+1} \le \varepsilon_t^{\text{quant}} + \gamma_t \varepsilon_t.$$

Unrolling this recursion from 0 to $T-1$ gives

$$\varepsilon_T \le \left(\prod_{t=0}^{T-1} \gamma_t \right) \varepsilon_0 + \sum_{\tau=0}^{T-1} \left(\prod_{t=\tau+1}^{T-1} \gamma_t \right) \varepsilon_\tau^{\text{quant}},$$

where the empty product is defined as 1. Since the full-precision and quantized recurrences start from the same initial state, $\widetilde{\boldsymbol{H}}_0 = \boldsymbol{H}_0$, we have $\varepsilon_0 = 0$. Therefore,

$$\varepsilon_T \le \sum_{\tau=0}^{T-1} \left(\prod_{t=\tau+1}^{T-1} \gamma_t \right) \varepsilon_\tau^{\text{quant}}.$$

Thus, the quantization error introduced at loop τ is propagated through all downstream sensitivity factors $\gamma_{\tau+1}, \ldots, \gamma_{T-1}$. Earlier errors pass through a longer chain of loop transformations and can therefore have a larger effect on the final state. The proposition follows. □

B Implementation and Calibration Details

B.1 Experimental and Calibration Setup

Calibration Data. We construct a calibration pool with equal token contributions from the Pile (Gao et al., 2020) and public reasoning datasets: OpenThoughts3-1.2M, AceReason-1.1-SFT,

Table 6: Transformation groups used as the minimum selection units.

Model	Candidate group	Weight side	Transform side
Ouro	QKV input	`q_proj, k_proj, v_proj`	`self_attn.ln_trans`
	Attention output	`o_proj`	`self_attn.o_trans`
	MLP gate/up	`gate_proj, up_proj`	`mlp.up_gate_trans`
	MLP down	`down_proj`	`mlp.down_trans`
	Key cache	—	`key-cache transform`
LoopFormer	QKV input	`attn.c_attn`	`attn.in_trans`
	Attention output	`attn.c_proj`	`attn.out_trans`
	MLP up	`mlp.c_fc`	`mlp.up_trans`
	MLP down	`mlp.c_proj`	`mlp.down_trans`
Parcae	QKV input	`attn.c_qkv`	`attn.in_trans`
	Attention output	`attn.c_proj`	`attn.out_trans`
	MLP up	`mlp.fc`	`mlp.up_trans`
	MLP down	`mlp.proj`	`mlp.down_trans`

OpenCodeReasoning, Llama-Nemotron-Post-Training-Dataset, and DeepWriting-20K. We sample 1,024 training examples and 128 validation examples and freeze backbone weights during calibration. All LoopLM PTQ baselines collect activations across the full recursive forward pass, and methods within each comparison use the same calibration data.

Quantization Settings. Static quantization uses groups of 32 for weights and activations, with loop-specific activation scales in LoopQ. Dynamic quantization uses per-output-channel weights and per-token activation groups of 128. KV4 uses asymmetric online quantization with groups of 32 in the static setting and up to 128 per attention head in the dynamic setting.

B.2 Loop-Aware Activation Scale Calibration

Each linear-layer activation quantizer maintains a scale for physical loop t and feature group g:

$$c_{t,\ell,g} = c^0_{t,\ell,g} \exp(\rho_{t,\ell,g}), \tag{9}$$

where $c^0_{t,\ell,g} > 0$ is the calibrated base scale and $\rho_{t,\ell,g}$ is a trainable log-scale correction initialized to zero. Starting from the FlatQuant MSE checkpoint, we refine the base scales by minimizing activation-reconstruction MSE under quantized forward passes. The base scales are then fixed while the corrections are optimized jointly with the other LoopQ parameters.

Scales are indexed by physical loop and feature group, independently of SLT selection. At export, $\exp(\rho_{t,\ell,g})$ is folded into the stored scale for retrieval at inference.

