LOOPQUANT: ACCURATE AND EFFICIENT POST-TRAINING QUANTIZATION FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Language Models (LoopLMs) improve parameter efficiency by recursively reusing shared Transformer modules, yet their repeated computation over evolving recurrent states poses new challenges for low-bit deployment. Existing post-training quantization (PTQ) methods, largely designed for feed-forward Transformers, fail to account for cross-loop activation dynamics, heterogeneous loop sensitivity, and recursive error propagation, leading to substantial performance degradation at low precision. To address these challenges, we propose LoopQuant, a loop-aware PTQ framework for accurate and efficient low-bit quantization of LoopLMs. LoopQuant is built on three core components: 1) Cross-Loop Distribution Canonicalization (CLDC) learns a shared quantization-friendly representation across recurrent states while preserving parameter sharing; 2) Interaction-Aware Loop-wise Precision Allocation (ILPA) allocates activation precision under a fixed bit budget by modeling task-level loop sensitivity and cross-loop interactions; and 3) Loop-Conditioned Task Calibration (LCTC) further aligns quantized inference with the low-bit recursive trajectory through lightweight task-space calibration. Extensive experiments across LoopLM architectures demonstrate that LoopQuant consistently outperforms existing low-bit PTQ methods. On Ouro-2.6B, LoopQuant under W4A4 surpasses all W4A8 baselines in performance, while reducing model memory by 65.3% with only 3.4% additional storage overhead over naive INT4 and achieving a 1.7$\times$ decoding speedup over BF16. These results demonstrate a strong balance among accuracy, compression, and inference efficiency, providing a practical and deployment-friendly solution for low-bit LoopLM inference. The code is available at LoopQuant.

1 INTRODUCTION

Transformer-based large language models (LLMs) have demonstrated strong capabilities in language understanding, code generation, and complex reasoning. However, these gains come with larger model sizes, higher inference costs, and increased memory access, making deployment challenging in memory- and bandwidth-constrained environments. Rather than simply scaling model size, Looped Language Models (LoopLMs) recursively reuse the same Transformer modules across multiple loops (Zhu et al., 2026), increasing effective computation depth under a nearly fixed parameter budget. This parameter sharing reduces storage while maintaining strong representation and reasoning capabilities, making LoopLMs attractive for constrained deployment. However, shared modules are repeatedly executed, with weights and intermediate activations frequently accessed across loops. Low-bit quantization of both weights and activations is therefore important for reducing computation and data movement costs. Achieving efficient low-bit quantization while preserving recursive reasoning capabilities remains a key challenge for practical LoopLM deployment.

Post-training quantization (PTQ) enables efficient LLM deployment by reducing storage and computation costs without expensive retraining. Existing methods employ weight reconstruction, activation clipping, channel scaling, rotation transformations (Xiao et al., 2023; Ashkboos et al., 2024), and mixed-precision allocation (Saxena et al., 2024) to enable low-bit weight and activation quantization for conventional Transformers. However, most target feed-forward Transformers, where layers are executed sequentially and quantization is calibrated from relatively stable layer-wise activation

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.

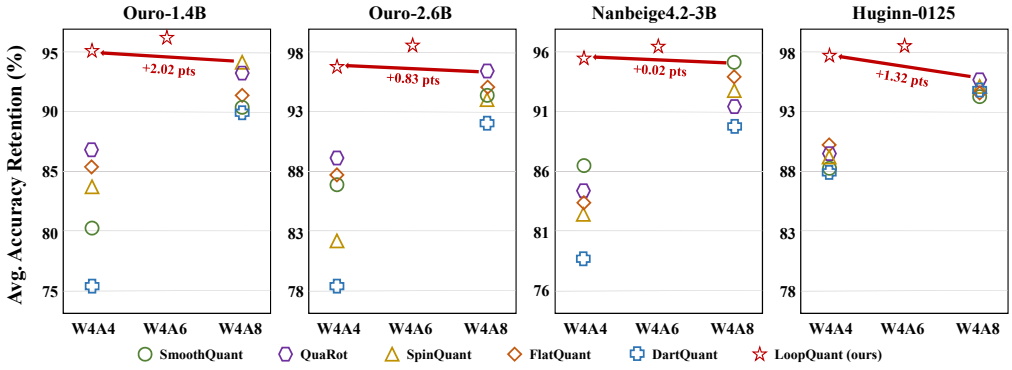


Figure 1: Accuracy retention of PTQ methods on four LoopLMs under different W/A precisions. LoopQuant consistently preserves higher accuracy, with W4A4 surpassing baselines at W4A8.

statistics, typically with local reconstruction objectives. LoopLMs break this assumption. The same Transformer modules are repeatedly invoked across loops, while evolving recurrent states expose them to different activation distributions. Quantization perturbations from earlier loops also propagate through recursive computation, altering subsequent hidden states and quantization behavior. LoopLM quantization must therefore account for cross-loop activation dynamics, loop-dependent quantization sensitivity, and recursive error propagation. As shown in Fig. 1, directly applying existing PTQ methods can cause substantial performance degradation at low bit-widths.

To further understand the core challenges of low-bit PTQ for LoopLMs, we identify three key issues. ❶ **Activation distributions vary substantially across loops.** As hidden states evolve, the same shared module receives different activation distributions across loops, with varying dynamic ranges, outlier magnitudes, and outlier channels. Static quantization parameters calibrated on a single distribution may fail to accommodate all recurrent states, while separate configurations for each loop introduce additional storage and deployment overhead. ❷ **Loops exhibit heterogeneous importance and quantization sensitivity.** Although all loops share the same Transformer parameters, they operate at different recurrent stages and contribute differently to the final prediction. Quantization errors introduced in earlier loops can propagate through later computation. Uniform activation precision may therefore waste the bit budget, while allocation based only on local quantization errors cannot reflect the true impact of each loop on the final output. ❸ **Local quantization objectives are misaligned with recursive task behavior.** Even when the local quantization error is small at one loop, low-bit computation can shift the hidden state and alter the input distributions of subsequent loops. Such deviations can accumulate along the recursive trajectory and cause substantial output errors. Accurate LoopLM quantization therefore requires joint modeling of cross-loop activation dynamics and loop sensitivity, followed by calibration along the actual low-bit recursive trajectory.

To this end, we propose **LoopQuant**, a loop-aware PTQ framework for accurate low-bit LoopLM inference. ❶ **Cross-Loop Distribution Canonicalization (CLDC)** learns a shared quantization-friendly representation across recurrent states through orthogonal-scaling transformation, robust cross-loop synchronization, and group-aware channel reordering, while preserving parameter sharing. ❷ **Interaction-Aware Loop-wise Precision Allocation (ILPA)** allocates A4/A8 activation precision under a fixed bit budget by estimating task-level loop sensitivity and modeling cross-loop interactions induced by recursive error propagation. ❸ **Loop-Conditioned Task Calibration (LCTC)** further calibrates quantization behavior along the actual low-bit recursive trajectory through task-space defect transport and Fisher-weighted trajectory calibration, while optimizing only lightweight loop-conditioned clipping, gain, and bias parameters. Together, they act on activation distributions, precision budgets, and recursive quantization behavior to enable accurate and deployment-friendly low-bit inference for LoopLMs. Our key contributions are summarized as follows:

- We identify three key challenges in low-bit PTQ for LoopLMs: cross-loop activation dynamics, heterogeneous loop sensitivity, and the mismatch between local quantization objectives and recursive task behavior.
- We propose **LoopQuant**, a loop-aware PTQ framework combining **CLDC** for cross-loop distribution canonicalization, **ILPA** for interaction-aware loop-wise precision allocation, and **LCTC** for task-aligned calibration along the low-bit recursive trajectory.

- Extensive experiments show that LoopQuant achieves a superior accuracy–efficiency trade-off across LoopLMs, with W4A4 outperforming existing PTQ baselines at W4A8. To facilitate efficient LoopLM deployment, we will release our implementation, which, to our knowledge, is the first open-source quantization framework for LoopLMs.

2 Related Work

2.1 Post-Training Quantization for Large Language Models

Post-training quantization (PTQ) compresses pretrained models without expensive end-to-end retraining, reducing the storage, memory access, and inference costs of LLMs. For low-bit weight–activation quantization, activation outliers remain a major obstacle to accurate quantization. SmoothQuant (Xiao et al., 2023) applies channel-wise scaling to shift quantization difficulty from activations to weights, while QuaRot (Ashkboos et al., 2024) uses orthogonal rotations to redistribute hidden-state outliers while preserving full-precision outputs. SpinQuant (Liu et al., 2025) further learns quantization-friendly rotations. More recent methods strengthen these transformations: FlatQuant (Sun et al., 2024) learns flexible affine transformations for weights and activations, while DartQuant (Shao et al., 2026) performs distribution-aware rotation calibration with efficient orthogonal optimization. Despite strong performance on standard Transformers, these methods typically calibrate each layer from local activation distributions and do not explicitly model evolving cross-loop activation dynamics or loop-dependent sensitivity of recursively shared modules.

2.2 Looped Language Models

Looped Language Models (LoopLMs) increase effective computation depth by recursively reusing shared Transformer modules, decoupling model capacity from parameter count and enabling parameter-efficient scaling (Dehghani et al., 2018). This paradigm has been adopted by modern LLMs. Relaxed Recursive Transformers (Bae et al., 2025) reuse shared Transformer layers with lightweight adapters to mitigate the expressiveness limits of strict parameter sharing. Huginn (Geiping et al., 2026) adopts a recurrent-depth architecture and scales test-time latent computation through recursive applications of shared modules. Ouro (Zhu et al., 2026) extends LoopLMs to large-scale pretraining, iteratively refining latent representations to improve reasoning under a fixed parameter budget. Recently, Nanbeige4.2-3B (Lab et al., 2026) employs a Looped Transformer trained from scratch, reusing a shared layer stack to increase effective depth and showing strong performance on reasoning, coding, and agentic tasks. Low-bit LoopLM quantization has also received attention. LoopQ (Fang et al., 2026) provides an initial study of cross-loop distribution shifts and recursive quantization errors in Recursive Transformers. Nevertheless, evolving cross-loop activation dynamics, heterogeneous loop-wise quantization sensitivity, and error propagation along recursive trajectories remain distinct challenges for LoopLM quantization and motivate our work.

3 Preliminaries

Looped Language Models Looped Language Models (LoopLMs) increase effective computation depth by recursively reusing shared Transformer layers. Given an input sequence $\mathbf{X}$, let $\mathcal{F}_\theta$ denote the shared Transformer module with L layers, where $\theta = \{\theta_1, \ldots, \theta_L\}$. The input is embedded as the initial hidden state $\mathbf{H}^{(0)}$, after which the shared module is recursively applied for T loops:

$$\mathbf{H}^{(t)} = \mathcal{F}_\theta\left(\mathbf{H}^{(t-1)}\right), \qquad t = 1, \ldots, T. \tag{1}$$

The final output is obtained from the last-loop hidden state:

$$\mathbf{Y} = \mathrm{LMHead}\left(\mathbf{H}^{(T)}\right). \tag{2}$$

For layer-wise notation, let $\mathbf{H}_l^{(t)}$ denote the output of layer l at loop t. The computation becomes

$$\mathbf{H}_l^{(t)} = \mathcal{F}_l\left(\mathbf{H}_{l-1}^{(t)}; \theta_l\right), \qquad \mathbf{H}_0^{(t)} = \mathbf{H}_L^{(t-1)}. \tag{3}$$

Parameters θ are shared across loops, while hidden states evolve recursively. Thus, L shared layers executed for T loops provide an effective depth of approximately LT while storing only L layers.

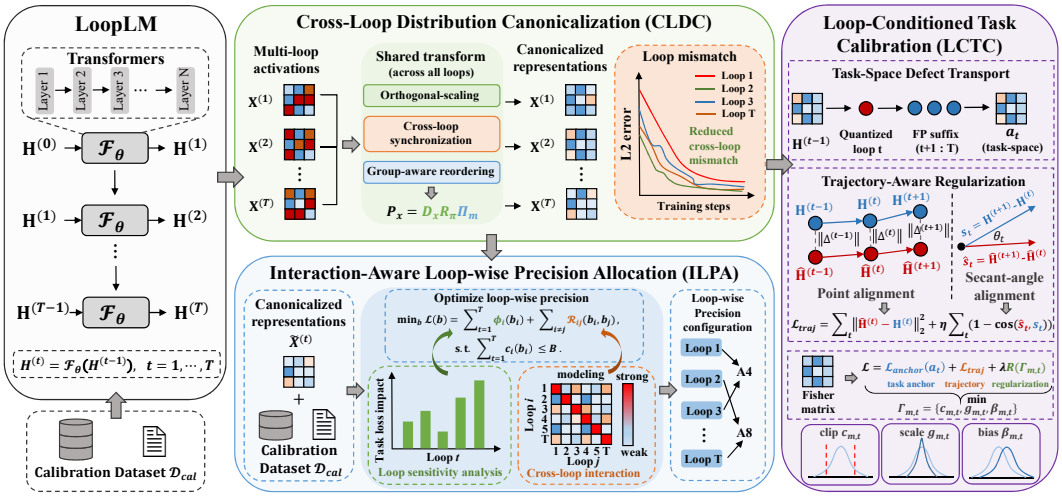


Figure 2: Overview of LoopQuant. LoopQuant consists of three stages: **CLDC** canonicalizes cross-loop activation distributions, **ILPA** allocates activation precision under a fixed bit budget, and **LCTC** aligns quantized models with recursive task behavior.

Despite sharing identical parameters, the layers receive activations from different recurrent states, distinguishing LoopLMs from standard feed-forward Transformers.

4 Method

Overview. In this section, we first formulate low-bit quantization under recursive computation (Sec. 4.1), decomposing task degradation into loop-dependent local errors, cross-loop sensitivity and interactions, and trajectory-dependent errors. Building on this formulation, **LoopQuant** consists of three stages (Fig. 2): CLDC learns a shared quantization-friendly representation across recursive states (Sec. 4.2); ILPA allocates activation precision by modeling loop sensitivity and cross-loop interactions under a fixed bit budget (Sec. 4.3); and LCTC calibrates quantization behavior along the low-bit recursive trajectory (Sec. 4.4). Pseudocode is provided in Appendix A.1.

4.1 Quantization under Recursive Computation

Following the LoopLM formulation in Sec. 3, we characterize low-bit quantization errors through recurrent state perturbation and the final task objective. For loop t, let $\mathcal{F}_\theta$ and $\widehat{\mathcal{F}}_\theta^{(t)}$ denote the full-precision and quantized mappings. Given an input state $\mathbf{H}$, define the quantization residual as

$$\mathbf{r}_t(\mathbf{H}) = \widehat{\mathcal{F}}_\theta^{(t)}(\mathbf{H}) - \mathcal{F}_\theta(\mathbf{H}). \tag{4}$$

During quantized inference, loop t receives $\widehat{\mathbf{H}}^{(t-1)}$, already affected by preceding quantization errors. The resulting perturbation is

$$\mathbf{u}_t = \mathbf{r}_t\left(\widehat{\mathbf{H}}^{(t-1)}\right) = \underbrace{\mathbf{r}_t\left(\mathbf{H}^{(t-1)}\right)}_{\boldsymbol{\epsilon}_t^{\mathrm{loc}}} + \underbrace{\mathbf{r}_t\left(\widehat{\mathbf{H}}^{(t-1)}\right) - \mathbf{r}_t\left(\mathbf{H}^{(t-1)}\right)}_{\boldsymbol{\epsilon}_t^{\mathrm{traj}}}. \tag{5}$$

Here, $\boldsymbol{\epsilon}_t^{\mathrm{loc}}$ denotes the loop-dependent local error, while $\boldsymbol{\epsilon}_t^{\mathrm{traj}}$ captures the additional mismatch caused by inherited state drift. Thus,

$$\widehat{\mathbf{H}}^{(t)} = \mathcal{F}_\theta\left(\widehat{\mathbf{H}}^{(t-1)}\right) + \mathbf{u}_t, \qquad t = 1, \ldots, T. \tag{6}$$

Viewing $\{\mathbf{u}_t\}_{t=1}^T$ as perturbations injected along the recursive trajectory, we write the task loss as $\mathcal{L} = \mathcal{L}(\mathbf{u}_1, \ldots, \mathbf{u}_T)$. A second-order expansion around the full-precision model gives

$$\Delta\mathcal{L} \approx \sum_{t=1}^{T} \mathbf{g}_t^\top \mathbf{u}_t + \frac{1}{2}\sum_{i=1}^{T}\sum_{j=1}^{T} \mathbf{u}_i^\top \mathbf{C}_{ij}\mathbf{u}_j, \quad \mathbf{g}_t = \left.\frac{\partial\mathcal{L}}{\partial\mathbf{u}_t}\right|_{\mathbf{u}=0}, \quad \mathbf{C}_{ij} = \left.\frac{\partial^2\mathcal{L}}{\partial\mathbf{u}_i\partial\mathbf{u}_j}\right|_{\mathbf{u}=0}. \tag{7}$$

Here, $\mathbf{g}_t$ measures task sensitivity to perturbations at loop t, while $\mathbf{C}_{ij}$ captures cross-loop interactions. Substituting $\mathbf{u}_t = \boldsymbol{\epsilon}_t^{\text{loc}} + \boldsymbol{\epsilon}_t^{\text{traj}}$ yields

$$\Delta\mathcal{L} \approx \sum_{t=1}^{T} \underbrace{\mathbf{g}_t^\top}_{\text{ILPA}} \left(\underbrace{\boldsymbol{\epsilon}_t^{\text{loc}}}_{\text{CLDC}} + \underbrace{\boldsymbol{\epsilon}_t^{\text{traj}}}_{\text{LCTC}} \right) + \frac{1}{2} \sum_{i=1}^{T} \sum_{j=1}^{T} \left(\boldsymbol{\epsilon}_i^{\text{loc}} + \boldsymbol{\epsilon}_i^{\text{traj}} \right)^\top \underbrace{\mathbf{C}_{ij}}_{\text{ILPA}} \left(\boldsymbol{\epsilon}_j^{\text{loc}} + \boldsymbol{\epsilon}_j^{\text{traj}} \right). \tag{8}$$

This decomposition links loop-dependent local error to CLDC, task sensitivity and cross-loop interaction to ILPA, and trajectory-dependent error to LCTC.

