DEPTH-AWARE KV CACHE COMPRESSION IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped transformers have advantages over their counterparts by reusing the same layer parameters across recurrences. Yet they typically store separate state caches for each recurrence step. The naive KV caching scheme would incur linearly growing overhead in terms of the number of recurrences, limiting the efficiency gain from looped transformers. There exist several techniques for mitigating this overhead by layer-wise mixed-precision KV quantization according to sensitivity, as well as predicting/reconstructing the states at runtime. Our investigation tries to discover connections between these two approaches through depth analysis. As observed, replacing cached states from earlier recurrences with the last recurrence's state causes less distortion in deep layers than in shallow layers. Contrarily, retaining all states with residual coding while increasing the precision of the first recurrence and reducing that of the last causes more distortion in deep layers than in shallow layers. Motivated by this finding, we introduce a depth-aware allocator that simultaneously selects which recurrent states to retain, along with quantization and coding schemes under certain capacity constraints. The allocator was calibrated on 16 documents and tested on 256 held-out documents for the models Ouro-1.4B and Ouro-2.6B. On the test set above, our allocator decreased KL divergence by 36% and 24%, respectively, compared with layer-wise mixed-precision allocation under the same tight storage budget. Furthermore, without recalibrating, allocations calibrated using 384-token prompts still reduced KL divergence by 57% and 33% w.r.t. the layer-wise allocation scheme on separate test documents with longer prompts of 2,048 tokens.

1 INTRODUCTION

Looped transformers iteratively apply the identical layer stack to every token in order to achieve greater computational depth. Following the initial efforts from the Universal Transformer (Dehghani et al., 2019), subsequent literature establishes a connection between iterative computation and latent-space reasoning, delivering improved reasoning capacity with no extra parameters (Saunshi et al., 2025). This yields a scalability solution—Huginn scales test-time compute by latent iterations (Geiping et al., 2025); Ouro scales looped language models to billions of parameters (Zhu et al., 2025). But it seems parameter sharing fails to address the growth of the cache during inference—the state should still be saved for each recurrence without reuse because the key-value (KV) states at different recurrent positions generally differ. Some literature seeks cache reduction via quantization and redundancy exploitation within KV states. For instance, KIVI, KVQuant and KVarN quantize the cached vectors (Liu et al., 2024b; Hooper et al., 2024; Muller et al., 2026); KVTuner decides per-layer quantization precision based on the sensitivity of layers (Li et al., 2025). Among layers, MiniCache merges similar states in mid/deep layers (Liu et al., 2024a), and cross-layer attention enables adjacent layers to share states (Brandon et al., 2024); AQUA-KV predicts a state from another layer, then encodes the residuals (Shutova et al., 2025). Among recurrences, Ouro and Huginn study cache reutilization (Zhu et al., 2025; Geiping et al., 2025), while LLA learns a low-rank representation of the recurrent trajectory (O'Neill & Reid, 2026). All these approaches suggest a trade-off, keeping more/fewer recurrent states with lower/higher precision, respectively. This paper seeks such a trade-off while being restricted to a budget of cache size. More specifically, we aim at analyzing KV states created at different recurrences within the same layer. At each layer depth we make decisions between keeping all recurrent states, the last or the first and last, while jointly choosing quantization bit-width

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.

and either direct or residual encoding. The option's calibration cost is defined by the change in output KL divergence caused by applying this option at several contiguous layers; the allocator finds allocations minimizing the sum of these costs subject to the overall cache size constraint (Section 3.2). Such an allocator admits a simple dynamic programming formulation without evaluating costs at every joint configuration; this keeps calibration lightweight, and the chosen allocation is reused during test-time inference.