B.3 Loop-Dependent Transformation Selection

Candidate Groups and Parameterization. Selection operates on transformation groups. Linear-path groups couple a transform with its corresponding linear weights, while KV quantization adds key-cache transformation sites (Table 6). Selected linear-path groups require loop-dependent weight representations at deployment; key-cache sites act on the cache path. Following FlatQuant (Sun et al., 2025), we use Kronecker-factored transforms. Only the effective Cayley coordinates of the orthogonal factors vary across loops; singular values and optional channel scales remain shared, and backbone weights stay frozen. Let $\phi_{\ell j} \in \mathbb{R}^T$ collect coordinate j of group ℓ across loops.

Curvature Estimation and Sharing-Sensitivity Score. We estimate cross-loop curvature blocks with output probes whose covariance matches the predictive Fisher. For student probabilities $\boldsymbol{p}_i$, sample zero-mean, identity-covariance noise $\boldsymbol{\epsilon}_i$ and construct

$$\boldsymbol{z}_i = \sqrt{\boldsymbol{p}_i} \odot \boldsymbol{\epsilon}_i - \boldsymbol{p}_i \big(\sqrt{\boldsymbol{p}_i}^\top \boldsymbol{\epsilon}_i\big).$$

We average outer products of the resulting vector–Jacobian products. When sampling k of M microbatches uniformly without replacement, each VJP is scaled by $\sqrt{M/(Nk)}$, where N counts

valid target tokens in the full optimizer update. Teacher-selected support is fixed during differentiation, quantization uses straight-through derivatives, and estimated blocks are reused between refreshes. The approximation omits cross-coordinate and cross-module coupling.

Following the second-order compression principle of Optimal Brain Surgeon (Hassibi & Stork, 1992), we estimate the local cost of tying parameters across loops. With damped curvature, let

$$\boldsymbol{C}_{\ell j} = \widehat{\boldsymbol{G}}_{\ell j} + \nu_s \boldsymbol{I}, \qquad \boldsymbol{y}_{\ell j} = \boldsymbol{\phi}_{\ell j} - \boldsymbol{C}_{\ell j}^{-1} \boldsymbol{g}_{\ell j},$$

where $\boldsymbol{g}_{\ell j}$ is the clipped calibration gradient. The shared coordinate minimizing the local quadratic surrogate and the resulting sharing-sensitivity score are

$$a_{\ell j} = \frac{\mathbf{1}^\top \boldsymbol{C}_{\ell j} \boldsymbol{y}_{\ell j}}{\mathbf{1}^\top \boldsymbol{C}_{\ell j} \mathbf{1}}, \qquad S_\ell = \frac{1}{2} \sum_j \|\boldsymbol{y}_{\ell j} - a_{\ell j} \mathbf{1}\|^2_{\boldsymbol{C}_{\ell j}}, \tag{10}$$

where $\|\boldsymbol{v}\|^2_{\boldsymbol{C}} = \boldsymbol{v}^\top \boldsymbol{C} \boldsymbol{v}$. A larger S_ℓ indicates a greater predicted loss from enforcing sharing. We set $\nu_s = \max(\nu_{\min}, \|\boldsymbol{g}\|_2 / r_{\text{trial}})$, where $\boldsymbol{g}$ concatenates candidate rotation gradients, to bound the unconstrained trial step before sharing projection.

Selection Schedule. After warmup, we gradually reduce the loop-dependent group budget, retaining groups with the highest sharing-sensitivity scores. Groups remain eligible for reselection until the structure is fixed; calibration then continues with that configuration. For Ouro 1.4B-Thinking, the final budget is six of 120 candidate groups. Appendix C.1 compares the resulting selections for Ouro 1.4B and Ouro 2.6B.