4.2 Cross-Loop Distribution Canonicalization (CLDC)

Motivation. The shared module in LoopLMs processes recurrent states, producing loop-dependent activation ranges, outliers, and quantization-sensitive directions. Loop-specific transformations can adapt to individual states but introduce storage and cannot be fused into shared weights.

We propose **Cross-Loop Distribution Canonicalization (CLDC)**, which learns a shared quantization-friendly geometry across recurrent states. For a shared linear module with weight $\mathbf{W}_m$ and loop-specific activation $\mathbf{X}_m^{(t)}$, CLDC applies the transformation $\mathbf{P}_m$ to every loop. We parameterize its continuous component as $\mathbf{P}_m^{\text{cont}} = \mathbf{D}_m\mathbf{R}_m$, where $\mathbf{D}_m = \text{Diag}(\mathbf{s}_m)$ performs channel scaling and $\mathbf{R}_m\mathbf{R}_m^\top = \mathbf{I}$ is orthogonal. Both are optimized using calibration states from all loops, with the parameterization of $\mathbf{R}_m$ in Appendix A.2.1. The transformed activations and weights are

$$\widetilde{\mathbf{X}}_m^{(t)} = \mathbf{X}_m^{(t)}\mathbf{D}_m\mathbf{R}_m, \qquad \widetilde{\mathbf{W}}_m = \mathbf{W}_m\mathbf{D}_m^{-1}\mathbf{R}_m. \tag{9}$$

Orthogonality preserves the full-precision computation, i.e., $\widetilde{\mathbf{X}}_m^{(t)}\widetilde{\mathbf{W}}_m^\top = \mathbf{X}_m^{(t)}\mathbf{W}_m^\top$ for all t. Under the target quantization configuration, we define the normalized reconstruction error as

$$\mathcal{E}_m^{(t)} = \frac{\left\| Q_A(\widetilde{\mathbf{X}}_m^{(t)}) Q_W(\widetilde{\mathbf{W}}_m)^\top - \mathbf{X}_m^{(t)}\mathbf{W}_m^\top \right\|_F^2}{\left\| \mathbf{X}_m^{(t)}\mathbf{W}_m^\top \right\|_F^2}. \tag{10}$$

Because severe distortion at any recurrent position can propagate forward, CLDC minimizes a smooth worst-loop objective:

$$\mathcal{L}_{\text{rob}} = \tau \log \left(\frac{1}{T} \sum_{t=1}^{T} \exp\left(\frac{\mathcal{E}_m^{(t)}}{\tau} \right) \right). \tag{11}$$

Reconstruction alone does not ensure consistent group-wise quantization because outlier channels may shift across loops. CLDC therefore synchronizes cross-loop anomaly profiles. Let $a_{m,c}^{(t)} = \text{Quantile}_q(|\widetilde{\mathbf{X}}_{m,:,c}^{(t)}|)$, $p_{m,c}^{(t)} = a_{m,c}^{(t)} / \sum_k a_{m,k}^{(t)}$, and $\overline{\mathbf{p}}_m = \frac{1}{T}\sum_t \mathbf{p}_m^{(t)}$. The synchronization loss is

$$\mathcal{L}_{\text{sync}} = \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{p}_m^{(t)} - \overline{\mathbf{p}}_m\|_2^2. \tag{12}$$

To avoid excessive scaling, we further use $\mathcal{L}_{\text{reg}} = \frac{1}{d_i}\|\mathbf{s}_m - \mathbf{1}\|_2^2$, giving

$$(\mathbf{D}_m^\star, \mathbf{R}_m^\star) = \arg\min \left[\mathcal{L}_{\text{rob}} + \lambda_{\text{sync}}\mathcal{L}_{\text{sync}} + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}} \right]. \tag{13}$$

Finally, CLDC learns a hard channel permutation $\mathbf{\Pi}_m^\star$ to improve quantization compatibility by grouping channels with similar cross-loop dynamic ranges; the grouping objective and optimization are described in Appendix A.2.3. The final transformation is $\mathbf{P}_m^\star = \mathbf{D}_m^\star\mathbf{R}_m^\star\mathbf{\Pi}_m^\star$. The resulting representation is frozen for subsequent precision allocation and task calibration.

4.3 Interaction-Aware Loop-wise Precision Allocation (ILPA)

Motivation. After CLDC, loops may still require different activation precisions because their perturbations have unequal task impact and interact through recursive propagation. Uniform precision ignores this heterogeneity, while local reconstruction errors cannot capture cross-loop coupling.

Inspired by cooperative optimization for mixed-precision quantization (Zhao et al., 2025), we propose **Interaction-Aware Loop-wise Precision Allocation (ILPA)**, which assigns loop-wise activation precision from complete recursive trajectories under a fixed bit budget. Let $\mathcal{T} = \{1, \dots, T\}$ and $\mathcal{S} \subseteq \mathcal{T}$ denote loops using low precision b_{L}, while the others use b_{H}. For each configuration, ILPA keeps weight quantization and model parameters fixed and evaluates the token-level NLL:

$$v(\mathcal{S}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\mathrm{cal}}}\left[-\frac{1}{|\mathbf{x}|}\sum_{n}\log p_{\mathcal{S}}(x_{n+1}|x_{\leq n})\right]. \tag{14}$$

Since $v(\mathcal{S})$ is evaluated over the complete low-bit trajectory, it captures loop-specific functionality, error propagation, and joint precision effects. To estimate loop sensitivity, ILPA aggregates context-dependent marginal degradation following the Shapley principle (Shapley et al., 1953):

$$\phi_t = \frac{1}{T!}\sum_{\pi\in\mathfrak{S}_T}\Delta_t^{(\pi)}. \tag{15}$$

The resulting ϕ_t provides a discrete task-level estimate of the first-order sensitivity induced by reducing the precision of loop t; the marginal construction is provided in Appendix A.2.4. ILPA further models pairwise recursive interaction:

$$\Gamma_{ij} = \frac{1}{T!}\sum_{\pi\in\mathfrak{S}_T}\left(\Delta_{ij}^{(\pi)} - \Delta_i^{(\pi)} - \Delta_j^{(\pi)}\right). \tag{16}$$

Γ_{ij} indicates stronger-than-additive degradation when loops i and j are jointly assigned low precision. We normalize Γ_{ij} across precision contexts to obtain $\widetilde{\Gamma}_{ij}$, which captures the discrete task-level interaction effect induced by joint precision reduction; details are given in Appendix A.2.5. Given target activation bit-width $\overline{b}$, ILPA minimizes a discrete surrogate objective:

$$\mathcal{J}(\mathbf{q}) = \sum_{t=1}^{T} q_t\phi_t + \lambda\sum_{i<j} q_i q_j\widetilde{\Gamma}_{ij}, \tag{17}$$

where $q_t = 1$ indicates low precision. The first term captures individual loop sensitivity, while the second accounts for recursive interaction. For small T, ILPA solves the constrained assignment by exact enumeration; for larger models, a greedy solver selects precision reductions with the smallest marginal increase in $\mathcal{J}$ until the bit budget is met. Solver and complexity details are provided in Appendix A.2.6. The resulting assignment remains fixed during inference.

4.4 Loop-Conditioned Task Calibration (LCTC)

Motivation. After CLDC and ILPA, a loop with small local quantization error may behave differently on states shifted by preceding low-bit computation, producing trajectory-dependent error $\boldsymbol{\epsilon}_t^{\mathrm{traj}}$. Local reconstruction alone therefore cannot ensure robust behavior along the recursive trajectory.

We propose **Loop-Conditioned Task Calibration (LCTC)**, which calibrates loop-dependent quantization responses while keeping the quantization structure and precision assignment fixed. For module m at loop t, LCTC introduces static parameters $\boldsymbol{\Gamma}_{m,t} = \{\mathbf{c}_{m,t}, \mathbf{g}_{m,t}, \boldsymbol{\beta}_{m,t}\}$, controlling group-wise clipping, gain, and bias. Given fixed quantized weight $\widehat{\mathbf{W}}_m$, the calibrated mapping is

$$\widehat{\mathbf{y}}_{m,t} = \mathbf{g}_{m,t}\odot\left[\widehat{\mathbf{W}}_m Q_{b_t}(\mathbf{x}_{m,t};\mathbf{c}_{m,t}) + \mathbf{b}_m\right] + \boldsymbol{\sigma}_{m,t}\odot\boldsymbol{\beta}_{m,t}, \tag{18}$$

where $\boldsymbol{\sigma}_{m,t}$ denotes the fixed group-wise output normalization statistic. These parameters depend only on module and loop index and are stored statically, requiring no token-wise history or online optimization, which keeps deployment lightweight without extra dynamic state. To align quantization defects with task behavior, let $\mathcal{S}_{t+1}(\cdot)$ map the output of loop t through the remaining computation to the final logits. We define the task-space defect as

$$\mathbf{d}_t(\mathbf{h};\boldsymbol{\Gamma}) = \mathcal{S}_{t+1}\left(\widehat{\mathcal{F}}_{t,\boldsymbol{\Gamma}}(\mathbf{h})\right) - \mathcal{S}_t(\mathbf{h}). \tag{19}$$

For the full-precision state $\mathbf{H}^{(t-1)}$, let $\mathbf{a}_t = \mathbf{d}_t(\mathbf{H}^{(t-1)};\boldsymbol{\Gamma})$. The low-bit trajectory inherits drift $\boldsymbol{\delta}_{t-1} = \widehat{\mathbf{H}}^{(t-1)} - \mathbf{H}^{(t-1)}$. LCTC measures its effect on the defect through the secant

$$\mathbf{s}_t(\alpha) = \frac{\mathbf{d}_t(\mathbf{H}^{(t-1)} + \alpha\boldsymbol{\delta}_{t-1}; \mathbf{\Gamma}) - \mathbf{d}_t(\mathbf{H}^{(t-1)}; \mathbf{\Gamma})}{\alpha}. \tag{20}$$

This term provides a task-space approximation of the trajectory-dependent error, with derivation given in Appendix A.2.8. Because logit perturbations have unequal effects on predictions, LCTC uses the categorical Fisher metric $\mathbf{F}(\mathbf{p}) = \mathrm{Diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$, where $\mathbf{p} = \mathrm{softmax}(\mathbf{z})$, with Fisher energy $\mathcal{E}_{\mathrm{F}}(\mathbf{v}) = \mathbf{v}^\top \mathbf{F}(\mathbf{p})\mathbf{v}$. We optimize the clean-state defect and trajectory sensitivity as

$$\mathcal{L}_{\mathrm{anchor}} = \mathbb{E}_{t,\mathbf{x}}[\mathcal{E}_{\mathrm{F}}(\mathbf{a}_t)], \qquad \mathcal{L}_{\mathrm{traj}} = \mathbb{E}_{t,\mathbf{x},\alpha}[\mathcal{E}_{\mathrm{F}}(\mathbf{s}_t(\alpha))]. \tag{21}$$

The objective combines teacher alignment, language modeling, and trajectory-aware correction:

$$\mathcal{L}_{\mathrm{LCTC}} = \mathcal{L}_{\mathrm{KD}} + \lambda_{\mathrm{NLL}}\mathcal{L}_{\mathrm{NLL}} + \lambda_{\mathrm{a}}\widetilde{\mathcal{L}}_{\mathrm{anchor}} + \lambda_{\mathrm{t}}\widetilde{\mathcal{L}}_{\mathrm{traj}} + \lambda_{\mathrm{reg}}\mathcal{R}(\mathbf{\Gamma}). \tag{22}$$

Normalization and regularization details are provided in Appendix A.2.9. Only $\mathbf{\Gamma}_{m,t}$ is optimized, while quantized weights, CLDC transformations, ILPA assignments, and parameters remain frozen. Thus, LCTC performs task-aligned calibration rather than retraining the quantized model.

5 Experiments

5.1 Experiment setup

Models and Datasets. We evaluate **LoopQuant** on representative LoopLMs, including Ouro (with thinking-enabled models) (Zhu et al., 2026), Huginn (Geiping et al., 2026), and Nanbeige4.2-3B (Lab et al., 2026). We report WikiText2 and C4 perplexity (PPL) (Merity et al., 2016; Raffel et al., 2023), together with seven zero-shot tasks: ARC-Challenge/Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA-openai/standard (Paperno et al., 2016), PIQA (Bisk et al., 2019), and WinoGrande (Sakaguchi et al., 2019). We further evaluate on reasoning and generation benchmarks, including MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and HumanEval (Chen et al., 2021).

Baseline Methods. We compare LoopQuant with representative low-bit PTQ methods, including SmoothQuant (Xiao et al., 2023) with channel scaling, QuaRot (Ashkboos et al., 2024) with orthogonal rotation, SpinQuant (Liu et al., 2025) with learnable rotation, FlatQuant (Sun et al., 2024) with affine transformation, and DartQuant (Shao et al., 2026) with distribution-aware rotation calibration. BF16 serves as the full-precision reference. We further compare with LoopQ (Fang et al., 2026); since its code is not publicly released, we report the results in its original paper in Appendix A.3.5.

Implementation Details. We use per-channel symmetric quantization for weights and per-token symmetric quantization for activations. For fair comparison, all methods use GPTQ (Frantar et al., 2022) as the weight quantization backend and 128 sampled WikiText2 sequences of length 1024 for calibration (Appendix A.3.6). CLDC optimizes loop-shared parameters with Adam (Appendix A.3.1), and LCTC details are provided in Appendix A.3.2. ILPA uses exact enumeration for fewer than 5 loops and 100 Monte Carlo permutations otherwise. All experiments are conducted on NVIDIA RTX PRO 6000 GPUs, and accuracy is evaluated with lm-eval (Gao et al., 2024).

5.2 Main Results

Comparison Results. As shown in Table 1, LoopQuant consistently outperforms low-bit PTQ methods across LoopLM architectures under $W4A6$ and $W4A4$. Here, $W4A6$ uses loop-wise mixed activation precision: half the loops use 4-bit activations and half use 8-bit, averaging 6 bits. Despite lower activation precision than $W4A8$ baselines, LoopQuant achieves superior performance across all models. For example, on Ouro-2.6B, LoopQuant achieves 70.20% average zero-shot accuracy under $W4A6$, outperforming the strongest baseline by 1.90 percentage points (pp) while reducing WikiText2 PPL from 9.09 to 8.04. Under $W4A4$, existing methods suffer substantial degradation, whereas LoopQuant maintains performance. On Huginn-0125 ($T = 6$), LoopQuant achieves 48.39% average zero-shot accuracy, substantially outperforming the best baseline (44.56%). These results show that LoopQuant mitigates quantization error propagation and provides a favorable accuracy–compression trade-off across LoopLMs. Full results are in Appendix A.3.4.

Table 1: Comparison of LoopQuant with representative low-bit PTQ methods on LoopLMs. Bold and underline denote the **best** and second-best quantized results.

Method	W/A	Ouro-1.4B			Ouro-2.6B			Nanbeige4.2-3B			Huginn-0125		
		Avg.↑	WT2↓	C4↓	Avg.↑	WT2↓	C4↓	Avg.↑	WT2↓	C4↓	Avg.↑	WT2↓	C4↓
BF16	16/16	66.85	8.60	12.61	71.12	7.61	11.42	58.68	24.17	31.45	49.89	12.10	16.28
SmoothQuant	4/8	60.65	11.25	15.38	67.12	9.48	13.46	56.10	28.63	32.38	46.95	13.41	17.65
QuaRot	4/8	62.53	10.78	15.21	68.30	9.09	13.02	53.83	34.09	41.54	47.73	13.38	17.63
SpinQuant	4/8	62.69	10.79	15.06	67.00	9.11	13.03	54.47	31.10	35.86	47.33	13.34	17.62
FlatQuant	4/8	61.37	10.65	14.84	67.31	9.26	13.18	55.14	25.99	32.93	47.31	13.45	17.71
DartQuant	4/8	60.52	11.45	15.63	65.65	10.04	14.01	53.06	30.69	38.15	47.32	13.50	17.76
LoopQuant	4/6	**64.62**	**9.48**	**13.57**	**70.20**	**8.04**	**11.94**	**56.74**	**25.84**	**32.19**	**49.09**	**12.47**	**16.73**
SmoothQuant	4/4	54.29	13.99	18.36	61.76	11.41	15.54	50.70	33.85	38.82	44.10	14.92	19.30
QuaRot	4/4	58.13	12.50	17.15	63.58	10.90	14.90	49.74	34.72	41.88	44.53	14.76	**19.07**
SpinQuant	4/4	55.77	13.54	18.43	58.94	13.23	18.00	48.33	33.91	37.95	44.50	14.95	19.35
FlatQuant	4/4	57.14	12.71	17.04	62.57	10.96	15.11	49.14	30.44	36.39	44.56	14.98	19.37
DartQuant	4/4	51.46	15.77	20.49	55.69	14.37	18.67	46.19	36.28	38.76	44.06	15.17	19.57
LoopQuant	4/4	**64.04**	**9.86**	**14.00**	**68.89**	**8.27**	**12.24**	**56.11**	**26.64**	**33.02**	**48.39**	**12.78**	17.11

Experiments on Reasoning-oriented Models. Thinking-oriented LLMs rely on extended reasoning and complex intermediate computation, making them highly sensitive to quantization errors. We further evaluate LoopQuant on Ouro-1.4B-thinking and Ouro-2.6B-thinking using MMLU, GSM8K, and HumanEval (Table 2). Under $W4A6$, LoopQuant outperforms all $W4A8$ baselines on both models. For example, on Ouro-1.4B-thinking, LoopQuant achieves 61.49% GSM8K and 70.73% HumanEval accuracy, outperforming the strongest baseline by 7.49 pp on HumanEval. At $W4A4$, existing methods suffer significant degradation on mathematical and code reasoning tasks, with some nearly collapsing. In contrast, LoopQuant retains over 97% of BF16 performance averaged across MMLU, GSM8K, and HumanEval on Ouro-2.6B-thinking. These results show that LoopQuant mitigates quantization-induced error accumulation in recursive reasoning (Appendix A.3.9) while preserving high-level reasoning under extremely low-bit inference.