This is what our analysis tells us when it comes to allocations of compression measurements that can be made independently within one layer group. These measurements contain quantization and reconstruction costs within one group. Combining multiple groups induces interactions in the comparison process, which we upper bound under the finite-change assumptions on measured KL divergence. The chosen allocation has lower mean KL than the other in test document collection if its calibration advantage exceeds the bound induced by interactions combined with another bound caused by the change in comparison process from calibration to test. Therefore this sufficient condition doesn't require an accurate prediction of the absolute joint KL divergence value. (See Section 3.4)

Firstly it makes little sense assuming the same layers are preferable for two different compression operations. Empirical results on Ouro-1.4B demonstrate a significant gap in output KL divergence between deep and shallow layers: keeping only the last state causes much less distortion in deep layers than shallow layers, in other words we observe a lower cost in deep layers for cost type one (keeping states); the same operation also causes less distortion in deep layers on Ouro-2.6B. Note further that when keeping the first and last states together most shallow-layer loss on Ouro-1.4B is recovered compared with the last-only policy with the additional storage cost spent, while keeping second/third and last states falls behind, and finally retaining all states with residual coding while moving precision from last to first state causes less distortion in shallow layers than in deep layers on Ouro-1.4B. All these facts tell us something very nontrivial regarding the complexity of optimization process over storage resource at each depth. Both types of cost measurements were collected and used beforehand by our allocator.

Calibration is performed on both models utilizing identical calibration document set of size 16. Next our method evaluates the model over a distinct test dataset of size 256. Baseline retains all recurrence states using the same calibration criterion, solver and budget cap as our method. Our method not only decides which states to keep but also encodes them with proper precision. Under the tested budget cap between uniform two- and three-bit storage that allows for mixing precision among layers, our approach reduces KL divergence by 36% and 24% respectively on Ouro-1.4B and Ouro-2.6B without using additional calibration documents or a larger storage budget (See Section 4.1). Without further calibration, the allocations also achieve lower mean KL than their counterparts on larger prompt token size of 2,048 tokens (See Section 4.2).

Contributions include

1. **Depth Dependent Retention Strategy:** Experimental results show on both Ouro models that replacing KV states of earlier recurrences with the last recurrence's state causes less distortion in deep layers than in shallow layers. Especially on shallow layers of Ouro-1.4B, keeping the first and last states performs better than keeping the second or third together with the last under the same budget.
2. **Adaptive Compression Strategy:** Joint optimization of which states to retain along with precision and encoding at each layer group reduces test KL divergence by 36%/24% relative to layer-wise precision allocation on Ouro-1.4B/Ouro-2.6B under the same tested tight budget cap respectively.
3. **Finite Change Analysis in Measured Allocation:** Sufficient conditions for transferring calibrated gains through combining groups and measuring test documents are established. Model evaluation of successful allocation decomposes the improvement into the sum of single-group improvements minus the between-group interaction penalty.

2 Depth-based Recurrent State Analysis

For a given cache budget, we need to make a trade-off decision on where to store more recurrent states. By focusing on the dimension of depth, we aim to detect where the prior states can be substituted at lower cost such that more storage could be devoted to layers that need more recurrent states.

Recall the setting with a looped transformer having L layers within the shared stack and R recurrences over the stack. Let ℓ be a layer index such that $\ell = 0, \ldots, L-1$ denotes a layer position. A small layer index indicates shallow layers while a large layer index indicates deep layers. The recurrence index $r = 1, \ldots, R$ denotes each pass over the layer stack. Ouro employs $R = 4$. Suppose a fixed block of tokens for one sequence is available. We then define $x_{\ell,r} := (K_{\ell,r}, V_{\ell,r})$ to be the keys and values for layer ℓ and recurrence r.

Usually each recurrence reads its own state. Here instead we consider what happens by retaining different combinations of recurrent states. Denote by $\widehat{x}_{\ell,r}$ the compressed reconstruction of a retained state $x_{\ell,r}$. Having all states means that each recurrence uses its own compressed reconstruction. Only retaining the final state gives last-only sharing, in which every recurrence prior to the last reads $\widehat{x}_{\ell,R}$ in place of $\widehat{x}_{\ell,r}$. Retaining the first and last states means using $\widehat{x}_{\ell,1}$ at recurrence $r = 1$ and $\widehat{x}_{\ell,R}$ at other recurrences. These strategies apply at certain chosen layers; otherwise all recurrent states are retained.

First of all, we evaluate whether replacing early states with the final state costs equally much in shallow and deep layers. To fairly measure it, we apply the same state replacement to equal numbers of layers, with equal precision of retained states and equal storage allocation across both deep and shallow layers. On both Ouro models, the substitution results in significantly less distortion in deep layers, particularly in the Ouro-1.4B model (Figure 1a,b; Table 1).