B.4 Cross-Loop Transition Adapter Implementation

Let $\mathcal{N}(\cdot)$ denote group-wise RMS normalization. The low-rank correction is

$$\begin{aligned} \boldsymbol{R}_t = \; & \mathrm{RMS}(\Delta \widetilde{\boldsymbol{H}}_t) \odot \\ & \left[\mathrm{SiLU}\Big([\mathcal{N}(\widetilde{\boldsymbol{H}}_{t,0}), \mathcal{N}(\Delta \widetilde{\boldsymbol{H}}_t)] \boldsymbol{V} + \boldsymbol{b}_t \Big) \boldsymbol{U}^\top \right], \end{aligned} \tag{11}$$

where $\boldsymbol{V} \in \mathbb{R}^{2d \times r}$ and $\boldsymbol{U} \in \mathbb{R}^{d \times r}$ are shared across loop boundaries, while $\boldsymbol{b}_t$ is boundary-specific. The RMS factor is broadcast within each feature group, and fixed output scaling is absorbed into $\boldsymbol{U}$.

The input-conditioned tanh gate g_t uses a shared MLP with boundary-specific biases. Its inputs are the relative update RMS, cosine similarities between the update and the loop input and output, and the normalized loop index. The projection $\Pi^{\perp}_{\Delta \widetilde{\boldsymbol{H}}_t}$ is applied per token and feature group, with the squared update norm clamped below by ϵ for numerical stability.

B.5 Adaptive Trajectory Weighting

For each valid prediction token i, we define the predictive KL and relative boundary error as

$$k_i = \mathrm{KL}(\boldsymbol{Z}_{i,:} \| \widetilde{\boldsymbol{Z}}_{i,:}), \qquad e_{t,i} = \frac{\|(\widetilde{\boldsymbol{H}}_{t+1,0} - \boldsymbol{H}_{t+1,0})_{i,:}\|_2^2}{\|(\boldsymbol{H}_{t+1,0})_{i,:}\|_2^2 + 10^{-6}}. \tag{12}$$

The KL compares teacher and student probabilities obtained from the terminal logits and renormalized on the same teacher-selected support. The boundary error compares the post-adapter student state with the corresponding teacher state.

At each boundary, we normalize $e_{t,i}$ by the median ratio of relative boundary error to $\max(k_i, 0) + 10^{-8}$ over valid tokens in the initial calibration microbatches. This normalization remains fixed throughout training; $e_{t,i}$ below denotes the normalized error. The effective coefficient in Eq. (8) satisfies

$$\mu_t \|\widetilde{\boldsymbol{H}}_{t+1,0} - \boldsymbol{H}_{t+1,0}\|_F^2 = \frac{1}{n(T-1)} \sum_{i=1}^{n} \frac{k_i^+ e_{t,i}}{e_{t,i} + k_i^+}, \tag{13}$$

where $k_i^+ = \max(k_i, 0)$ removes small negative KL values caused by numerical error, and n counts valid prediction tokens. A summand is defined as zero when both errors vanish, and $\mu_t = 0$ when the boundary states match. Gradients flow through the adaptive weights. All loss reductions exclude padding and non-target prefixes.