Table 2: Evaluation on reasoning-oriented LoopLMs.

Method	W/A	Ouro-1.4B-thinking			Ouro-2.6B-thinking		
		MMLU	GSM8K	HumanEval	MMLU	GSM8K	HumanEval
BF16	16/16	67.71	69.57	71.34	76.20	73.31	82.32
SmoothQuant	4/8	59.42	34.38	63.24	68.97	60.56	74.22
QuaRot	4/8	57.24	34.23	58.98	71.50	65.94	79.71
SpinQuant	4/8	56.68	53.25	60.80	71.26	63.67	75.44
FlatQuant	4/8	60.50	59.09	61.02	70.21	63.59	77.27
DartQuant	4/8	57.71	53.25	58.98	67.19	63.75	71.17
LoopQuant	4/6	**61.20**	**61.49**	**70.73**	**72.74**	**72.84**	**81.71**
SmoothQuant	4/4	50.54	11.83	41.34	62.46	43.69	61.85
QuaRot	4/4	51.86	29.42	49.66	67.09	58.70	71.00
SpinQuant	4/4	44.78	18.20	14.63	63.67	43.69	64.90
FlatQuant	4/4	52.59	41.46	52.32	64.86	57.26	61.85
DartQuant	4/4	41.99	6.52	20.73	50.76	20.92	4.88
LoopQuant	4/4	**59.51**	**60.88**	**60.98**	**72.46**	**72.02**	**81.15**

5.3 Ablation Studies

Component Ablation. We conduct an ablation on Nanbeige4.2-3B under $W4A6$ to evaluate each LoopQuant component. As shown in Table 3, each component improves quantization performance, addressing different sources of recursive inference error. Starting from the vanilla $W4A6$ baseline, CLDC reduces WikiText2 PPL from 47.39 to 28.41 by regularizing activation distributions (Appendix A.3.10). Adding ILPA further reduces WikiText2 and C4 PPL to 27.05 and 34.17, respectively, by allocating activation precision according to loop-specific sensitivity. With LCTC, LoopQuant achieves 25.84 WikiText2 and 32.19 C4 PPL, demonstrating the importance of trajectory-aware correction for accumulated quantization errors (Appendix A.3.8). Moreover, the framework adds only 53.08 MB storage, corresponding to 1.39% of the model size, indicating negligible memory cost. These results confirm that CLDC, ILPA, and LCTC are complementary, enabling effective and efficient low-bit quantization for recursive language models.

Table 3: Impact of different components in LoopQuant.

W/A	CLDC $\mathbf{D}_m^\star$	CLDC $\mathbf{R}_m^\star$	CLDC $\mathbf{\Pi}_m^\star$	ILPA	LCTC	Storage Overhead	Wiki2 PPL(↓)	C4 PPL(↓)
W4A6	✗	✗	✗	✗	✗	0	47.39	53.65
	✓	✗	✗			+1.42 MB (0.04%)	36.91	41.52
	✓	✓	✗			+46.84 MB (1.22%)	28.41	36.31
	✓	✓	✓			+52.51 MB (1.37%)	27.85	35.10
		✓		✓	✗	+52.51 MB (1.37%)	27.05	34.17
				✓	✓	+53.08 MB (1.39%)	**25.84**	**32.19**

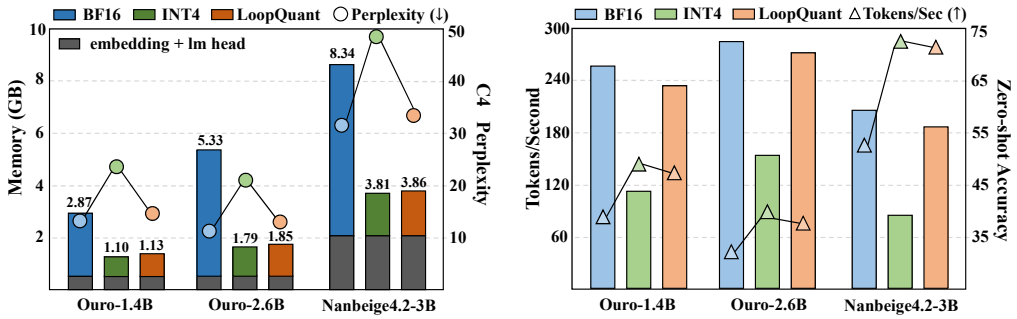


Figure 3: Efficiency analysis. **Left**: Memory consumption and C4 perplexity of BF16, INT4, and LoopQuant across three LoopLMs, where gray regions denote the embedding and LM head. **Right**: Decoding throughput (Tokens/Sec) and average zero-shot accuracy under the same settings.

Ablation of Loop-wise Bit Allocation. We further analyze ILPA by comparing different loop-wise activation bit allocation strategies on Huginn-0125 ($T = 6$). As shown in Table 4, all strategies maintain the same average activation precision of 6 bits. Compared with uniform allocation, random allocation (averaged over five sampled patterns) may assign low precision to sensitive loops, degrading performance. Although local and marginal sensitivity-based heuristics achieve better results, they still underperform ILPA, indicating that sensitivity alone cannot capture the optimal precision assignment. By globally optimizing loop-wise precision allocation, ILPA achieves 12.47 WikiText2 perplexity and 49.09% average zero-shot accuracy. Further validation of ILPA across different quantization backbones is provided in Appendix A.3.7.

Table 4: Comparison of loop-wise bit allocation strategies.

Model	Allocation Strategy	Loop-wise Bits ($t = 1 \rightarrow 6$)	Wiki2 PPL(↓)	Avg. Acc.↑
Huginn-0125 ($T = 6$)	Uniform	[6,6,6,6,6,6]	13.82	46.92
	Random	$3 \times$ A4 + $3 \times$ A8	15.21	44.83
	Local Sensitivity	[4,8,4,8,8,4]	13.05	48.01
	Marginal Sensitivity	[4,8,8,4,4,8]	13.67	47.54
	ILPA	**[4,4,4,8,8,8]**	**12.47**	**49.09**

5.4 Efficiency Analysis of LoopQuant

Memory Savings. We evaluate the storage efficiency of LoopQuant against BF16 and naive INT4 RTN across three LoopLMs, as shown in Fig. 3 (left). LoopQuant reduces overall model memory by 53.7–65.3% compared with BF16, while the quantized linear layers alone achieve approximately 74% memory savings. Compared with naive INT4 RTN, LoopQuant introduces at most 3.4% additional memory overhead, largely preserving the storage efficiency of standard 4-bit quantization (Appendix A.3.3). Despite this negligible overhead, LoopQuant substantially mitigates C4 perplexity degradation caused by naive INT4 and achieves performance close to BF16.

Inference Speedup. We evaluate LoopQuant inference efficiency with PyTorch on NVIDIA RTX PRO 6000 GPUs, using one GPU per model. With batch size 4, we measure decoding throughput by generating 1,024 tokens after prefilling 4,096 tokens. As shown in Fig. 3 (right), LoopQuant improves throughput by 1.5–1.7$\times$ over BF16 across three models. Although naive INT4 attains slightly higher throughput, it suffers severe accuracy degradation; LoopQuant preserves most INT4 efficiency gain while maintaining zero-shot accuracy close to BF16.

6 Conclusion

We present LoopQuant, a loop-aware PTQ framework for low-bit LoopLM quantization. LoopQuant addresses cross-loop activation dynamics, heterogeneous loop sensitivity, and misalignment between local objectives and recursive task behavior. CLDC learns a shared quantization-friendly representation, ILPA allocates loop-wise activation precision under a fixed bit budget, and LCTC calibrates quantization along the low-bit recursive trajectory. Across LoopLMs, LoopQuant consistently outperforms low-bit PTQ methods; W4A4 matches or surpasses W4A8 baselines while preserving reasoning capability. It delivers substantial memory savings and decoding speedup with negligible storage overhead, enabling accurate and deployment-friendly low-bit LoopLM inference.

AI USE STATEMENT

Generative AI tools were used during manuscript preparation only for language-related assistance, including proofreading, translation, concision, and improvements to readability. They were not used to generate the core research ideas, methodological contributions, experimental protocols, or scientific conclusions. Any AI-supported edits were manually inspected and, when necessary, revised by the authors before inclusion in the manuscript. Responsibility for the technical content, reported results, and final presentation rests entirely with the authors.

ETHICS STATEMENT

This work studies LoopQuant, a post-training quantization framework for improving the deployment efficiency of Looped Language Models. Our experiments are conducted on existing pretrained models and publicly available calibration and evaluation benchmarks. Reducing memory and inference costs may broaden access to LoopLMs, but greater deployment efficiency can also facilitate unintended or harmful uses. Quantized models may inherit biases, factual inaccuracies, and unsafe behaviors from their full-precision counterparts, and low-bit quantization may alter model behavior in ways not captured by standard accuracy benchmarks. Our experiments evaluate predictive performance and system efficiency rather than safety, fairness, privacy, or robustness guarantees. We therefore recommend application-specific evaluation and appropriate safeguards before real-world deployment, together with compliance with the licenses and usage policies of the underlying models and datasets.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we present the recursive quantization formulation and the three components of LoopQuant in Secs. 4.1–4.4, with detailed pseudocode, derivations, and implementation details provided in Appendices A.1 and A.2. Sec. 5 specifies the evaluated models, datasets, calibration procedure, quantization settings, evaluation protocol, and hardware. Detailed task-level results and calibration robustness analyses are reported in Appendices A.3.4 and A.3.6, while deployment storage accounting is provided in Appendix A.3.3. We also provide an anonymous repository containing the LoopQuant implementation, quantization and evaluation scripts, and experimental configurations to support reproduction of the reported results.

REFERENCES

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, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *International Conference on Learning Representations*, volume 2025, pp. 34282–34327, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 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, 2018. URL https://arxiv.org/abs/1803.05457.

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, 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.

Rui Fang, Hsi-Wen Chen, and Ming-Syan Chen. Loopq: Quantization for recursive transformers. *arXiv preprint arXiv:2605.16343*, 2026.

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, Jonathan Tow, Behnam Abbasi, Stella Biderman, Sidney Black, Anthony DiPofi, Colin Foster, Luke Golding, Jenness Hsu, Alexandre Le Noac'h, Henry Li, Kyle McDonell, Niklas Muennighoff, Charles Ociepa, Jason Phang, Laura Reynolds, Hailey Schoelkopf, Anthony Skowron, Lintang Sutawika, Eileen Tang, Akshata Thite, Bowen Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. https://zenodo.org/records/12608602, July 2024. Accessed: 2024-07.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Nanbeige Lab, Chen Yang, Chengrui Huang, Fufeng Lan, Hanhui Chen, Hao Zhou, Huatong Song, Jiaqi Cao, Jiaying Zhu, Jinlin Niu, et al. Nanbeige4. 2-3b: Unlocking agentic capabilities in a compact model. *arXiv preprint arXiv:2607.22083*, 2026.

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 *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025.

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, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016. URL https://arxiv.org/abs/1606.06031.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Utkarsh Saxena, Sayeh Sharify, Kaushik Roy, and Xin Wang. Resq: Mixed-precision quantization of large language models with low-rank residuals. *arXiv preprint arXiv:2412.14363*, 2024.

Yuantian Shao, Yuanteng Chen, Peisong Wang, Jianlin Yu, Jing Lin, Zhihui Wei, Jian Cheng, et al. Dartquant: efficient rotational distribution calibration for llm quantization. *Advances in Neural Information Processing Systems*, 38:143936–143970, 2026.

Lloyd S Shapley et al. A value for n-person games. 1953.

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. *arXiv preprint arXiv:2410.09426*, 2024.

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, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Junchen Zhao, Ali Derakhshan, Jayden Kana Hyman, Junhao Dong, Sangeetha Abdu Jyothi, and Ian Harris. Coopq: Cooperative game inspired layerwise mixed precision quantization for llms, 2025. URL https://arxiv.org/abs/2509.15455.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models, 2026. URL https://arxiv.org/abs/2510.25741.

A APPENDIX

APPENDIX OVERVIEW

This appendix is organized as follows.

- Section A.1: Pseudocode of LoopQuant.
  - Section A.1.1: Overall Workflow of LoopQuant.
  - Section A.1.2: Cross-Loop Distribution Canonicalization (CLDC).
  - Section A.1.3: Interaction-Aware Loop-wise Precision Allocation (ILPA).
  - Section A.1.4: Loop-Conditioned Task Calibration (LCTC).
- Section A.2: Detailed Derivations and Implementation Details.
  - Section A.2.1: Orthogonal Canonical Geometry Parameterization.
  - Section A.2.2: Robust Cross-Loop Synchronization Objective.
  - Section A.2.3: Group-Aware Channel Reordering.
  - Section A.2.4: Shapley-Style Loop Sensitivity Estimation.
  - Section A.2.5: Recursive Interaction Modeling.
  - Section A.2.6: Exact and Greedy Precision Allocation.
  - Section A.2.7: Loop-Conditioned Calibration Parameterization.
  - Section A.2.8: Task-Space Defect Transport.
  - Section A.2.9: Fisher-Weighted Trajectory Calibration.
- Section A.3: Additional Experimental Details.
  - Section A.3.1: CLDC Optimization Details.
  - Section A.3.2: LCTC Optimization Details.
  - Section A.3.3: Deployment Storage Overhead.
  - Section A.3.4: Detailed Main Results.
  - Section A.3.5: Comparison with LoopQ.
  - Section A.3.6: Ablation on Calibration Data.
  - Section A.3.7: Effectiveness of ILPA with Different Quantization Backbones.
  - Section A.3.8: Ablation on LCTC Calibration Parameters.
  - Section A.3.9: Layer-wise Quantization Error during Reasoning.
  - Section A.3.10: Activation Distribution Transformation by CLDC.

A.1 PSEUDOCODE OF LOOPQUANT

This section provides the detailed pseudocode of LoopQuant. The complete quantization procedure consists of three stages: Cross-Loop Distribution Canonicalization (CLDC), Interaction-Aware Loop-wise Precision Allocation (ILPA), and Loop-Conditioned Task Calibration (LCTC). CLDC learns a shared quantization-friendly representation across recursive states, ILPA determines loop-wise activation precision under a fixed bit-width budget, and LCTC further aligns the quantized model with recursive task behavior through task-space calibration.

A.1.1 OVERALL WORKFLOW OF LOOPQUANT

Algorithm 1 summarizes the overall workflow of LoopQuant. The framework first canonicalizes recursive activation distributions through CLDC, then allocates loop-wise activation precision with ILPA under a fixed bit-width budget, and finally performs task-aligned calibration along the low-bit recursive trajectory through LCTC.

Algorithm 1: LoopQuant: Overall Quantization Pipeline

Input: Pretrained LoopLM $\mathcal{M}$, calibration dataset $\mathcal{D}$, target activation bit-width budget $\bar{b}$
Output: Quantized LoopLM $\widehat{\mathcal{M}}$

for *each recursively reused linear module* m **do**
  `// Stage 1: Cross-Loop Distribution Canonicalization (CLDC)`
  Collect loop-wise calibration activations $\{\mathbf{X}_m^{(t)}\}_{t=1}^{T}$;
  Initialize the shared canonical transformation;
  Optimize cross-loop reconstruction robustness, anomaly synchronization, and group-aware reordering to obtain $\mathbf{P}_m^\star = \mathbf{D}_m^\star \mathbf{R}_m^\star \mathbf{\Pi}_m^\star$;
  Apply the canonical transformation to obtain $\widetilde{\mathbf{W}}_m$;
  Quantize the canonicalized weight with GPTQ: $\widehat{\mathbf{W}}_m = Q_W(\widetilde{\mathbf{W}}_m)$;

`// Stage 2: Interaction-Aware Loop-wise Precision Allocation (ILPA)`
Construct candidate activation precision configurations: $b_t \in \{b_{\mathrm{L}}, b_{\mathrm{H}}\}$;
Evaluate trajectory-level task cost under recursive precision contexts;
Estimate loop sensitivity ϕ_t and cross-loop interaction $\widetilde{\Gamma}_{ij}$;
Solve the precision assignment by exact enumeration for small T, or greedy allocation otherwise, under budget $\bar{b}$;
Obtain the final loop-wise precision configuration $\{b_t^\star\}_{t=1}^{T}$;

`// Stage 3: Loop-Conditioned Task Calibration (LCTC)`
Freeze quantized weights, CLDC transformations, and ILPA precision assignments;
Introduce static loop-conditioned calibration parameters: $\mathbf{\Gamma}_{m,t} = \{\mathbf{c}_{m,t}, \mathbf{g}_{m,t}, \boldsymbol{\beta}_{m,t}\}$;
Optimize task-space calibration objective $\mathcal{L}_{\mathrm{LCTC}}$;
Remove all temporary optimization variables;
Assemble the final deployable model: $\widehat{\mathcal{M}} = \{\widehat{\mathbf{W}}_m, \mathbf{P}_m^\star, \mathbf{\Gamma}_{m,t}, b_t^\star\}$
return $\widehat{\mathcal{M}}$

A.1.2 Cross-Loop Distribution Canonicalization (CLDC)

CLDC aims to construct a shared quantization-friendly representation across recursive states while preserving the parameter-sharing structure of LoopLMs. It first learns a shared canonical geometry from all loop-wise calibration states, then reduces cross-loop distribution mismatch through robust synchronization, and finally performs group-aware channel reordering to improve group-wise quantization compatibility.

Algorithm 2 summarizes the CLDC procedure. CLDC first learns a shared canonical geometry across recursive states, then aligns cross-loop activation distributions through robust synchronization, and finally improves group-wise quantization compatibility with channel reordering. The resulting transformation is shared across all loops and remains fixed for subsequent ILPA and LCTC stages.

A.1.3 Interaction-Aware Loop-wise Precision Allocation (ILPA)

ILPA determines loop-wise activation precision under a fixed bit-width budget. It first evaluates the task impact of different precision configurations along the recursive trajectory, then estimates loop sensitivity and cross-loop interaction, and finally allocates activation precision through a budget-aware optimization procedure.