Table 1: The lower-cost location depends on the compression operation, on the shared 64-document test set. Each row compares deep and shallow interventions with equal layer counts, precision schedules and storage. Ratios are deep/shallow KL, with descriptive 95% intervals; all three pass their respective Holm-corrected tests. Last-only: every retained state is direct three-bit. All retained: selected layers use residual 4332 (first to last: four, three, three, two bits), all others 3333.

Operation	Model	Layers	Deep/shallow KL [95% CI]
Last-only	Ouro-1.4B	10	0.061 [0.045, 0.086]
Last-only	Ouro-2.6B	19	0.51 [0.39, 0.67]
All retained, precision shifted	Ouro-1.4B	8	1.62 [1.42, 1.81]

Since last-only sharing incurs significant loss in shallow layers, we further compare retaining the first state, second state or third state along with the last state. These three choices consume equal amounts of storage capacity and use equivalent precision levels.

On Ouro-1.4B, retaining the first state compensates for the loss in shallow layers more effectively than the other two choices (Figure 1c,d). The three choices use more storage than last-only sharing. For deep layers, we find no clear advantage to retaining the first state over the other two choices.

A different comparison arises when all recurrent states are retained under residual coding. We increase the precision of the first state and decrease that of the last state. On Ouro-1.4B, this precision change incurs greater distortion in deep than shallow layers (Table 1). Hence for Ouro-1.4B, substituting states by the final state is comparatively cheaper in deep layers, while shifting precision in this way is comparatively cheaper in shallow layers.

Taking the above observations into consideration, the allocator compares retaining all states, only the last state, and the first and last states. With output costs measured through model evaluations, the allocator decides how many groups should employ what kind of state-retention scheme, subject to storage budget constraints.

3 Joint allocation of recurrent-state retention and compression

From the exposition in Section 2 one has a great many ways to allocate the cache in light of the effects of different depths. Whether one way incurs smaller costs at the output level highly relies on both concrete layer depths and how compression proceeds. Accordingly we measure the cost of each candidate plan and pick out suitable allocations subject to the overall storage budget.