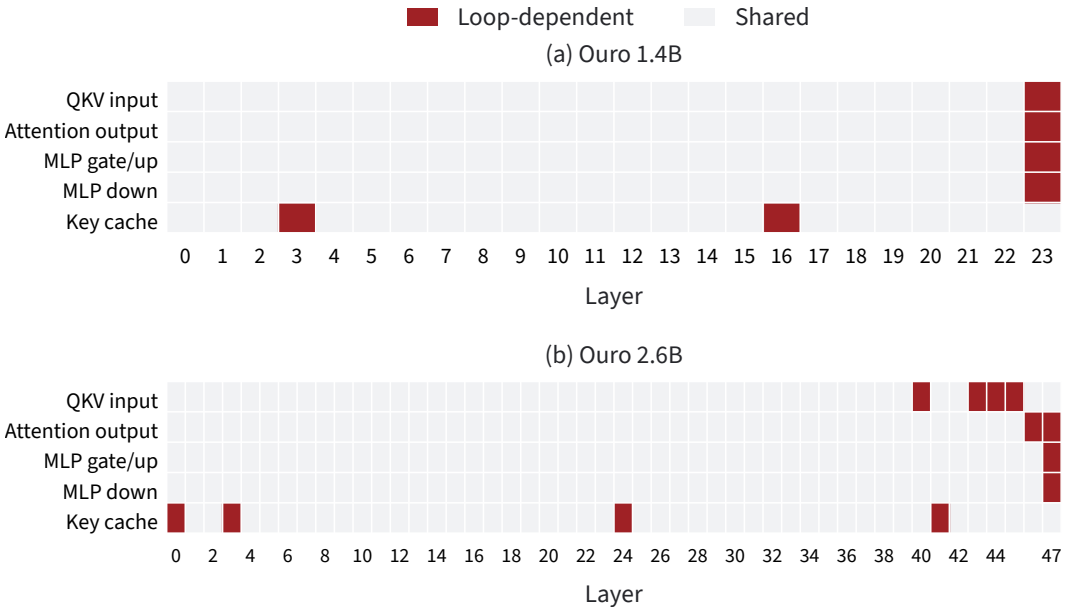


Figure 4: Final loop-dependent group selections for (a) Ouro 1.4B and (b) Ouro 2.6B. Red indicates independent transformations for the four loops.

C Additional Experiments and Analysis

C.1 Allocation of Loop-Dependent Transformation Groups

Figure 4 shows the groups selected for loop-dependent transformations in Ouro 1.4B and Ouro 2.6B. Each shared layer has five candidates: QKV input, attention output, MLP gate/up, MLP down, and key cache. Under a 5% group budget, the models select 6 of 120 and 12 of 240 candidates, respectively. Selection uses the sharing-sensitivity scores in Appendix B.3, which estimate the calibration cost of tying a group's transformations across loops.

Linear-path selections concentrate near the end of the shared stack. In Ouro 1.4B, all four selected linear-path groups are at the final layer, covering QKV input, attention output, and both MLP groups. In Ouro 2.6B, eight linear-path groups span layers 40–47: four QKV inputs, two attention outputs, and both MLP groups at the final layer. This concentration is consistent with the pronounced error spikes near loop transitions in Figure 2. Under recursive reuse, the final layer of one loop directly supplies the state processed by the first layer of the next, making these boundary layers particularly influential in how quantization errors enter and propagate through subsequent loops. Adapting late-layer transformations can therefore reduce the mismatch passed across loop boundaries.

Key-cache selections are more dispersed, appearing at layers 3 and 16 in Ouro 1.4B and layers 0, 3, 24, and 41 in Ouro 2.6B. They account for one-third of each model's selected groups. This difference suggests that useful selection sites depend on group type as well as depth; restricting selection to late layers would exclude the earlier key-cache sites.

C.2 Alternative Quantization Settings

Table 7 provides the task-level results summarized in Table 4 for INT4 and NVFP4 under dynamic per-token-per-group (PTG) quantization on Ouro 1.4B-Thinking. Both formats estimate activation ranges online. MR-GPTQ is designed specifically for NVFP4; its INT4 entries are therefore N/A.

INT4. Under dynamic INT4, FlatQuant is the strongest baseline at 0.5461 five-task mean accuracy, while LoopQ reaches 0.5842. LoopQ improves all five task accuracies over FlatQuant and reduces WikiText-2 word PPL from 18.84 to 16.63. These gains indicate that online range estimation alone does not eliminate the benefit of loop-aware adaptation. The quantization granularities described in

Table 7: Task-level results for INT4 and NVFP4 under dynamic PTG quantization on Ouro 1.4B-Thinking. HS, WG, LB, AC, and MM denote HellaSwag, WinoGrande, LAMBADA, ARC-C, and MMLU accuracy on a 0–1 scale. Mean averages these five tasks. WT2 denotes WikiText-2 word perplexity. Best values within each format are bolded.