Algorithm 3 summarizes the ILPA procedure. ILPA first evaluates precision configurations on the complete recursive trajectory, then identifies loop-wise sensitivity and cross-loop interaction, and finally allocates activation precision under the target bit-width budget. The obtained precision configuration remains fixed during inference and is used by the subsequent LCTC stage.

Algorithm 2: Cross-Loop Distribution Canonicalization (CLDC)

Input: Shared linear weight $\mathbf{W}_m$, loop-wise calibration activations $\{\mathbf{X}_m^{(t)}\}_{t=1}^{T}$
Output: Canonicalized weight $\widetilde{\mathbf{W}}_m$, shared transformation $\mathbf{P}_m^\star$

// Stage I: Shared canonical geometry learning
Initialize the shared transformation: $\mathbf{P}_m^{\mathrm{cont}} = \mathbf{D}_m\mathbf{R}_m$;
Jointly optimize $\mathbf{D}_m$ and $\mathbf{R}_m$ using all loop calibration states;
Apply the shared transformation: $\widetilde{\mathbf{X}}_m^{(t)} = \mathbf{X}_m^{(t)}\mathbf{D}_m\mathbf{R}_m$;
Absorb the inverse transformation into the weight: $\widetilde{\mathbf{W}}_m = \mathbf{W}_m\mathbf{D}_m^{-1}\mathbf{R}_m$;
// Stage II: Robust cross-loop synchronization
Compute loop-wise quantization reconstruction error $\mathcal{E}_m^{(t)}$;
Optimize the smooth worst-loop objective $\mathcal{L}_{\mathrm{rob}}$;
Extract cross-loop anomaly profiles $\mathbf{p}_m^{(t)}$;
Optimize anomaly synchronization objective $\mathcal{L}_{\mathrm{sync}}$;
Obtain optimized continuous canonical geometry: $(\mathbf{D}_m^\star, \mathbf{R}_m^\star)$;
// Stage III: Group-aware channel reordering
Construct cross-loop channel signatures;
Initialize channel permutation according to worst-case dynamic range;
Optimize hard permutation: $\mathbf{\Pi}_m^\star$;
Construct final shared transformation: $\mathbf{P}_m^\star = \mathbf{D}_m^\star\mathbf{R}_m^\star\mathbf{\Pi}_m^\star$;
Apply final canonicalization to obtain $\widetilde{\mathbf{W}}_m$;
return $\widetilde{\mathbf{W}}_m, \mathbf{P}_m^\star$

A.1.4 Loop-Conditioned Task Calibration (LCTC)

LCTC performs the final task-level calibration after CLDC and ILPA. It keeps the quantization structure and precision assignment fixed, and only optimizes lightweight loop-conditioned calibration parameters along the low-bit recursive trajectory. The calibration objective jointly considers teacher alignment, language modeling loss, clean-state task defects, and trajectory-dependent defects.

Algorithm 4 summarizes the final calibration procedure of LoopQuant. LCTC does not modify the quantization structure or precision allocation. Instead, it optimizes lightweight loop-conditioned parameters to correct task-space quantization defects and trajectory-dependent error amplification. After calibration, only static calibration metadata are retained in addition to the quantized model; temporary optimization states are discarded.

A.2 Detailed Derivations and Implementation Details

A.2.1 Orthogonal Canonical Geometry Parameterization

The goal of CLDC is to learn a shared transformation that improves quantization compatibility across all recurrent states while preserving the original computation of LoopLMs. For a recursively reused linear module with shared weight $\mathbf{W}_m \in \mathbb{R}^{d_o \times d_i}$, let $\mathbf{X}_m^{(t)}$ denote the input activation collected from the t-th loop. A naive approach would optimize an independent transformation $\mathbf{P}_m^{(t)}$ for each loop. Although this formulation can better fit individual activation distributions, it violates the parameter-sharing property of LoopLMs and introduces additional deployment overhead. Therefore, CLDC seeks a single canonical transformation shared across all recurrent states:

$$\mathbf{P}_m^{(1)} = \mathbf{P}_m^{(2)} = \cdots = \mathbf{P}_m^{(T)} \triangleq \mathbf{P}_m. \tag{23}$$

We decompose the continuous canonical transformation into a channel-wise scaling component and an orthogonal rotation component:

$$\mathbf{P}_m^{\mathrm{cont}} = \mathbf{D}_m\mathbf{R}_m, \qquad \mathbf{D}_m = \mathrm{Diag}(\mathbf{s}_m), \qquad \mathbf{s}_m > 0, \tag{24}$$

Algorithm 3: Interaction-Aware Loop-wise Precision Allocation (ILPA)

Input: Quantized LoopLM, calibration dataset $\mathcal{D}$, low/high activation precision $b_{\mathrm{L}}, b_{\mathrm{H}}$, target average bit-width $\bar{b}$

Output: Loop-wise precision configuration $\{b_t^\star\}_{t=1}^T$

// Stage I: Trajectory-level precision evaluation

Construct candidate precision configurations: $\mathcal{S} \subseteq \{1, \dots, T\}$;

Evaluate recursive forward passes under different precision configurations;

Compute task-level cost: $v(\mathcal{S})$;

// Stage II: Sensitivity and interaction estimation

Estimate context-dependent loop sensitivity: ϕ_t;

Estimate pairwise recursive interaction: $\widetilde{\Gamma}_{ij}$;

Construct the precision allocation objective: $\mathcal{J}(\mathbf{q}) = \sum_t q_t \phi_t + \lambda \sum_{i<j} q_i q_j \widetilde{\Gamma}_{ij}$;

// Stage III: Budget-aware precision allocation

if T *is small* **then**

  Enumerate feasible assignments satisfying the bit budget and select the minimum-$\mathcal{J}$ configuration;

else

  Initialize all loops with high activation precision: $q_t = 0$;

  while *activation bit-width budget is not satisfied* **do**

    Compute marginal precision reduction cost for each candidate loop;

    Select the loop with minimum marginal cost: $t^\star = \arg\min_t \Delta \mathcal{J}_t$;

    Assign loop $t^\star$ to low precision: $q_{t^\star} = 1$;

Obtain final precision assignment: $\{b_t^\star\}_{t=1}^T$;

return $\{b_t^\star\}_{t=1}^T$

Algorithm 4: Loop-Conditioned Task Calibration (LCTC)

Input: Quantized LoopLM, CLDC transformations, ILPA precision configuration, teacher model $\mathcal{M}$, calibration dataset $\mathcal{D}$

Output: Final deployable LoopQuant model $\widehat{\mathcal{M}}$

// Stage I: Loop-conditioned calibration initialization

Freeze quantized weights, CLDC transformations, and ILPA precision assignment;

Initialize static calibration parameters: $\mathbf{\Gamma}_{m,t} = \{\mathbf{c}_{m,t}, \mathbf{g}_{m,t}, \boldsymbol{\beta}_{m,t}\}$;

// Stage II: Task-space defect calibration

Collect low-bit recursive trajectories;

Compute clean-state task defects: $\mathbf{a}_t$;

Compute trajectory-dependent defects: $\mathbf{s}_t(\alpha)$;

Evaluate Fisher-weighted task losses: $\mathcal{L}_{\mathrm{anchor}}$ and $\mathcal{L}_{\mathrm{traj}}$;

// Stage III: Task-aligned optimization

Optimize loop-conditioned calibration parameters:

$\mathbf{\Gamma}_{m,t} \leftarrow \arg\min \mathcal{L}_{\mathrm{LCTC}}$;

Keep quantized weights, CLDC transformations, ILPA precision assignments, and backbone parameters frozen;

Remove temporary optimization variables;

Assemble final deployable model: $\widehat{\mathcal{M}} = \{\widehat{\mathbf{W}}_m, \mathbf{P}_m^\star, \mathbf{\Gamma}_{m,t}, b_t^\star\}$;

return $\widehat{\mathcal{M}}$

where $\mathbf{D}_m$ adjusts the channel-wise dynamic range and $\mathbf{R}_m$ redistributes cross-channel geometry while preserving the Euclidean structure. The orthogonality constraint is imposed as

$$\mathbf{R}_m \mathbf{R}_m^\top = \mathbf{I}. \tag{25}$$

Directly optimizing a dense orthogonal matrix introduces a large number of constrained parameters and makes optimization inefficient. To obtain an expressive yet tractable parameterization, we represent the rotation matrix as

$$\mathbf{R}_m = \mathbf{R}_m^{(0)} \mathbf{U}_m, \tag{26}$$

where $\mathbf{R}_m^{(0)}$ is a fixed structured orthogonal transformation and $\mathbf{U}_m$ is a learnable orthogonal correction. Instead of optimizing a full dense matrix, we adopt a block-diagonal parameterization:

$$\mathbf{U}_m = \mathrm{BlkDiag}(\mathbf{U}_{m,1}, \ldots, \mathbf{U}_{m,B}). \tag{27}$$

For the b-th block, we introduce an unconstrained matrix $\mathbf{Z}_{m,b}$ and construct its corresponding skew-symmetric matrix:

$$\mathbf{A}_{m,b} = \mathbf{Z}_{m,b} - \mathbf{Z}_{m,b}^\top. \tag{28}$$

The orthogonal correction is then obtained through the Cayley transform:

$$\mathbf{U}_{m,b} = \left(\mathbf{I} - \frac{1}{2}\mathbf{A}_{m,b}\right)^{-1} \left(\mathbf{I} + \frac{1}{2}\mathbf{A}_{m,b}\right). \tag{29}$$

Since $\mathbf{A}_{m,b}^\top = -\mathbf{A}_{m,b}$, the Cayley transform guarantees

$$\mathbf{U}_{m,b}\mathbf{U}_{m,b}^\top = \mathbf{I}, \tag{30}$$

and consequently

$$\mathbf{R}_m \mathbf{R}_m^\top = \mathbf{R}_m^{(0)} \mathbf{U}_m \mathbf{U}_m^\top (\mathbf{R}_m^{(0)})^\top = \mathbf{I}. \tag{31}$$

Therefore, the learned rotation remains strictly orthogonal throughout optimization without requiring explicit projection onto the orthogonal manifold.

We initialize all learnable correction matrices as $\mathbf{Z}_{m,b} = \mathbf{0}$. At initialization, this gives

$$\mathbf{U}_{m,b} = \mathbf{I}, \qquad \mathbf{R}_m = \mathbf{R}_m^{(0)}, \tag{32}$$

meaning that optimization starts from the original structured rotation space and only learns data-supported orthogonal corrections from cross-loop calibration states.

For every loop, CLDC applies the same transformation to activations and compensates the inverse transformation in weights:

$$\widetilde{\mathbf{X}}_m^{(t)} = \mathbf{X}_m^{(t)} \mathbf{D}_m \mathbf{R}_m, \qquad \widetilde{\mathbf{W}}_m = \mathbf{W}_m \mathbf{D}_m^{-1} \mathbf{R}_m. \tag{33}$$

The transformed computation satisfies

$$\begin{aligned} \widetilde{\mathbf{X}}_m^{(t)} \widetilde{\mathbf{W}}_m^\top &= \mathbf{X}_m^{(t)} \mathbf{D}_m \mathbf{R}_m \mathbf{R}_m^\top \mathbf{D}_m^{-1} \mathbf{W}_m^\top \\ &= \mathbf{X}_m^{(t)} \mathbf{W}_m^\top, \qquad \forall t, \end{aligned} \tag{34}$$

which proves that CLDC preserves the original full-precision function while modifying only the quantization geometry.

A.2.2 Robust Cross-Loop Synchronization Objective

After obtaining the shared canonical geometry, CLDC further optimizes the transformation by jointly considering the quantization behavior across all recurrent states. For the t-th loop, we measure the quantization distortion of the transformed representation using the normalized reconstruction error:

$$\mathcal{E}_m^{(t)} = \frac{\left\| Q_A(\widetilde{\mathbf{X}}_m^{(t)}) Q_W(\widetilde{\mathbf{W}}_m)^\top - \mathbf{X}_m^{(t)} \mathbf{W}_m^\top \right\|_F^2}{\left\| \mathbf{X}_m^{(t)} \mathbf{W}_m^\top \right\|_F^2}, \tag{35}$$

where $Q_A(\cdot)$ and $Q_W(\cdot)$ denote the activation and weight quantizers under the target quantization configuration, respectively. This reconstruction error serves as a local approximation of the loop-dependent quantization perturbation $\boldsymbol{\epsilon}_t^{\mathrm{loc}}$ introduced in Sec. 4.1.

A straightforward objective is to minimize the average reconstruction error:

$$\mathcal{L}_{\mathrm{avg}} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{E}_m^{(t)}. \tag{36}$$

However, this objective treats all recurrent states equally and may allow a small number of difficult loops to be dominated by easily quantized states. Such behavior is undesirable in LoopLMs because severe distortion at an early recurrent position can propagate through subsequent loops and accumulate along the recursive trajectory.

To explicitly control the worst-case quantization risk across loops, we introduce a smooth worst-loop objective:

$$\mathcal{L}_{\mathrm{rob}} = \tau \log \left(\frac{1}{T} \sum_{t=1}^{T} \exp \left(\frac{\mathcal{E}_m^{(t)}}{\tau} \right) \right), \tag{37}$$

where τ controls the smoothness of the approximation. When τ approaches zero, this objective converges to the maximum loop-wise error:

$$\lim_{\tau \to 0} \mathcal{L}_{\mathrm{rob}} = \max_t \mathcal{E}_m^{(t)}. \tag{38}$$

Therefore, $\mathcal{L}_{\mathrm{rob}}$ encourages the shared canonical geometry to reduce the largest quantization risk among all recurrent states rather than optimizing only the average behavior.

Nevertheless, reducing reconstruction error does not guarantee that different loops exhibit consistent activation outlier structures. After transformation, different recurrent states may still produce shifted anomaly channels, which can degrade group-wise quantization because channels with different dynamic ranges may share identical quantization parameters.

To characterize the cross-loop anomaly dynamics, we define the high-percentile activation magnitude of channel c at loop t as

$$a_{m,c}^{(t)} = \mathrm{Quantile}_q \left(\left| \widetilde{\mathbf{X}}_{m,:,c}^{(t)} \right| \right), \tag{39}$$

where q denotes a high percentile threshold. The anomaly statistic is further normalized across channels:

$$p_{m,c}^{(t)} = \frac{a_{m,c}^{(t)}}{\sum_{k=1}^{d_i} a_{m,k}^{(t)}}. \tag{40}$$

The resulting vector $\mathbf{p}_m^{(t)}$ represents the relative distribution of activation anomalies across channels for loop t. Instead of selecting a specific loop as a reference state, CLDC assumes that all recurrent states should share a consistent anomaly structure. Therefore, we define the cross-loop anomaly barycenter as

$$\overline{\mathbf{p}}_m = \arg\min_{\mathbf{p}} \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{p}_m^{(t)} - \mathbf{p}\|_2^2. \tag{41}$$

The above objective has a closed-form solution:

$$\overline{\mathbf{p}}_m = \frac{1}{T} \sum_{t=1}^{T} \mathbf{p}_m^{(t)}. \tag{42}$$

Based on this shared anomaly center, we define the cross-loop synchronization loss:

$$\mathcal{L}_{\text{sync}} = \frac{1}{T} \sum_{t=1}^{T} \|\mathbf{p}_m^{(t)} - \overline{\mathbf{p}}_m\|_2^2. \tag{43}$$

Unlike choosing one loop as an artificial anchor, this formulation jointly reduces the distribution discrepancy among all recurrent states and encourages different loops to share similar anomaly structures in the learned canonical space.

Finally, because channel scaling can arbitrarily enlarge activation scales while reducing activation quantization error, excessive scaling may transfer quantization difficulty from activations to weights. We therefore introduce a scale regularization term:

$$\mathcal{L}_{\text{reg}} = \frac{1}{d_i} \|\mathbf{s}_m - \mathbf{1}\|_2^2. \tag{44}$$

Combining reconstruction robustness, anomaly synchronization, and scale regularization, the final optimization objective for the shared canonical geometry is:

$$(\mathbf{D}_m^\star, \mathbf{R}_m^\star) = \arg\min \left[\mathcal{L}_{\text{rob}} + \lambda_{\text{sync}} \mathcal{L}_{\text{sync}} + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}}\right]. \tag{45}$$

The optimized transformation therefore represents a consensus quantization geometry determined by the complete recursive trajectory, rather than a local optimum for any individual loop.

A.2.3 Group-Aware Channel Reordering

Although the continuous canonical geometry improves the overall quantization compatibility across recurrent states, group-wise quantization introduces an additional constraint: channels within the same quantization group must share identical quantization parameters. Therefore, even after canonicalization, channels with inconsistent dynamic ranges across loops may still be assigned to the same group and introduce intra-group quantization mismatch.

To address this issue, CLDC performs a discrete channel reordering step after optimizing the continuous transformation. Unlike continuous permutation relaxation, which introduces additional optimization variables and deployment overhead, we formulate reordering as a hard group assignment problem that directly optimizes the final quantization layout.

After fixing the optimized scaling and rotation matrices $\mathbf{D}_m^\star$ and $\mathbf{R}_m^\star$, we first characterize each channel using its complete cross-loop dynamic behavior. For channel c at loop t, we define the logarithmic range signature as

$$z_{m,c}^{(t)} = \log\left(a_{m,c}^{(t)} + \varepsilon\right), \tag{46}$$

where $a_{m,c}^{(t)}$ is the anomaly statistic defined in Appendix A.2.2 and ε is a small constant for numerical stability. The complete cross-loop signature of channel c is represented as

$$\mathbf{z}_{m,c} = \left[z_{m,c}^{(1)}, z_{m,c}^{(2)}, \ldots, z_{m,c}^{(T)} \right]. \tag{47}$$

A channel permutation matrix $\mathbf{\Pi}_m$ divides input channels into K consecutive quantization groups:

$$\{\mathcal{G}_{m,1}(\mathbf{\Pi}_m), \ldots, \mathcal{G}_{m,K}(\mathbf{\Pi}_m)\}. \tag{48}$$

For a given loop t, we measure the quantization incompatibility inside each group by the variance of channel signatures:

$$\mathcal{C}_m^{(t)}(\mathbf{\Pi}_m) = \frac{1}{K} \sum_{k=1}^{K} \mathrm{Var}\left(\left\{ z_{m,c}^{(t)} : c \in \mathcal{G}_{m,k}(\mathbf{\Pi}_m) \right\} \right). \tag{49}$$

A lower value indicates that channels within each group have more consistent dynamic ranges and are therefore more suitable for sharing quantization parameters.