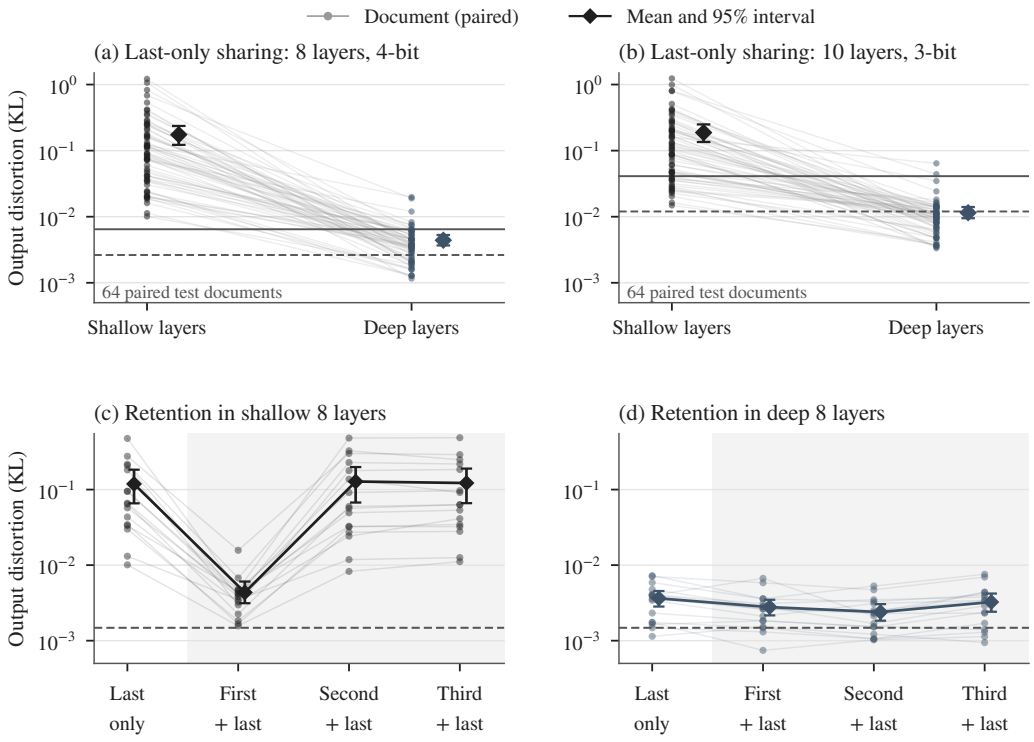


Figure 1: Depth and state-retention interventions on Ouro-1.4B. Each small point gives KL averaged over the scored next-token positions of one document; lower means closer to the uncompressed output distribution. Thin lines pair choices within a document; diamonds show means with descriptive 95% intervals. (a,b) The same 64 documents, with last-only storage in eight layers at four bits or ten layers at three bits. Solid/dashed horizontal lines: uniform direct/residual means at three-bit (a) or two-bit (b) storage. (c,d) A separate 16-document set compares retained recurrences in shallow/deep layers, all at four bits; the dashed baseline retains all states. The three shaded choices use equal storage, 3.7% more than last-only.

We divide a shared layer stack into G contiguous groups, denoted as $g = 1, \ldots, G$. The layers in each group adopt identical allocation strategies. Eight groups are considered in experiments. For Ouro-1.4B there are three layers per group; and six layers per group for Ouro-2.6B.

3.1 Candidate compression plans

At any specific layer ℓ and given a completed token block, denote the unquantized KV state at recurrence r by $x_r = x_{\ell,r}$. A candidate plan specifies which states to reserve; at which precision level; and according to which rules to reconstruct the states that the model will read. These selections uniquely determine a reconstruction of the entire sequence $\widehat{x}_1, \ldots, \widehat{x}_R$. Such a reconstruction rule is referred to as a *complete-history plan*.

Denote quantizing a state to b bits and then reconstructing it by C_b. When keeping all states, we adopt the following two coding schemes for each r:

$$\text{direct: } \widehat{x}_r = C_{b_r}(x_r), \qquad \text{residual: } \widehat{x}_r = \widehat{x}_{r-1} + C_{b_r}(x_r - \widehat{x}_{r-1}), \quad \widehat{x}_0 = 0. \tag{1}$$

Under direct coding the current state is directly quantized, whereas under residual coding only the residual relative to the previous reconstruction needs to be quantized. Reading the current state requires reconstructing the quantized residual, which is added to the preceding reconstruction (Jayant & Noll, 1984; Gersho & Gray, 1992). If retaining some states only, candidates encode all retained states directly; they must always save the last state. Omitted states are replaced by a reconstruction of the last state.

Four recurrences are utilized by Ouro models. Thus, the selection of quantization levels involves four digits where zeros indicate omitted states. So 3333 indicates retaining all four states with three-bit precision; 0003 implies saving the final state with three-bit precision; while 3003 retains the initial and final states, again at three bits. Under the 3003 scheme the four recurrences read $(\widehat{x}_1, \widehat{x}_4, \widehat{x}_4, \widehat{x}_4)$. In the allocator comparisons, candidates retaining all four states use residual coding; other candidates use direct coding. Candidates like 4222 retain the first state with four bits and each remaining state with two bits. They are analyzed in Section 4.1.

We estimate the cost of a complete-history plan here due to the interdependency of selections. For instance, under residual coding, variations of precision on the first state may alter its reconstruction and thus the residual encoded in the next recurrence, which falls into a dependent quantization framework (Ramchandran et al., 1994). In particular, varying the precision on the last state under the 3003 scheme affects the reconstruction read at recurrences 2 to 4. All of these interdependencies are considered in evaluating a complete candidate cost, before approximating their combined effects across groups via (2) below.

3.2 Calibration of candidate costs

Let $\mathcal{O}_g$ be a candidate set associated with group g. Denote an allocation as $a = (a_1, \ldots, a_G)$ where $a_g \in \mathcal{O}_g$ for $g = 1, \ldots, G$. Given a calibration document set $\mathcal{C}$, denote the average next-token KL incurred by running the model with cache allocation a by $\mathcal{L}_{\mathcal{C}}(a)$. In each KL divergence, the uncompressed version is used as the reference. KL averaging is made across the same continuation positions and calibration document collection.