Setting	Metric	RTN	MR-GPTQ	SmoothQuant	QuaRot	MixQuant	FlatQuant	LoopQ
INT4	HS ↑	0.2627	N/A	0.2650	0.4580	0.3215	0.6427	**0.6612**
	WG ↑	0.5004		0.4893	0.4996	0.4886	0.6125	**0.6219**
	LB ↑	0.0000		0.0000	0.2272	0.0749	0.4820	**0.5544**
	AC ↑	0.2526		0.2543	0.3541	0.2577	0.4735	**0.4974**
	MM ↑	0.2619		0.2602	0.3222	0.2449	0.5197	**0.5862**
	Mean ↑	0.2555		0.2538	0.3722	0.2775	0.5461	**0.5842**
	WT2 PPL ↓	$> 10^3$		$> 10^3$	62.89	225.54	18.84	**16.63**
NVFP4	HS ↑	0.6656	0.6714	0.6809	0.6874	0.6842	0.6703	**0.6880**
	WG ↑	0.6014	0.6243	0.6196	0.6281	**0.6361**	0.6267	0.6350
	LB ↑	0.5317	0.5721	0.5519	0.5732	0.5844	0.5614	**0.5851**
	AC ↑	0.4915	0.5000	0.4787	0.5139	**0.5316**	0.5111	0.5251
	MM ↑	0.5907	0.6133	0.6037	0.6238	0.6215	0.6331	**0.6384**
	Mean ↑	0.5762	0.5962	0.5870	0.6053	0.6116	0.6005	**0.6143**
	WT2 PPL ↓	16.3457	14.9685	15.4972	15.3845	15.1600	15.2118	**14.8861**

Appendix B.1 also differ from the static setting, so comparisons with static results do not isolate the effect of online range estimation. Runtime costs are reported in Appendix C.4.

NVFP4. LoopQ extends effectively to NVFP4, a recent low-precision floating-point format. With a per-loop global scale, LoopQ attains the highest five-task mean accuracy of 0.6143 and the lowest WikiText-2 word PPL of 14.8861. Its mean accuracy exceeds MixQuant's 0.6116 by 0.28 percentage points and FlatQuant's 0.6005 by 1.38 points. LoopQ leads on HellaSwag, LAMBADA, and MMLU, while MixQuant leads on WinoGrande and ARC-C. These results show that loop-aware adaptation remains beneficial with the newer floating-point format and online activation scaling.

C.3 Early Exit and Loop-Depth Extrapolation

We evaluate early exit on Ouro 1.4B and execution beyond the calibrated loop depth on Parcae. For early exit, we sweep $q \in \{0.125, 0.250, 0.375, 0.500, 0.625, 0.750, 0.875\}$ with a maximum of four loops, using the W4A4KV16 checkpoint from Table 1. Figure 5(a,b) reports five-task mean accuracy and average executed iterations. Without early exit, LoopQ achieves 0.5909 accuracy at four iterations.

Following Ouro's deterministic Q-exit criterion (Zhu et al., 2025), let $\lambda_t(x)$ denote the learned halting probability at loop t. The cumulative exit probability is

$$\mathrm{CDF}(t \mid x) = \sum_{i=1}^{t} \lambda_i(x) \prod_{j=1}^{i-1} (1 - \lambda_j(x)). \tag{14}$$

Inference stops at the first loop with $\mathrm{CDF}(t \mid x) \geq q$, or after four loops if the threshold is not reached. Smaller thresholds permit earlier exits.

LoopQ retains a controllable accuracy–computation trade-off. At $q = 0.250$, it averages 3.035 iterations, 24.1% fewer than full depth, with an accuracy decrease of 1.32 percentage points. At $q = 0.500$, it averages 3.590 iterations, a 10.3% reduction, with a decrease of 0.52 percentage points. Higher thresholds approach four iterations as accuracy plateaus.

For loop-depth extrapolation, we calibrate Parcae over eight loops and evaluate it under W4A4 at 10, 12, and 16 loops. Each additional loop uses the geometric mean of activation scales from the last four calibrated loops. LoopQ's WikiText-2 PPL decreases through eight loops, then remains between 33.57 and 33.82 at the evaluated extrapolation depths, compared with 32.85 at eight loops. The full-precision reference ranges from 25.68 to 26.35 at the extrapolation depths, compared with