Similar to the robust optimization in CLDC, optimizing only the average group variance may hide difficult recurrent states. Therefore, we define a cross-loop robust group objective:

$$\mathcal{L}_{\mathrm{grp}}(\mathbf{\Pi}_m) = \tau_g \log \left(\frac{1}{T} \sum_{t=1}^{T} \exp \left(\frac{\mathcal{C}_m^{(t)}(\mathbf{\Pi}_m)}{\tau_g} \right) \right). \tag{50}$$

The optimal channel ordering is obtained by solving the discrete optimization problem:

$$\mathbf{\Pi}_m^{\star} = \arg \min_{\mathbf{\Pi} \in \mathcal{P}_{d_i}} \mathcal{L}_{\mathrm{grp}}(\mathbf{\Pi}), \tag{51}$$

where $\mathcal{P}_{d_i}$ denotes the set of all d_i-dimensional permutation matrices.

Directly searching the permutation space is computationally infeasible due to its combinatorial size. Therefore, we adopt a two-stage optimization strategy. First, we construct an initialization by sorting channels according to their worst-case dynamic range:

$$\rho_{m,c} = \max_t a_{m,c}^{(t)}, \tag{52}$$

and assign channels according to the ordering of $\rho_{m,c}$ to obtain an initial permutation $\mathbf{\Pi}_m^{(0)}$.

Starting from this initialization, we further perform local pairwise-swap refinement. At each step, two channels from different quantization groups are exchanged, and the update is accepted only when it decreases $\mathcal{L}_{\mathrm{grp}}(\mathbf{\Pi}_m)$. This procedure preserves a hard permutation structure and avoids introducing additional continuous parameters during deployment.

Finally, combining the optimized continuous canonical geometry and discrete group-aware ordering, CLDC obtains the final shared transformation:

$$\mathbf{P}_m^{\star} = \mathbf{D}_m^{\star} \mathbf{R}_m^{\star} \mathbf{\Pi}_m^{\star}. \tag{53}$$

The corresponding transformed activation and weight are defined as

$$\widetilde{\mathbf{X}}_m^{(t)} = \mathbf{X}_m^{(t)} \mathbf{P}_m^{\star}, \qquad \widetilde{\mathbf{W}}_m = \mathbf{W}_m (\mathbf{P}_m^{\star})^{-\top}. \tag{54}$$

Since $\mathbf{\Pi}_m^{\star}$ is an orthogonal permutation matrix, the final transformation preserves the original full-precision computation:

$$\widetilde{\mathbf{X}}_m^{(t)}\widetilde{\mathbf{W}}_m^{\top} = \mathbf{X}_m^{(t)}\mathbf{W}_m^{\top}, \quad \forall t. \tag{55}$$

The resulting permutation is shared across all recurrent states, ensuring that the quantization layout remains consistent with the parameter-sharing design of LoopLMs.

A.2.4 Shapley-Style Loop Sensitivity Estimation

The goal of ILPA is to estimate the task-level sensitivity of each loop to activation precision reduction. Unlike layer-wise quantization sensitivity, the importance of a loop in LoopLMs depends on the precision states of other loops because quantization perturbations propagate through the recursive trajectory. Therefore, we evaluate each loop under different recursive precision contexts.

Let $\mathcal{T} = \{1, \ldots, T\}$ denote all loops. For a precision configuration $\mathcal{S} \subseteq \mathcal{T}$, the loops in $\mathcal{S}$ are assigned low activation precision b_{L}, while the remaining loops use high activation precision b_{H}. The corresponding precision assignment is

$$b_t(\mathcal{S}) = \begin{cases} b_{\mathrm{L}}, & t \in \mathcal{S}, \\ b_{\mathrm{H}}, & t \notin \mathcal{S}. \end{cases} \tag{56}$$

Given a precision configuration $\mathcal{S}$, the model executes a complete recursive forward pass with the selected activation precision assignment. The task-level cost is defined as the calibration-set token-level negative log-likelihood:

$$v(\mathcal{S}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\mathrm{cal}}}\left[-\frac{1}{|\mathbf{x}|}\sum_n \log p_{\mathcal{S}}(x_{n+1}|x_{\le n})\right]. \tag{57}$$

This trajectory-level objective differs from local reconstruction metrics because it directly measures the influence of precision decisions on the final model behavior.

To estimate the marginal contribution of each loop, we consider a permutation $\pi \in \mathfrak{S}_T$, where $\mathfrak{S}_T$ denotes the set of all permutations over T loops. For a given permutation, let $\mathcal{S}_t^{(\pi)}$ denote the set of loops appearing before loop t and already assigned low precision. The marginal degradation caused by additionally reducing the precision of loop t is defined as

$$\Delta_t^{(\pi)} = v(\mathcal{S}_t^{(\pi)} \cup \{t\}) - v(\mathcal{S}_t^{(\pi)}). \tag{58}$$

The above quantity measures the additional task degradation introduced by loop t under a specific recursive precision context. Since different permutations correspond to different combinations of preceding low-precision loops, averaging over all permutations provides a context-independent estimation of loop importance.

Following the Shapley value formulation, the loop sensitivity is computed as

$$\phi_t = \frac{1}{T!}\sum_{\pi\in\mathfrak{S}_T} \Delta_t^{(\pi)}. \tag{59}$$

A larger ϕ_t indicates that reducing the precision of loop t consistently introduces larger task degradation across different recursive contexts. Therefore, ϕ_t provides a discrete task-level estimate of the first-order sensitivity induced by reducing the precision of loop t.

In practice, enumerating all $T!$ permutations is unnecessary. Instead, ILPA samples a subset of permutations and estimates the expectation using Monte Carlo approximation:

$$\phi_t \approx \frac{1}{M}\sum_{m=1}^{M} \Delta_t^{(\pi_m)}, \tag{60}$$

where $\{\pi_m\}_{m=1}^{M}$ are randomly sampled loop permutations. The sampled estimation provides a scalable approximation while preserving the context-aware sensitivity property required for loop-wise precision allocation.

A.2.5 Recursive Interaction Modeling

The sensitivity term ϕ_t in Appendix A.2.4 measures the individual contribution of each loop when its activation precision is reduced. However, the recursive structure of LoopLMs introduces additional coupling effects: two loops that are individually insensitive may still cause significant degradation when simultaneously quantized at low precision because their perturbations interact along the recursive trajectory. Therefore, ILPA explicitly models the pairwise interaction among loops to capture non-additive task effects under discrete precision perturbations.

For two loops i and j, we first define the joint marginal degradation under a recursive precision context. Given a permutation $\pi \in \mathfrak{S}_T$, let $\mathcal{S}^{(\pi)}$ denote the set of loops assigned low precision before evaluating the joint contribution of loops i and j. The joint marginal degradation is defined as

$$\Delta_{ij}^{(\pi)} = v(\mathcal{S}^{(\pi)} \cup \{i, j\}) - v(\mathcal{S}^{(\pi)}). \tag{61}$$

The individual marginal degradations of loops i and j under the same context are

$$\Delta_{i}^{(\pi)} = v(\mathcal{S}^{(\pi)} \cup \{i\}) - v(\mathcal{S}^{(\pi)}), \tag{62}$$

$$\Delta_{j}^{(\pi)} = v(\mathcal{S}^{(\pi)} \cup \{j\}) - v(\mathcal{S}^{(\pi)}). \tag{63}$$

If two loops contribute independently, their joint degradation should approximately equal the sum of individual degradations. Therefore, we define the additional recursive interaction as the non-additive component:

$$\Gamma_{ij} = \frac{1}{T!} \sum_{\pi \in \mathfrak{S}_T} \left(\Delta_{ij}^{(\pi)} - \Delta_{i}^{(\pi)} - \Delta_{j}^{(\pi)} \right). \tag{64}$$

A positive Γ_{ij} indicates that jointly reducing the precision of loops i and j introduces stronger-than-additive degradation, implying that their quantization perturbations interact along the recursive trajectory. In contrast, a value close to zero indicates that the two loops behave approximately independently under precision reduction.

The interaction term Γ_{ij} captures the discrete task-level interaction effect induced by jointly reducing the precision of loops i and j. While the second-order formulation in Sec. 4.1 motivates modeling interactions among perturbations injected at different loops, Γ_{ij} measures the corresponding non-additive task degradation under discrete precision configurations.

The above formulation connects the trajectory-level objective $v(\mathcal{S})$ with ILPA's discrete allocation objective. Starting from the second-order perturbation analysis in Sec. 4.1, precision reduction can be viewed as a discrete perturbation variable. The first-order contribution is captured by the marginal task degradation ϕ_t, while the additional interaction among multiple precision reductions is captured by Γ_{ij}. Therefore, ILPA formulates $\mathcal{J}(\mathbf{q})$ as a discrete second-order surrogate of the original trajectory-level objective, enabling efficient precision allocation without exhaustively evaluating all possible precision configurations.

Since different loop pairs may exhibit different degradation scales across recursive precision contexts, directly comparing Γ_{ij} may introduce magnitude bias. We therefore normalize the interaction strength using the empirical variance of marginal degradation:

$$\widetilde{\Gamma}_{ij} = \frac{\Gamma_{ij}}{\sqrt{\sigma_i^2 \sigma_j^2 + \epsilon}}, \tag{65}$$

where σ_i^2 and σ_j^2 are estimated as

$$\sigma_i^2 = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \left(\Delta_i^{(\pi)} - \phi_i \right)^2, \tag{66}$$

$$\sigma_j^2 = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \left(\Delta_j^{(\pi)} - \phi_j \right)^2, \tag{67}$$

where Π denotes the sampled permutation set used for context estimation.

The normalization removes the influence of different marginal degradation scales and makes $\widetilde{\Gamma}_{ij}$ comparable across different loop pairs. The resulting normalized interaction matrix $\widetilde{\Gamma}$ is then used in ILPA's precision allocation objective to model recursive coupling among low-precision loops.

A.2.6 Exact and Greedy Precision Allocation

After estimating loop sensitivity ϕ_t and normalized interaction $\widetilde{\Gamma}_{ij}$, ILPA determines the final activation precision configuration under a fixed bit-width budget. The precision allocation objective is

$$\mathcal{J}(\mathbf{q}) = \sum_{t=1}^{T} q_t \phi_t + \lambda \sum_{i<j} q_i q_j \widetilde{\Gamma}_{ij}, \tag{68}$$

where $q_t = 1$ indicates low precision and $q_t = 0$ high precision. The first term measures individual loop sensitivity, while the second captures additional degradation from interactions among low-precision loops.

The assignment is constrained by the target average activation bit-width:

$$\sum_{t=1}^{T} [b_{\mathrm{H}} - (b_{\mathrm{H}} - b_{\mathrm{L}}) q_t] = T\overline{b}. \tag{69}$$

For small T, we enumerate all feasible binary assignments satisfying the budget and select the configuration with the minimum $\mathcal{J}(\mathbf{q})$, yielding the exact optimum under the estimated objective. As T increases, exhaustive search over up to 2^T assignments becomes expensive. We therefore use a greedy reduction strategy that progressively converts high-precision loops into low-precision loops.

Starting from

$$q_t = 0, \qquad \forall t, \tag{70}$$

we evaluate the marginal objective increase from reducing loop t to low precision:

$$\Delta\mathcal{J}_t = \phi_t + \lambda \sum_{j:q_j=1} \widetilde{\Gamma}_{tj}. \tag{71}$$

The next loop is selected as

$$t^\star = \arg\min_{t:q_t=0} \Delta\mathcal{J}_t, \tag{72}$$

after which $q_{t^\star} = 1$. The process repeats until the bit-width constraint is satisfied, prioritizing loops with smaller task sensitivity and weaker interactions for low-bit execution.

The greedy solver requires at most T selection steps and, with cached sensitivity and interaction values, has overall complexity $\mathcal{O}(T^2)$. It avoids exponential search while preserving the task-aware allocation of ILPA.

A.2.7 LOOP-CONDITIONED CALIBRATION PARAMETERIZATION

After CLDC and ILPA, the shared quantization transformation and loop-wise precision assignment are fixed. LCTC therefore adjusts only lightweight loop-conditioned calibration parameters while leaving quantized weights and backbone parameters unchanged. For module m at loop t, we define

$$\mathbf{\Gamma}_{m,t} = \{\mathbf{c}_{m,t}, \mathbf{g}_{m,t}, \boldsymbol{\beta}_{m,t}\}, \tag{73}$$

where $\mathbf{c}_{m,t}$ controls group-wise activation clipping, while $\mathbf{g}_{m,t}$ and $\boldsymbol{\beta}_{m,t}$ provide group-wise gain and bias correction, respectively. Given the fixed quantized weight $\widehat{\mathbf{W}}_m$ and the activation precision b_t selected by ILPA, the calibrated mapping is

$$\widehat{\mathbf{y}}_{m,t} = \mathbf{g}_{m,t} \odot \left[\widehat{\mathbf{W}}_m Q_{b_t}\left(\mathbf{x}_{m,t}; \mathbf{c}_{m,t}\right) + \mathbf{b}_m\right] + \boldsymbol{\beta}_{m,t}. \tag{74}$$

The calibration parameters depend only on the physical module and loop index, not on individual tokens or inference histories. They are therefore stored as static lookup parameters and introduce no token-wise state cache or online optimization during inference. This parameterization changes only the loop-conditioned quantized response and preserves the quantization structure determined by CLDC and ILPA.

A.2.8 TASK-SPACE DEFECT TRANSPORT

The objective of LCTC is to calibrate quantization responses according to their actual impact on the final task output rather than minimizing intermediate hidden-state errors. To achieve this goal, we explicitly transport loop-wise quantization defects from hidden-state space into task space.

For the t-th loop, we denote the full-precision recursive computation as $\mathcal{F}_{\theta,t}$ and the loop-conditioned quantized computation as $\widehat{\mathcal{F}}_{t,\mathbf{\Gamma}}$. Let $\mathcal{S}_{t+1}(\cdot)$ denote the suffix operator that maps the output state of loop t through the remaining full-precision recursive computation and the final language modeling head:

$$\mathcal{S}_{t+1}(\mathbf{h}_t) = \mathrm{LMHead}\left(\mathcal{F}_{\theta,T} \circ \cdots \circ \mathcal{F}_{\theta,t+1}(\mathbf{h}_t)\right). \tag{75}$$

Similarly, the full suffix mapping starting from the input state of loop t is defined as

$$\mathcal{S}_t(\mathbf{h}_{t-1}) = \mathcal{S}_{t+1}\left(\mathcal{F}_{\theta,t}(\mathbf{h}_{t-1})\right). \tag{76}$$

Given an input state $\mathbf{h}$, the task-space quantization defect introduced by loop t is defined as

$$\mathbf{d}_t(\mathbf{h}; \mathbf{\Gamma}) = \mathcal{S}_{t+1}\left(\widehat{\mathcal{F}}_{t,\mathbf{\Gamma}}(\mathbf{h})\right) - \mathcal{S}_t(\mathbf{h}). \tag{77}$$

This formulation replaces only the current loop with the quantized operator while keeping subsequent computation in full precision. It is designed to isolate the task-space contribution of the current loop's quantization error rather than reproduce the final deployed trajectory. The complete low-bit recursive behavior is jointly optimized through the teacher alignment and language modeling objectives in Eq. equation **??**.

Let $\mathbf{H}^{(t-1)}$ denote the hidden state of the full-precision trajectory before loop t. The clean-state task defect is defined as

$$\mathbf{a}_t = \mathbf{d}_t(\mathbf{H}^{(t-1)}; \mathbf{\Gamma}). \tag{78}$$

Although $\mathbf{a}_t$ measures the direct task impact of loop t under its clean input state, the actual quantized model does not arrive at loop t through the full-precision trajectory. Instead, previous quantization errors accumulate and introduce a trajectory drift:

$$\boldsymbol{\delta}_{t-1} = \widehat{\mathbf{H}}^{(t-1)} - \mathbf{H}^{(t-1)}. \tag{79}$$

The inherited drift changes the input distribution encountered by the current loop and therefore modifies its quantization response. To capture this effect without storing historical token-level states, LCTC evaluates the directional response of the task-space defect along the inherited drift direction.

Specifically, we define the task-space secant as

$$\mathbf{s}_t(\alpha) = \frac{\mathbf{d}_t(\mathbf{H}^{(t-1)} + \alpha\boldsymbol{\delta}_{t-1}; \boldsymbol{\Gamma}) - \mathbf{d}_t(\mathbf{H}^{(t-1)}; \boldsymbol{\Gamma})}{\alpha}. \tag{80}$$

The secant formulation approximates the first-order sensitivity of the task-space defect to the inherited trajectory deviation. By applying the first-order Taylor expansion of $\mathbf{d}_t$ around $\mathbf{H}^{(t-1)}$, we obtain

$$\mathbf{d}_t(\mathbf{H}^{(t-1)} + \alpha\boldsymbol{\delta}_{t-1}; \boldsymbol{\Gamma}) = \mathbf{d}_t(\mathbf{H}^{(t-1)}; \boldsymbol{\Gamma}) + \alpha\mathbf{J}_{\mathbf{d}_t}(\mathbf{H}^{(t-1)})\boldsymbol{\delta}_{t-1} + \mathcal{O}(\alpha^2), \tag{81}$$

where $\mathbf{J}_{\mathbf{d}_t}$ denotes the Jacobian of the task-space defect with respect to the recurrent state.