Our calibration starts with the uniform background allocation a^0 for all documents. For the tight-budget comparisons, the background adopts residual coding with two bits on all states. We first evaluate the performance of the background plan; subsequently change the background plan in a specific group to an arbitrary candidate o and evaluate the incurred cost:

$$c_g(o) = \mathcal{L}_{\mathcal{C}}(a^0_{g \leftarrow o}) - \mathcal{L}_{\mathcal{C}}(a^0), \qquad \widehat{\mathcal{L}}_{\mathcal{C}}(a) = \mathcal{L}_{\mathcal{C}}(a^0) + \sum_g c_g(a_g). \tag{2}$$

Here $a^0_{g \leftarrow o}$ means making changes in group g only, using candidate o. The former expression measures the change in KL incurred; and the latter sums up individual change costs with the background KL to obtain the total performance estimate. Negative $c_g(o)$ implies improvement from the change. Each candidate cost includes the complete processes of quantization, reconstruction and model computation for that candidate.

The latter formula in (2) is the cost function approximation used in our allocator heuristic. Though interdependencies within a replaced group have been incorporated above, this approximation neglects interaction effects when simultaneously replacing multiple candidates in different groups. We investigate the impact in Section 3.4.

All candidates' costs are calibrated according to the mean KL divergence calculated on 16 calibration documents. Define $K_g = |\mathcal{O}_g|$ as the cardinality of each candidate set under the assumption that the background candidate $a^0_g \in \mathcal{O}_g$. The calibration step evaluates $1 + \sum_g (K_g - 1)$ compressed allocations on each calibration document. As one can observe, the number of model evaluations scales with the sum of candidate counts, whereas the number of joint allocations is their product, $\prod_g K_g$. The allocator heuristic uses the calibrated cost table to choose an allocation via dynamic programming, without requiring further model evaluations.

3.3 Selecting an Allocation under a Budget Constraint

The distortion cost of a candidate refers to the variation in KL divergence resulting from substituting that candidate for the background option in its group. The storage cost of candidate $o \in \mathcal{O}_g$ in group g, under fixed cache length and block structure, is denoted by $B_g(o)$ and includes quantization metadata. A common uncompressed part is maintained in all allocations with B_0 bytes. Given the storage budget B we have

$$\min_a \sum\nolimits_g c_g(a_g) \quad \text{subject to} \quad B_0 + \sum\nolimits_g B_g(a_g) \le B. \tag{3}$$

The background cost in (2) is constant and has no effect on the selected allocation. Fewer retained states in one group free storage for increased precision in another group. Cost estimates are used to compare benefits.

This multiple-choice knapsack problem is solved by dynamic programming (Kellerer et al., 2004). The solver keeps track of the minimum cumulative cost for every attainable byte total, iteratively considers candidates from each group and performs backtracking to retrieve the optimal additive allocation respecting the budget constraint.

See Algorithm 1 in Appendix B for a detailed description.

Calibration occurs offline and the chosen allocation is reused at the testing stage. Model parameters stay unchanged throughout calibration and testing.

The same algorithm gives a layer-wise baseline by applying constraints on candidate sets: retaining all states and using the same precision across all recurrences within each group. Further expansion allows state-retention choices and mixed precision across recurrences. Calibration documents, costs, allocation procedure and budgets are shared with the baseline.

3.4 Conditions for joint allocation gains

The cost function approximation in (2) uses cost estimates from changing groups one by one. Jointly applying changes to several groups incurs interaction effects among groups. Another change appears during testing of the selected allocations, in which one should consider differences between test and calibration costs. Effects internal to each individual candidate have already been considered in cost estimation. Henceforth, the interaction between selected candidates shall be studied via finite differences in measured costs.

Any allocation a yields its interaction remainder as

$$R_{\mathcal{C}}(a) = \mathcal{L}_{\mathcal{C}}(a) - \widehat{\mathcal{L}}_{\mathcal{C}}(a). \tag{4}$$

The remainder is zero for the background and single-group modifications, since they have been evaluated independently. For any allocations a and b, the estimated gain of a over b is defined by $m_{\mathcal{C}}(a, b) = \widehat{\mathcal{L}}_{\mathcal{C}}(b) - \widehat{\mathcal{L}}_{\mathcal{C}}(a)$. Their measured calibration gain becomes

$$\mathcal{L}_{\mathcal{C}}(b) - \mathcal{L}_{\mathcal{C}}(a) = m_{\mathcal{C}}(a, b) - [R_{\mathcal{C}}(a) - R_{\mathcal{C}}(b)]. \tag{5}$$

The difference $R_{\mathcal{C}}(a) - R_{\mathcal{C}}(b)$ measures how much the gain has been overestimated when positive, or underestimated when negative.