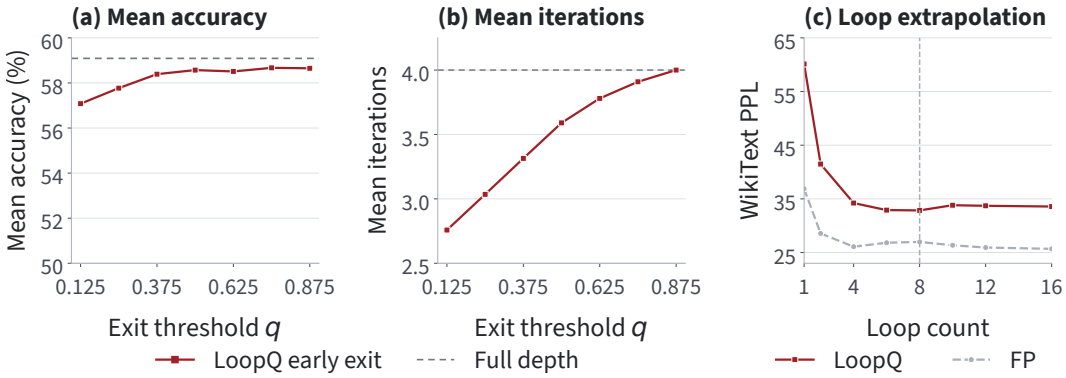


Figure 5: Early exit and loop-depth extrapolation with LoopQ. On Ouro 1.4B, (a) five-task mean accuracy and (b) average executed iterations vary with the exit threshold. (c) WikiText-2 PPL on Parcae as loop depth increases; the vertical dashed line marks the eight-loop calibration horizon.

Table 8: Parameter overhead and theoretical deployment memory under static W4A4.

Metric	**Ouro 1.4B**	**LoopFormer 3×8**	**Parcae 370M**
Parameter counts and metadata			
Total values	1.48B	0.29B	0.40B
Shared metadata	41.13M	3.86M	5.92M
Shared ratio	2.77%	1.36%	1.49%
Loop-dependent metadata	8.93M	3.25M	2.83M
Loop-dependent ratio	0.60%	1.14%	0.71%
Deployment memory			
BF16	2.67 GiB	0.52 GiB	0.72 GiB
LoopQ W4A4	1.13 GiB	0.34 GiB	0.34 GiB
Relative reduction	−57.68%	−36.01%	−52.99%

26.98 at eight loops. Thus, LoopQ supports early exit with a measurable accuracy–computation trade-off and maintains stable perplexity beyond its calibrated depth.

C.4 Memory Footprint and Inference Throughput

Model-State Accounting. Table 8 reports parameter overhead and theoretical deployment memory under the static W4A4 setting in Appendix B.1. Shared and loop-dependent metadata are measured relative to total reconstructible values, counting backbone weights once and excluding derived weight copies. Deployment memory instead includes expanded loop-specific weight copies, INT4 codes, FP32 quantization scales, retained state, and embedding codes and scales; it excludes KV caches and temporary workspaces. LoopFormer and Parcae use per-row INT8 embeddings and tied output heads with FP32 scales, while Ouro retains BF16 embeddings. Across the three backbones, loop-dependent metadata accounts for 0.60%–1.14% of total values, and LoopQ reduces theoretical deployment memory by 36.01%–57.68% relative to BF16.

Accuracy–Memory Comparison. Table 3 and Figure 6 compare Ouro 1.4B with Qwen3 dense models using five-task mean accuracy. The unrolled Ouro reference applies FlatQuant to separate copies of the loop instances. The Static SOTA point combines the best task accuracies among the static W4A4 baselines for Ouro 1.4B in Table 1 and uses FlatQuant's deployment memory as its representative footprint; it is a metric-wise reference rather than one deployable checkpoint. Quantized Qwen3 checkpoints use NVFP4 weights and activations. Their memory is measured after loading, whereas Ouro memory is estimated from deployment state.