Substituting the above expansion into the secant definition gives

$$\mathbf{s}_t(\alpha) = \mathbf{J}_{\mathbf{d}_t}(\mathbf{H}^{(t-1)})\boldsymbol{\delta}_{t-1} + \mathcal{O}(\alpha). \tag{82}$$

Therefore, when $\alpha \to 0$,

$$\lim_{\alpha \to 0} \mathbf{s}_t(\alpha) = \mathbf{J}_{\mathbf{d}_t}(\mathbf{H}^{(t-1)})\boldsymbol{\delta}_{t-1}. \tag{83}$$

This result shows that the secant term captures how inherited trajectory drift is amplified by the current loop's quantization response in task space. Therefore, $\mathbf{s}_t(\alpha)$ provides a first-order approximation of the trajectory-dependent error $\boldsymbol{\epsilon}_t^{\mathrm{traj}}$ introduced in Sec. 4.1.

For the first loop, no previous quantization perturbation exists, resulting in

$$\boldsymbol{\delta}_0 = \mathbf{0}, \tag{84}$$

and consequently the trajectory-dependent term naturally becomes zero. This property enables LCTC to distinguish direct loop-wise quantization defects from errors accumulated through recursive computation.

A.2.9 Fisher-Weighted Trajectory Calibration

The goal of LCTC is to optimize lightweight calibration parameters according to their impact on the final prediction distribution. However, logit perturbations with identical Euclidean magnitude may lead to different prediction changes because the output probability distribution is highly non-uniform. Therefore, LCTC adopts a Fisher-weighted metric to measure task-space defects in a distribution-aware manner.

Given the teacher logits $\mathbf{z} \in \mathbb{R}^V$, where V denotes the vocabulary size, the corresponding probability distribution is

$$\mathbf{p} = \mathrm{softmax}(\mathbf{z}). \tag{85}$$

For categorical distributions, the Fisher information matrix with respect to the logits is defined as

$$\mathbf{F}(\mathbf{p}) = \mathrm{Diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top. \tag{86}$$

For an arbitrary logit perturbation $\mathbf{v}$, the Fisher energy is defined as

$$\mathcal{E}_{\mathrm{F}}(\mathbf{v}) = \mathbf{v}^{\top}\mathbf{F}(\mathbf{p})\mathbf{v}. \tag{87}$$

This metric measures the perturbation magnitude under the local geometry of the prediction distribution rather than the Euclidean space. To show its connection with task-level prediction change, we consider the KL divergence between the teacher distribution and the perturbed distribution:

$$D_{\mathrm{KL}}\left(\mathbf{p} \| \operatorname{softmax}(\mathbf{z}+\mathbf{v})\right). \tag{88}$$

Using the second-order Taylor expansion around $\mathbf{v} = \mathbf{0}$, the first-order term vanishes because the teacher distribution is the local optimum of the KL divergence. Therefore,

$$D_{\mathrm{KL}}\left(\mathbf{p} \| \operatorname{softmax}(\mathbf{z}+\mathbf{v})\right) = \frac{1}{2}\mathbf{v}^{\top}\mathbf{F}(\mathbf{p})\mathbf{v} + \mathcal{O}(\|\mathbf{v}\|^3). \tag{89}$$

Hence, the Fisher energy provides a second-order approximation of the task-level distribution shift induced by logit perturbations.

Based on this metric, LCTC separately measures the direct quantization defect and trajectory-dependent sensitivity. For the clean-state task defect $\mathbf{a}_t$ defined in Appendix A.2.8, the anchor loss is defined as

$$\mathcal{L}_{\mathrm{anchor}} = \mathbb{E}_{t,\mathbf{x}}[\mathcal{E}_{\mathrm{F}}(\mathbf{a}_t)]. \tag{90}$$

This objective encourages each loop-conditioned quantizer to minimize its direct influence on the final prediction distribution.

For the trajectory secant $\mathbf{s}_t(\alpha)$, we define the trajectory calibration loss as

$$\mathcal{L}_{\mathrm{traj}} = \mathbb{E}_{t,\mathbf{x},\alpha}[\mathcal{E}_{\mathrm{F}}(\mathbf{s}_t(\alpha))]. \tag{91}$$

Unlike $\mathcal{L}_{\mathrm{anchor}}$, which focuses on the clean recursive state, $\mathcal{L}_{\mathrm{traj}}$ explicitly constrains the sensitivity of quantization defects to inherited trajectory drift.

Since the two task-space losses may have different scales during optimization, we normalize them before combining with other objectives:

$$\widetilde{\mathcal{L}}_{\mathrm{anchor}} = \frac{\mathcal{L}_{\mathrm{anchor}}}{\operatorname{stopgrad}(\mathcal{L}_{\mathrm{anchor}}) + \epsilon}, \tag{92}$$

$$\widetilde{\mathcal{L}}_{\mathrm{traj}} = \frac{\mathcal{L}_{\mathrm{traj}}}{\operatorname{stopgrad}(\mathcal{L}_{\mathrm{traj}}) + \epsilon}. \tag{93}$$

The complete LCTC objective is formulated as

$$\mathcal{L}_{\mathrm{LCTC}} = \mathcal{L}_{\mathrm{KD}} + \lambda_{\mathrm{NLL}}\mathcal{L}_{\mathrm{NLL}} + \lambda_{\mathrm{a}}\widetilde{\mathcal{L}}_{\mathrm{anchor}} + \lambda_{\mathrm{t}}\widetilde{\mathcal{L}}_{\mathrm{traj}} + \lambda_{\mathrm{reg}}\mathcal{R}(\mathbf{\Gamma}). \tag{94}$$

The regularization term constrains calibration parameters to remain close to their initialization:

$$\mathcal{R}(\mathbf{\Gamma}) = \sum_{m,t}\left[\|\mathbf{c}_{m,t} - \mathbf{c}_{m,t}^{(0)}\|_2^2 + \|\mathbf{g}_{m,t} - \mathbf{1}\|_2^2 + \|\boldsymbol{\beta}_{m,t}\|_2^2\right]. \tag{95}$$

The first term preserves the initial clipping behavior, the second term regularizes gain correction around identity mapping, and the third term prevents excessive bias compensation.

During optimization, all quantized weights, CLDC transformations, ILPA precision assignments, and backbone parameters remain frozen. Only the loop-conditioned calibration parameters $\mathbf{\Gamma}_{m,t}$ are updated, making LCTC a lightweight task-aligned calibration procedure rather than additional model training.

A.3 Additional Experimental Details

A.3.1 CLDC Optimization Details

For each quantized linear layer, CLDC learns a loop-shared diagonal scaling and block-diagonal orthogonal transformation to construct a unified quantization-friendly representation across recursive states. The scaling parameters are parameterized as logarithmic channel vectors, where the actual scaling factors are obtained through exponential mapping and clipped to $[10^{-3}, 10^{3}]$ for numerical stability. The orthogonal transformations are parameterized by block-diagonal Cayley maps with a block size of 32. The Cayley parameters are initialized to zero, corresponding to identity transformations, and incomplete final blocks are masked according to their actual dimensions.

Both scaling and orthogonal transformation parameters are jointly optimized using Adam with a learning rate of 1×10^{-2} for 80 steps. No learning-rate scheduler is applied. Gradient norms are clipped to 1.0, and the parameters achieving the lowest optimization objective are retained.

A.3.2 LCTC Optimization Details

LCTC optimizes only the loop-conditioned calibration parameters $\mathbf{\Gamma}_{m,t}$ while keeping quantized weights, CLDC transformations, ILPA precision assignments, and backbone parameters frozen. The teacher-alignment term is implemented as the token-averaged KL divergence between the teacher and student output distributions without temperature scaling (i.e., $\tau_{\mathrm{KD}} = 1$).

The interpolation coefficient α used in the trajectory-aware defect estimation is sampled from $\{0.5, 1.0, 1.5\}$ with probabilities $(0.25, 0.50, 0.25)$. For the first loop, the trajectory term is disabled since no inherited quantization drift exists.

The clipping parameters are initialized through local reconstruction search. Specifically, for each module and loop, we select the clipping ratio from $\{0.50, 0.51, \ldots, 1.00\}$ that minimizes the reconstruction error on calibration activations. The gain parameters are initialized to one and the bias parameters are initialized to zero.

During calibration, the loss weights remain fixed: $\lambda_{\mathrm{NLL}} = 0.1$, $\lambda_{\mathrm{a}} = 0.1$, $\lambda_{\mathrm{t}} = 0.03$, and $\lambda_{\mathrm{reg}} = 10^{-4}$. The anchor and trajectory losses are normalized by their corresponding pre-calibration reference values before being combined with other optimization objectives.

A.3.3 Deployment Storage Overhead

We further analyze the deployment storage overhead introduced by LoopQuant. The analysis focuses on the additional persistent memory required by the deployed model, including the standard W4 model storage and the static metadata retained by CLDC, ILPA, and LCTC. All calibration-time activation statistics, sensitivity scores, trajectory caches, and optimization states are discarded after quantization and therefore incur no deployment storage.

We first formulate the storage of the W4 base model. Let N_q denote the number of weights in the quantized linear layers, N_s the number of retained BF16 quantization scales, and N_u the number of parameters that remain unquantized in BF16, including embeddings, the LM head, and normalization parameters. The W4 base storage is

$$\mathcal{M}_{\mathrm{W4}} = \frac{N_q}{2} + 2N_s + 2N_u \quad \text{bytes}, \tag{96}$$

where the three terms correspond to packed 4-bit weights, BF16 quantization scales, and unquantized BF16 parameters, respectively. Thus, the actual W4 footprint is larger than the idealized model size obtained by assigning 4 bits to every parameter, since quantization scales and non-quantized parameters remain in BF16.

We next derive the additional storage introduced by LoopQuant. Let $\mathcal{M}$ denote the set of transformed linear modules, d_m the channel dimension associated with module m, and

$$D_{\Sigma} = \sum_{m \in \mathcal{M}} d_m. \tag{97}$$

For CLDC, each transformed module retains a BF16 diagonal scaling vector $\mathbf{D}_m$, a BF16 block-diagonal rotation $\mathbf{R}_m$, and a permutation vector $\mathbf{\Pi}_m$. With a rotation block size B, a module of

dimension d_m contains d_m/B blocks of size $B \times B$, resulting in Bd_m stored rotation elements. Summed over all transformed modules, the corresponding storage is

$$\mathcal{M}_D = 2D_\Sigma, \tag{98}$$
$$\mathcal{M}_R = 2BD_\Sigma, \tag{99}$$
$$\mathcal{M}_\Pi = 8D_\Sigma, \tag{100}$$

bytes, respectively, where $\mathbf{D}$ and $\mathbf{R}$ are stored in BF16 and $\mathbf{\Pi}$ uses 64-bit integer indices. Therefore, the total CLDC overhead is

$$\mathcal{M}_{\text{CLDC}} = (2B + 10)D_\Sigma. \tag{101}$$

ILPA only retains the final loop-wise activation-precision assignment. If N_{mask} uint8 entries are used to encode the assignment, its deployment storage is

$$\mathcal{M}_{\text{ILPA}} = N_{\text{mask}} \quad \text{bytes.} \tag{102}$$

All sensitivity and interaction statistics used to obtain this assignment are discarded after calibration.

For LCTC, each calibrated activation group retains three FP32 parameters corresponding to clipping, gain, and bias. Let $G_{m,t}$ denote the number of retained groups for module m at recurrent step t, and define

$$G_\Sigma = \sum_{m \in \mathcal{M}} \sum_{t=1}^{T} G_{m,t}. \tag{103}$$

Since each group retains three FP32 scalars, the LCTC overhead is

$$\mathcal{M}_{\text{LCTC}} = 3 \times 4 \times G_\Sigma = 12G_\Sigma \quad \text{bytes.} \tag{104}$$

Combining the three components, the total additional deployment storage introduced by LoopQuant is

$$\boxed{\mathcal{M}_{\text{extra}} = (2B + 10)D_\Sigma + N_{\text{mask}} + 12G_\Sigma} \tag{105}$$

and the complete LoopQuant deployment footprint is

$$\boxed{\mathcal{M}_{\text{LoopQuant}} = \frac{N_q}{2} + 2N_s + 2N_u + (2B + 10)D_\Sigma + N_{\text{mask}} + 12G_\Sigma}. \tag{106}$$

Accordingly, the relative storage overhead over the standard W4 model is

$$\rho_{\text{overhead}} = \frac{(2B + 10)D_\Sigma + N_{\text{mask}} + 12G_\Sigma}{\frac{N_q}{2} + 2N_s + 2N_u} \times 100\%. \tag{107}$$

The above formulation shows that the additional storage introduced by LoopQuant scales only with the transformed channel dimensions, the number of recurrent precision assignments, and the number of calibrated groups, rather than with the full number of model weights. Moreover, all retained quantities are static model metadata and require no token-dependent cache or recurrent-history storage during inference. The above analysis focuses on persistent storage overhead; the runtime cost introduced by activation transformations and loop-conditioned calibration is reflected in the end-to-end throughput evaluation in Sec. **??**. W4A4 and W4A6 use the same retained tensor shapes, differing only in the loop-wise activation-precision assignment, and therefore incur essentially identical deployment storage.

A.3.4 Detailed Main Results

Table 5 expands the averaged zero-shot results in Table 1 into task-level accuracy on ARC-Easy, ARC-Challenge, HellaSwag, LAMBADA-openai, LAMBADA-standard, PIQA, and WinoGrande. Under W4A6, LoopQuant achieves the highest average accuracy on all four LoopLMs despite using a lower average activation precision than the W4A8 baselines, outperforming the strongest W4A8 competitor by 1.93, 1.90, 0.64, and 1.36 percentage points on Ouro-1.4B, Ouro-2.6B, Nanbeige4.2-3B, and Huginn-0125, respectively. The advantage becomes more pronounced under the challenging

W4A4 setting, where LoopQuant improves the average accuracy over the strongest W4A4 baseline by 5.91, 5.31, 5.41, and 3.83 percentage points across the four models. The task-level breakdown further shows that these gains are broadly distributed across benchmarks rather than arising from a single task; in particular, LoopQuant consistently achieves leading results across reasoning, language modeling, and completion tasks. Notably, under W4A4, LoopQuant obtains the best quantized accuracy on all seven tasks across all four evaluated LoopLMs, demonstrating substantially stronger robustness to aggressive activation quantization. These detailed results are consistent with the averaged accuracy and perplexity results in Table 1, further validating the effectiveness of LoopQuant for low-bit quantization under recursive computation.

A.3.5 Comparison with LoopQ

LoopQ Fang et al. (2026) is the first work that studies low-bit quantization for recursive Transformer architectures, making it the closest related approach to LoopQuant. Both methods recognize that recursive computation introduces quantization challenges beyond conventional feed-forward Transformers. However, they focus on different aspects of the problem.

LoopQ mainly investigates cross-loop distribution shifts and recursive quantization errors, and improves quantization robustness through loop-aware calibration. In contrast, LoopQuant develops a more comprehensive framework that jointly considers three key challenges in LoopLM quantization: cross-loop activation dynamics, heterogeneous loop sensitivity, and the misalignment between local quantization objectives and recursive task behavior. Specifically, CLDC learns a shared quantization-friendly activation geometry across recurrent states, ILPA performs loop-wise activation precision allocation by modeling task sensitivity and cross-loop interactions, and LCTC further calibrates quantization behavior along the low-bit recursive trajectory.

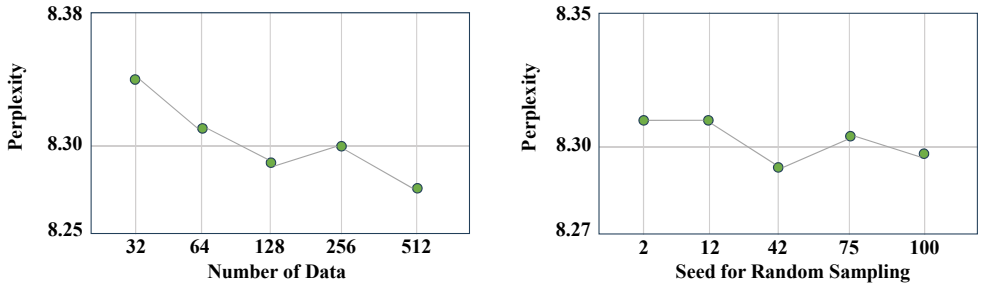
To further compare with LoopQ, we report the results originally published in the LoopQ paper and compare them with our corresponding results. Since LoopQ does not provide publicly available implementation and complete experimental configurations, we cannot perform a fully controlled reproduction. Therefore, we only compare the overlapping models and evaluation metrics reported by both works.

Table 6 presents the comparison between LoopQ and LoopQuant under different low-bit quantization settings. Under the more challenging W4A4 setting, LoopQuant consistently achieves better robustness on both Ouro-1.4B and Ouro-2.6B. For example, on Ouro-2.6B, LoopQuant improves the four zero-shot metrics from 68.60/62.27/56.80/50.34 to 75.07/66.75/65.88/56.09, while reducing WikiText2 perplexity from 15.89 to 8.27. These results indicate that LoopQuant better preserves model capability under extremely low activation precision.

Beyond the quantitative comparison, the two methods adopt different optimization perspectives. LoopQ mainly focuses on mitigating distribution shifts introduced by recursive execution, whereas LoopQuant further models loop-wise precision sensitivity and task-level error propagation along the complete recursive trajectory. By jointly optimizing activation geometry, precision allocation, and trajectory-aware calibration, LoopQuant provides a more comprehensive solution for low-bit LoopLM deployment.

A.3.6 Ablation on Calibration Data

We first investigate the sensitivity of LoopQuant to calibration-set size and sampling randomness. Following the default calibration protocol, we vary the number of calibration samples over $\{32, 64, 128, 256, 512\}$ while keeping all other quantization hyperparameters unchanged. For the seed-sensitivity study, we fix the calibration-set size to the default setting and vary the random sampling seed over $\{2, 12, 42, 75, 100\}$. As shown in Fig. 4(a), LoopQuant remains stable across different calibration sizes, with a perplexity variation of less than 0.05. The relationship is not strictly monotonic: increasing the number of samples generally provides only marginal gains, and the difference between 256 and 512 samples is approximately 0.02 in perplexity. Similarly, Fig. 4(b) shows only minor fluctuations across different sampling seeds, with a maximum difference of approximately 0.02. These results indicate that LoopQuant is insensitive to both calibration-set size and sampling randomness, suggesting that a modest calibration set is sufficient to obtain stable quantization performance.



(a) Change of Number (b) Change of Seed

Figure 4: Sensitivity of LoopQuant to calibration-data configurations. (a) WikiText2 perplexity with different numbers of calibration samples. (b) WikiText2 perplexity under different random sampling seeds. All other quantization hyperparameters are kept unchanged.

We further study the effect of calibration-corpus choice by calibrating LoopQuant on either WikiText2 or C4 under W4A4. As summarized in Table 7, the choice of calibration corpus has only a moderate impact on the averaged zero-shot accuracy, with WikiText2 calibration achieving slightly higher accuracy on both Ouro-1.4B and Ouro-2.6B. In contrast, perplexity exhibits a clear domain-matching effect. Calibrating on WikiText2 yields lower WikiText2 perplexity, reducing it from 10.57 to 9.86 on Ouro-1.4B and from 9.41 to 8.27 on Ouro-2.6B compared with C4 calibration. Conversely, C4 calibration substantially improves C4 perplexity, from 14.00 to 11.86 and from 12.24 to 10.15 on the two models, respectively. These results suggest that the calibration corpus primarily affects the reconstruction quality of the corresponding evaluation domain, while LoopQuant maintains relatively stable downstream zero-shot accuracy across different calibration sources.

A.3.7 Effectiveness of ILPA with Different Quantization Backbones

To further isolate the contribution of ILPA from the complete LoopQuant pipeline, we apply the loop-wise precision allocation generated by ILPA to different quantization backbones on Huginn-0125 ($T = 6$). Specifically, we replace the original uniform activation precision setting with the ILPA-selected allocation while keeping other components of each backbone unchanged. All methods are evaluated under the same average activation precision budget of 6 bits.

As shown in Table 8, applying ILPA consistently improves different quantization backbones under the same activation budget. For QuaRot, ILPA reduces WikiText2 perplexity from 14.02 to 13.59 and improves the average zero-shot accuracy from 46.28% to 47.35%. Similarly, SpinQuant benefits from the ILPA allocation, improving the average accuracy from 45.97% to 47.04%. These results demonstrate that the performance gains of ILPA come from task-aware loop-wise precision allocation rather than simply using a mixed-precision configuration. Combined with CLDC and LCTC, LoopQuant further achieves 12.47 WikiText2 perplexity and 49.09% average accuracy, demonstrating the complementary effects of precision allocation and trajectory-aware calibration.

A.3.8 Ablation on LCTC Calibration Parameters

We further ablate the three learnable calibration parameters in LCTC, namely clipping $\mathbf{c}_{m,t}$, gain $\mathbf{g}_{m,t}$, and bias $\boldsymbol{\beta}_{m,t}$, while keeping CLDC and ILPA fixed. As shown in Table 9, removing LCTC entirely results in WikiText2 and C4 perplexities of 27.05 and 34.17, respectively. Introducing clipping alone substantially improves the reconstruction quality, reducing the two perplexities to 26.48 and 33.38. Adding gain further improves them to 26.09 and 32.74, while incorporating bias yields the full LCTC performance of 25.84 and 32.19. Overall, the complete parameterization reduces WikiText2 and C4 perplexity by 1.21 and 1.98, respectively, compared with the variant without LCTC. These results indicate that the three parameters provide complementary calibration capabilities: clipping controls the effective activation range, while gain and bias further correct the scale and distributional offset accumulated during recursive computation. Importantly, these improvements require only 1.46 MB of additional storage, corresponding to approximately 0.08% of the W4 base-model footprint, demonstrating that LCTC provides consistent performance gains with negligible deployment overhead.

A.3.9 LAYER-WISE QUANTIZATION ERROR DURING REASONING

To further understand how quantization errors propagate through recursive reasoning, we analyze the layer-wise representation deviation of different W4A4 methods on GSM8K. For each evaluated reasoning trace, we run the BF16 and quantized models with the same input and record the hidden representation after each decoder layer. Specifically, for decoder layer l, we measure the relative ℓ_2 error with respect to the BF16 reference as

$$e_l = \frac{\left\| \widehat{\mathbf{H}}_l - \mathbf{H}_l \right\|_2}{\|\mathbf{H}_l\|_2}, \tag{108}$$

where $\mathbf{H}_l$ and $\widehat{\mathbf{H}}_l$ denote the BF16 and quantized hidden representations, respectively. We report the mean error collected over the evaluated GSM8K reasoning traces. The dashed vertical lines in Fig. 5 indicate the boundaries between successive recurrent passes in the unrolled computation.

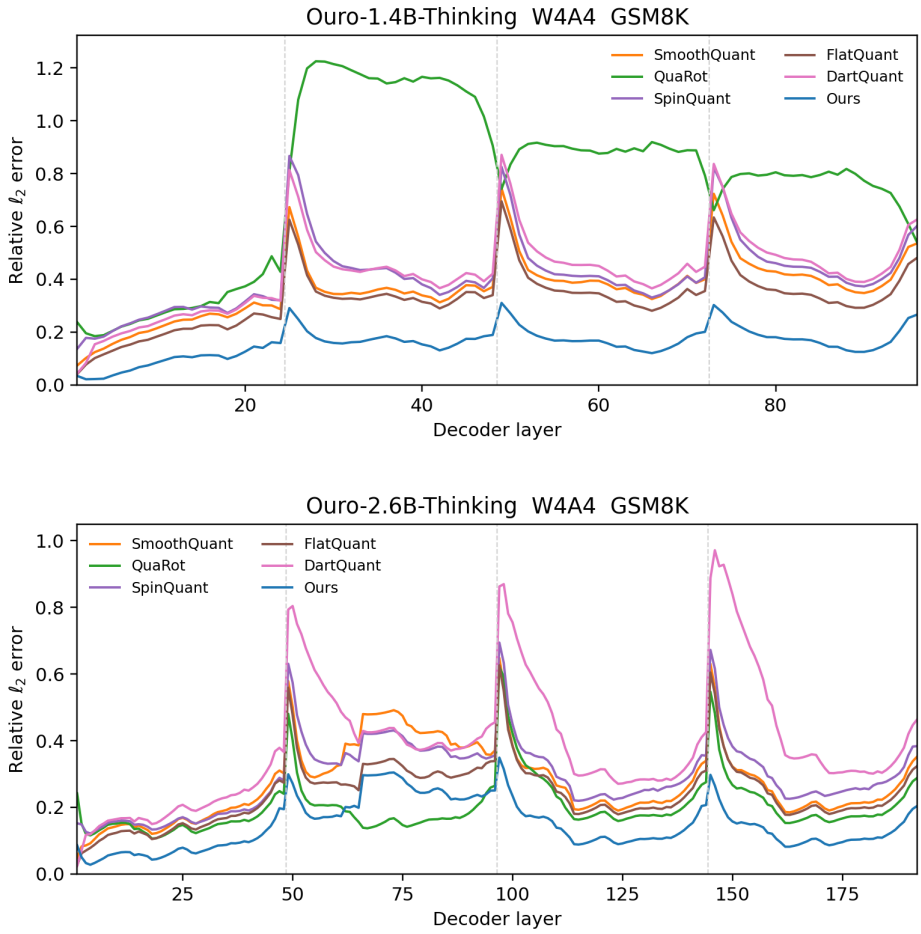


Figure 5: Layer-wise relative ℓ_2 representation error on GSM8K under W4A4 quantization for Ouro-1.4B-thinking (top) and Ouro-2.6B-thinking (bottom). The error is measured against the corresponding BF16 hidden representation at each decoder layer. Dashed vertical lines indicate the boundaries between successive recurrent passes in the unrolled reasoning computation.

As shown in Fig. 5, conventional PTQ methods exhibit pronounced error accumulation and periodic error spikes throughout recursive reasoning. These spikes appear particularly around the beginning of a new recurrent pass, indicating that quantization perturbations inherited from the preceding computation can be re-amplified when the shared decoder is repeatedly applied. This effect is especially evident on Ouro-1.4B-thinking: for example, QuaRot exhibits a sharp increase in relative error after

the first recurrent boundary, with the deviation exceeding 1.2 over several subsequent layers. Other baselines, including SmoothQuant, SpinQuant, FlatQuant, and DartQuant, also show repeated error surges at later recurrent boundaries.

In contrast, LoopQuant substantially suppresses both the overall representation deviation and the magnitude of these periodic error spikes. On Ouro-1.4B-thinking, its relative error remains markedly below the competing methods throughout nearly the entire reasoning trajectory, including immediately after recurrent transitions where other methods deteriorate most severely. A similar trend can be observed on Ouro-2.6B-thinking. Although some baselines remain competitive within individual local regions, LoopQuant consistently limits the peak error around recurrent boundaries and rapidly returns to a lower-error trajectory after each transition. In particular, the error peaks of several baselines approach approximately 0.6–1.0, whereas LoopQuant keeps the corresponding peaks substantially lower.

These observations provide direct evidence for the recursive error-accumulation problem studied in this work. Unlike a conventional feed-forward Transformer, a LoopLM repeatedly applies shared parameters to hidden states that have already been perturbed by preceding quantized computation. Consequently, even moderate local quantization errors can be repeatedly inherited, transformed, and amplified across recurrent passes. The substantially flatter error trajectory of LoopQuant suggests that its loop-aware calibration and precision allocation more effectively preserve the intermediate representations encountered during recursive inference. This layer-wise behavior is also consistent with the stronger W4A4 reasoning accuracy reported in the main results, showing that the benefit of LoopQuant arises not only from reducing local reconstruction error, but also from preventing quantization perturbations from accumulating along the reasoning trajectory.

A.3.10 Activation Distribution Transformation by CLDC

To provide a more direct view of how CLDC reshapes the activation space before low-bit quantization, we visualize the input distributions of different linear modules in the first recurrent layer of Ouro-1.4B. For each module and loop, we collect its input activations over sampled sequences and compute channel-wise statistics across all token positions. Specifically, for every hidden dimension, we report the minimum/maximum, the $0.01/99.99$ percentiles, the $1/99$ percentiles, and the $25/75$ percentiles. Figures 6–9 compare the original module inputs with the corresponding representations after applying CLDC for all recurrent loops.

The raw activations exhibit pronounced channel-wise anisotropy and sparse outliers across different module types. For the attention projections, including q, k, and v, a small subset of channels can reach values several times larger than the central activation range, and these extreme channels become particularly pronounced in later loops. Similar behavior appears in the MLP gate and up projections. An especially severe case occurs at the input of the down projection: in the first loop, isolated channels exceed a magnitude of 20 even though the central percentiles remain concentrated close to zero. Such large discrepancies between a few extreme channels and the majority of activations considerably enlarge the effective quantization range, making uniform low-bit activation quantization inefficient.

After CLDC, the activation distributions become substantially more balanced across hidden dimensions. The extreme channel-wise spikes are strongly suppressed, while the percentile envelopes become smoother and more homogeneous. For $q/k/v$ and the MLP gate/up projections, CLDC consistently converts the highly irregular raw distributions into compact ranges across all recurrent loops. The down projection shows an even more pronounced transformation: the extremely large outliers observed in the raw input are reduced to a compact sub-unit range, with later loops exhibiting particularly narrow distributions. Although the exact activation range remains module- and loop-dependent, CLDC greatly reduces the disparity between outlier channels and the bulk activation distribution.

These visualizations illustrate that the difficulty of activation quantization in LoopLMs is not only caused by the absolute magnitude of activations, but also by their highly heterogeneous channel-wise geometry and its variation across recurrent computation. By regularizing these distributions before quantization, CLDC provides a substantially better-conditioned activation space, reducing the influence of isolated outliers on the quantization range while preserving the relative structure of the dominant activation mass. This observation complements the layer-wise error analysis in

Sec. A.3.9: the more balanced intermediate representations produced by CLDC provide a favorable basis for suppressing the accumulation and re-amplification of quantization errors across recurrent loops.

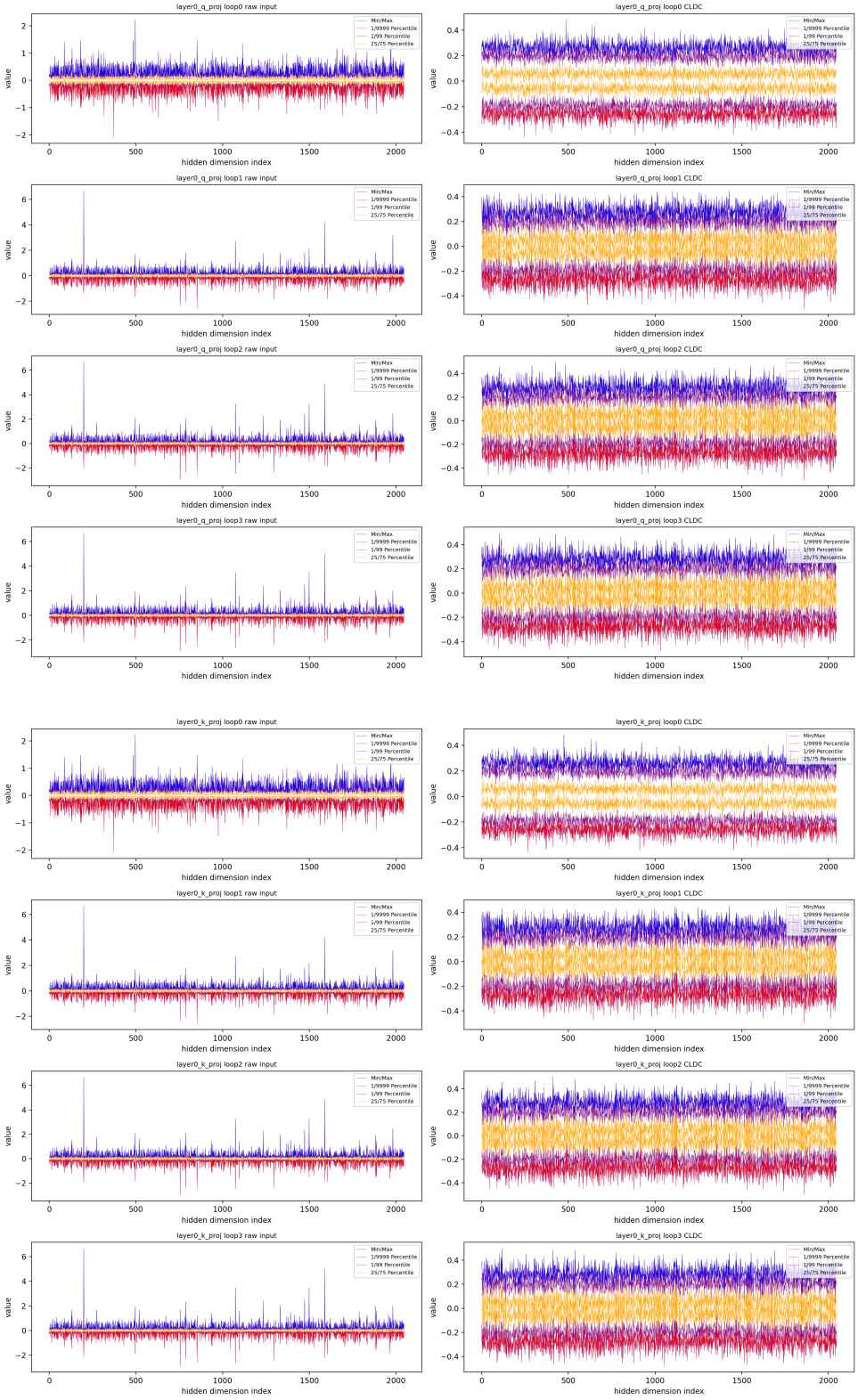


Figure 6: Channel-wise input activation distributions of the q projection (top) and k projection (bottom) in the first recurrent layer of Ouro-1.4B. Within each image, the left and right columns show the raw input and the corresponding representation after CLDC, respectively, while the four rows correspond to different recurrent loops.

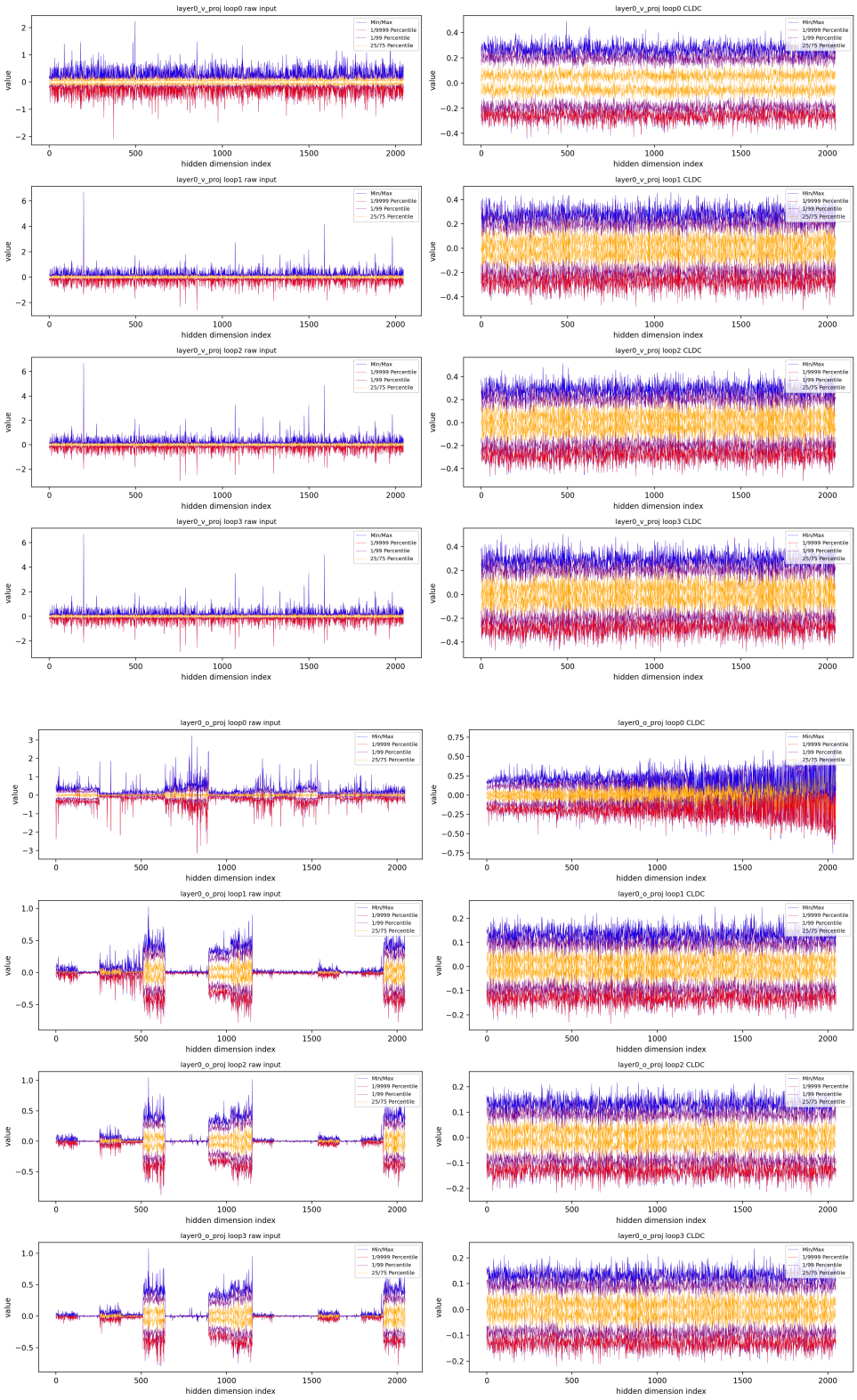


Figure 7: Channel-wise input activation distributions of the v projection (top) and o projection (bottom) in the first recurrent layer of Ouro-1.4B, before and after CLDC. The four rows correspond to different recurrent loops. CLDC substantially suppresses channel-wise outliers and produces more balanced activation ranges across hidden dimensions.