LoopQ exceeds quantized Qwen3-1.7B by 9.82 percentage points at a smaller reported footprint (1.13 versus 1.92 GiB). Relative to quantized Qwen3-4B, it is 1.04 percentage points lower while

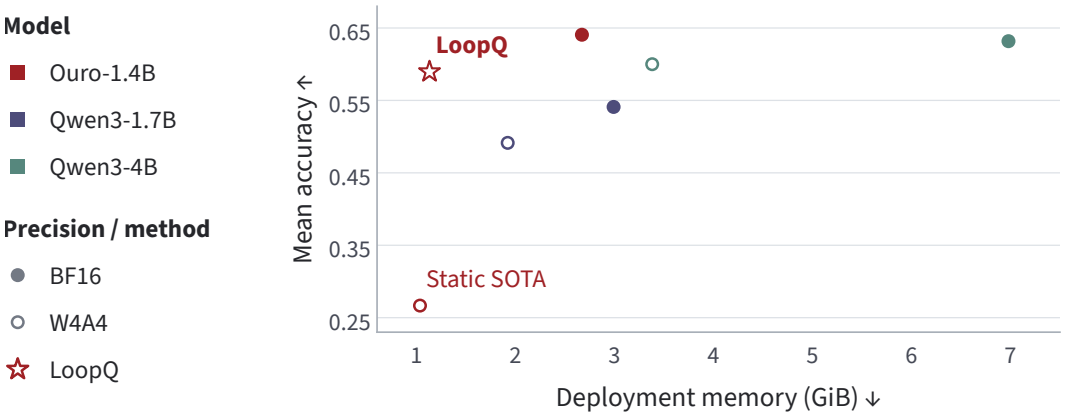


Figure 6: Five-task mean accuracy versus memory footprint across models and precisions.

Table 9: Decode throughput and component time shares for packed W4A4KV4 on Ouro 1.4B-Thinking with RTX 5090.

(a) Decode throughput (tokens/s)

Input tokens	Throughput		Decode speedup
	LoopQ	BF16	
512	1,040.1	836.1	1.24×
1,024	844.5	611.2	1.38×
2,048	567.4	387.3	1.47×
4,096	337.9	205.6	1.64×
8,192	180.1	84.7	2.13×

(b) GPU execution time shares

Input tokens	Prefill		Decode	
	Transforms	CTA	Transforms	CTA
512	11.57%	0.46%	13.46%	0.53%
1,024	11.45%	0.45%	12.87%	0.53%
2,048	11.05%	0.43%	12.42%	0.54%
4,096	10.73%	0.42%	11.85%	0.71%
8,192	9.65%	0.40%	11.74%	0.70%

using approximately one-third of the reported memory. Unrolled Ouro gains 2.47 percentage points over LoopQ but requires an estimated 2.83 GiB, slightly more than Ouro BF16.

Packed Inference Protocol. We evaluate Ouro 1.4B-Thinking on an NVIDIA GeForce RTX 5090 using packed W4A4KV4 with static activation scales and FP32 KV scales and offsets. The BF16 baseline combines vLLM scheduling, FlashAttention 2, and full decode CUDA Graph replay. Both runtimes use vLLM 0.27.1/V1; LoopQ uses custom attention. Inductor compilation, prefix caching, chunked prefill, and asynchronous scheduling are disabled, with a maximum sequence length and prefill token budget of 9,216.

Prompts contain 512–8,192 tokens, and each request generates exactly 1,024 tokens with greedy decoding and no EOS stopping. All requests are enqueued before scheduling. Decode throughput is measured from the point at which every request has produced its first token until all requests finish, including CPU scheduling and the completion tail. Measurements use a local offline engine without HTTP overhead or KV-cache preemption. All BF16 and W4A4 results are medians of five runs. Concurrency is selected separately for each runtime, so speedups represent achieved throughput rather than matched-concurrency execution.

Throughput and Component Overhead. Table 9 and Figure 3c show decode speedups of 1.24–2.13× over BF16 across the evaluated prompt lengths. Transformations account for 9.65–11.57% of prefill and 11.74–13.46% of decode GPU execution time, while CTA accounts for less than 1% in both phases. Dynamic PTG performs online range reduction before A4 packing, whereas LAS retrieves an offline-calibrated scale for the current loop. In a separate RTX 5090 packed-runtime comparison, replacing LAS with Dynamic PTG increases prefill latency by 30.02% and decode latency by 2.15%.