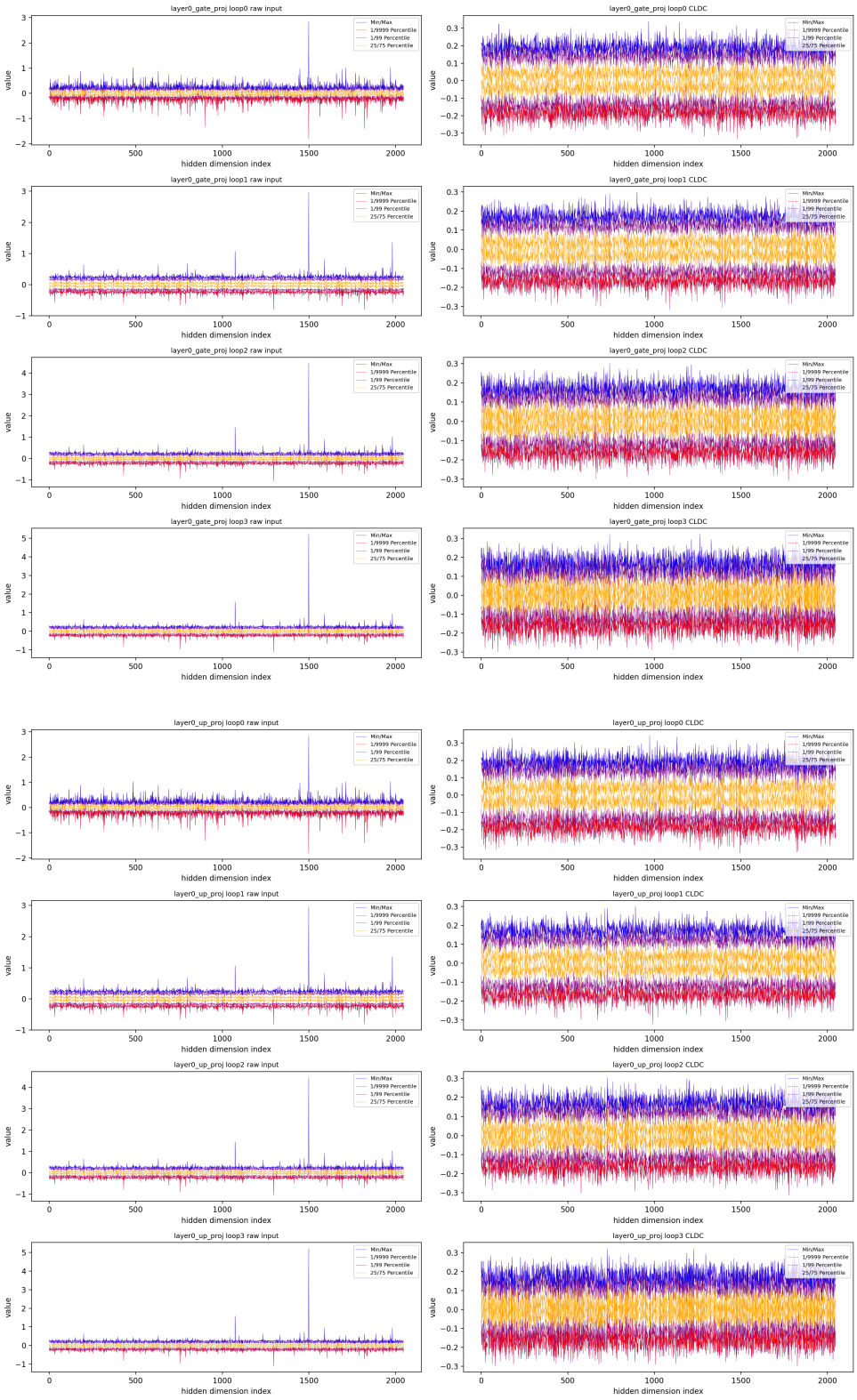


Figure 8: Channel-wise input activation distributions of the MLP gate projection (top) and up projection (bottom) in the first recurrent layer of Ouro-1.4B. Raw activations exhibit sparse but large channel-wise outliers across recurrent loops, whereas CLDC produces substantially more compact and homogeneous percentile envelopes.

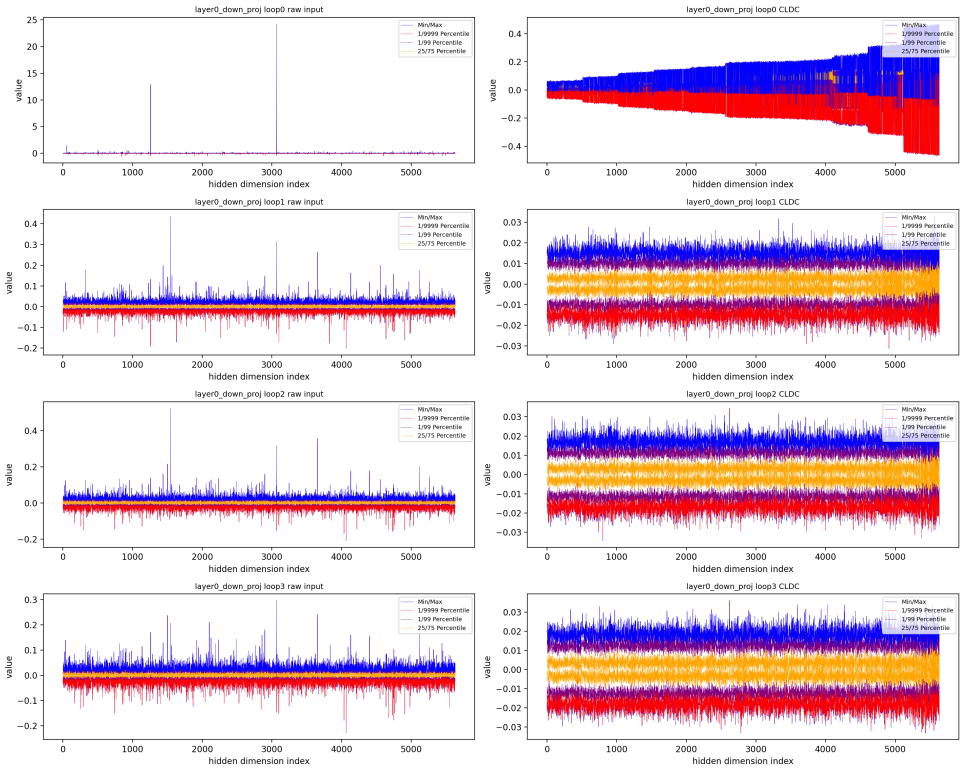


Figure 9: Channel-wise input activation distributions of the MLP down projection in the first recurrent layer of Ouro-1.4B. The left and right columns show the raw input and the representation after CLDC, respectively, across four recurrent loops. The raw input contains highly concentrated channel-wise outliers, including extreme values above 20 in the first loop, while most percentile statistics remain close to zero. CLDC strongly suppresses these extreme channels and transforms the activation space into a substantially more compact and balanced distribution.

Table 5: Detailed zero-shot accuracy on ARC-Easy (AE), ARC-Challenge (AC), HellaSwag (HS), LAMBADA-openai (LO), LAMBADA-standard (LS), PIQA (PQ), and WinoGrande (WG). Bold and underline denote the best and second-best quantized results within each comparison group.

Model	Method	W/A	AE↑	AC↑	HS↑	LO↑	LS↑	PQ↑	WG↑	Avg.[7] ↑
Ouro-1.4B	BF16	16/16	76.81	52.13	71.59	65.17	57.75	78.51	65.98	66.85
	SmoothQuant	4/8	69.42	46.21	66.33	54.69	48.67	75.31	63.90	60.65
	QuaRot	4/8	74.85	50.65	67.55	56.51	51.39	74.28	62.48	62.53
	SpinQuant	4/8	73.59	48.00	68.02	58.99	52.18	75.48	62.56	62.69
	FlatQuant	4/8	73.72	47.49	66.68	55.25	48.57	75.04	62.87	61.37
	DartQuant	4/8	71.27	46.21	65.44	56.02	47.74	75.20	61.77	60.52
	LoopQuant	**4/6**	**77.51**	**51.69**	**68.75**	**60.22**	**53.10**	**76.38**	**64.67**	**64.62**
	SmoothQuant	4/4	65.28	40.37	61.10	45.29	38.91	70.16	58.90	54.29
	QuaRot	4/4	71.34	45.49	63.96	50.98	46.55	71.24	57.33	58.13
	SpinQuant	4/4	69.19	44.29	62.02	47.08	40.52	70.65	56.62	55.77
	FlatQuant	4/4	70.07	45.15	62.46	49.91	43.76	71.57	57.09	57.14
	DartQuant	4/4	61.49	37.47	57.59	43.64	37.78	67.71	54.56	51.46
	LoopQuant	**4/4**	**77.33**	**51.42**	**68.94**	**59.36**	**52.26**	**75.58**	**63.37**	**64.04**
Ouro-2.6B	BF16	16/16	80.93	58.53	76.52	69.98	62.97	80.14	68.75	71.12
	SmoothQuant	4/8	77.21	55.00	72.75	63.20	57.98	77.92	65.80	67.12
	QuaRot	4/8	78.22	56.36	73.45	66.54	59.67	77.98	65.88	68.30
	SpinQuant	4/8	75.65	53.63	73.40	64.80	57.64	78.09	65.80	67.00
	FlatQuant	4/8	77.84	55.42	72.49	63.09	59.01	77.49	65.80	67.31
	DartQuant	4/8	77.84	54.57	71.35	59.67	54.22	76.67	65.25	65.65
	LoopQuant	**4/6**	**80.29**	**57.45**	**76.25**	**68.04**	**60.97**	**80.16**	**68.26**	**70.20**
	SmoothQuant	4/4	72.43	48.30	68.25	56.76	51.46	73.69	61.43	61.76
	QuaRot	4/4	73.02	51.46	69.63	60.54	54.22	74.56	61.60	63.58
	SpinQuant	4/4	70.49	46.17	66.09	50.71	46.30	72.66	60.17	58.94
	FlatQuant	4/4	75.08	51.29	68.33	56.70	52.41	73.20	60.96	62.57
	DartQuant	4/4	68.31	43.01	62.72	48.44	40.19	69.83	57.33	55.69
	LoopQuant	**4/4**	**79.50**	**56.09**	**75.07**	**65.88**	**60.35**	**78.58**	**66.75**	**68.89**
Nanbeige4.2-3B	BF16	16/16	65.28	42.92	66.42	52.65	47.78	72.69	63.05	58.68
	SmoothQuant	4/8	**63.74**	42.37	62.30	49.78	**44.32**	69.98	60.19	56.10
	QuaRot	4/8	61.64	40.06	61.27	45.33	40.48	69.16	58.85	53.83
	SpinQuant	4/8	61.85	39.64	61.39	47.45	39.94	68.84	62.17	54.47
	FlatQuant	4/8	61.64	40.15	62.26	**50.09**	44.01	69.49	58.37	55.14
	DartQuant	4/8	61.09	38.70	60.01	45.02	39.18	68.95	58.46	53.06
	LoopQuant	**4/6**	62.45	**42.82**	**63.95**	49.70	44.08	**71.32**	**62.83**	**56.74**
	SmoothQuant	4/4	59.47	38.92	57.61	43.20	36.91	64.55	54.25	50.70
	QuaRot	4/4	57.36	36.87	57.76	41.00	35.65	64.66	54.88	49.74
	SpinQuant	4/4	53.03	34.57	57.05	39.90	33.92	64.28	55.59	48.33
	FlatQuant	4/4	53.70	35.76	57.96	41.29	34.91	63.19	57.17	49.14
	DartQuant	4/4	50.80	33.97	54.79	37.49	28.49	62.81	54.96	46.19
	LoopQuant	**4/4**	**61.46**	**43.14**	**63.44**	**49.34**	**42.84**	**71.08**	**61.50**	**56.11**
Huginn-0125	BF16	16/16	53.07	32.25	56.22	42.60	39.88	69.48	55.72	49.89
	SmoothQuant	4/8	49.89	29.06	53.31	38.99	37.26	66.72	53.41	46.95
	QuaRot	4/8	50.61	30.25	53.60	40.44	37.76	67.21	54.27	47.73
	SpinQuant	4/8	50.15	29.83	53.68	40.19	37.05	67.04	53.40	47.33
	FlatQuant	4/8	49.98	30.08	53.49	39.90	37.16	66.82	53.72	47.31
	DartQuant	4/8	50.86	30.34	53.51	39.70	36.15	67.37	53.33	47.32
	LoopQuant	**4/6**	**52.38**	**31.38**	**54.92**	**42.25**	**38.60**	**69.72**	**54.35**	**49.09**
	SmoothQuant	4/4	46.59	26.72	50.44	36.64	34.13	63.57	50.62	44.10
	QuaRot	4/4	47.98	27.14	51.06	36.81	34.37	64.23	50.14	44.53
	SpinQuant	4/4	46.67	28.34	49.99	37.04	34.40	63.90	51.17	44.50
	FlatQuant	4/4	47.14	27.48	50.22	36.09	33.98	64.93	52.05	44.56
	DartQuant	4/4	47.14	27.23	50.55	36.99	33.40	63.52	49.59	44.06
	LoopQuant	**4/4**	**51.70**	**30.43**	**54.48**	**40.41**	**37.60**	**68.58**	**55.53**	**48.39**

Table 6: Comparison with LoopQ under different low-bit quantization settings. LoopQ results are directly taken from its original paper due to the lack of public implementation. We only compare overlapping models and evaluation metrics.

Model	W/A	Method	HS↑	WG↑	LB↑	AC↑	WT2↓
Ouro-1.4B	W4A8	LoopQ	**69.54**	64.56	**62.55**	51.45	13.29
	W4A6	LoopQuant	68.75	**64.67**	60.22	**51.69**	**9.48**
	W4A4	LoopQ	65.82	59.51	56.59	48.81	15.81
	W4A4	LoopQuant	**68.94**	**63.37**	**59.36**	**51.42**	**9.86**
Ouro-2.6B	W4A8	LoopQ	75.55	67.80	**68.37**	**58.19**	11.38
	W4A6	LoopQuant	**76.25**	**68.26**	68.04	57.45	**8.04**
	W4A4	LoopQ	68.60	62.27	56.80	50.34	15.89
	W4A4	LoopQuant	**75.07**	**66.75**	**65.88**	**56.09**	**8.27**

Table 7: Ablation on the choice of calibration corpus under W4A4. We compare WikiText2 and C4 calibration data on Ouro-1.4B and Ouro-2.6B. Bold and underline denote the best and second-best results within each model, respectively.

Model	**Calibration Data**	**WikiText2↓**	**C4↓**	**Avg. Acc.↑**
Ouro-1.4B	WikiText2	**9.86**	<u>14.00</u>	**64.04**
	C4	<u>10.57</u>	**11.86**	<u>63.79</u>
Ouro-2.6B	WikiText2	**8.27**	<u>12.24</u>	**68.89**
	C4	<u>9.41</u>	**10.15**	<u>68.07</u>

Table 8: Effectiveness of ILPA with different quantization backbones on Huginn-0125 ($T = 6$) under the same average activation precision budget of 6 bits.

Method	Allocation	Wiki2↓	Avg. Acc.↑
QuaRot	Uniform A6	14.02	46.28
QuaRot + ILPA	[4,4,4,8,8,8]	13.59	47.35
SpinQuant	Uniform A6	14.21	45.97
SpinQuant + ILPA	[4,4,4,8,8,8]	13.63	47.04
LoopQuant	ILPA + CLDC + LCTC	**12.47**	**49.09**

Table 9: Ablation of the three calibration parameters in LCTC under W4A6. We progressively introduce clipping, gain, and bias while keeping CLDC and ILPA fixed. The storage overhead is measured relative to the W4 base model.

W/A	**LCTC**			**Storage Overhead**	**WikiText2↓**	**C4↓**
	$\mathbf{c}_{m,t}$	$\mathbf{g}_{m,t}$	$\boldsymbol{\beta}_{m,t}$			
W4A6	×	×	×	0	27.05	34.17
	✓	×	×	+0.49 MB (0.03%)	26.48	33.38
	✓	✓	×	+0.97 MB (0.05%)	26.09	32.74
	✓	✓	✓	+1.46 MB (0.08%)	**25.84**	**32.19**