An upper bound on the absolute interaction remainder can be obtained through sequential replacements in order of groups. Let $a^{(g)}$ represent using allocation a in groups 1 to g and the background elsewhere, where $a^{(0)} = a^0$ and $a^{(G)} = a$. Summing differences in measured costs and cost estimates along this replacement path leads to

$$R_{\mathcal{C}}(a) = \sum_{g=1}^{G} \left[\mathcal{L}_{\mathcal{C}}(a^{(g)}) - \mathcal{L}_{\mathcal{C}}(a^{(g-1)}) - c_g(a_g) \right]. \tag{6}$$

Each term represents the change in marginal cost after previous replacements have occurred. Denote the cost induced by switching group g from its background to candidate o in configuration z by

$$d_g(o; z) = \mathcal{L}_{\mathcal{C}}(z_{g \leftarrow o}) - \mathcal{L}_{\mathcal{C}}(z_{g \leftarrow a_g^0}), \qquad c_g(o) = d_g(o; a^0).$$

One makes an assumption regarding the maximum change in this marginal cost after replacing the plan in another group h: its magnitude is bounded by $\eta_{gh} = \eta_{hg} \geq 0$. The bound must hold for all configurations of the other groups, including effects involving multiple interacting groups.

Proposition 3.1 (Allocation gains under finite changes)**.** *Fix the background, candidate sets and calibration documents. Suppose that for all $g \neq h$, candidates $o \in \mathcal{O}_g$, $v \in \mathcal{O}_h$ and configurations z,*

$$\left| d_g(o; z_{h \leftarrow v}) - d_g(o; z_{h \leftarrow a_h^0}) \right| \leq \eta_{gh} = \eta_{hg}, \qquad \eta_{gh} \geq 0.$$

Define the changed-group index set for allocation a as $S(a) = \{g : a_g \neq a_g^0\}$ and its interaction bound sum as

$$E(a) = \sum_{\substack{g<h \\ g,h \in S(a)}} \eta_{gh}. \tag{7}$$

Then $|R_{\mathcal{C}}(a)| \leq E(a)$. If allocation a minimizes the additive objective within budget, any feasible allocation b from the same candidate sets satisfies $m_{\mathcal{C}}(a, b) \geq 0$. Furthermore, if the test cost satisfies

$$|[\mathcal{L}_{\mathcal{T}}(a) - \mathcal{L}_{\mathcal{T}}(b)] - [\mathcal{L}_{\mathcal{C}}(a) - \mathcal{L}_{\mathcal{C}}(b)]| \leq \delta_{a,b}, \tag{8}$$

then

$$\mathcal{L}_{\mathcal{T}}(a) - \mathcal{L}_{\mathcal{T}}(b) \leq -m_{\mathcal{C}}(a, b) + E(a) + E(b) + \delta_{a,b}. \tag{9}$$

Specifically, $m_{\mathcal{C}}(a, b) > E(a) + E(b) + \delta_{a,b}$ guarantees lower test KL under a.

Proof. We omit the terms with $a_g = a_g^0$ in (6). For each of the remaining terms, one switches the preceding nonbackground groups one by one; each change in group h modifies the marginal cost of group g by at most η_{gh} in magnitude. As each changed pair appears only once in computing the sum, the interaction remainder satisfies $|R_{\mathcal{C}}(a)| \leq E(a)$. By optimality of the additive objective, the estimated gain fulfills $m_{\mathcal{C}}(a, b) \geq 0$, while (5) yields $\mathcal{L}_{\mathcal{C}}(a) - \mathcal{L}_{\mathcal{C}}(b) \leq -m_{\mathcal{C}}(a, b) + E(a) + E(b)$. Combining this with the transfer assumption in (8), the claim in (9) follows.

The criterion in Proposition 3.1 requires the advantage estimated via calibration to exceed the sum of the interaction and calibration-to-test discrepancy bounds. The interaction error accounts for effects between selected candidates in different layer groups. The latter bound accounts for changes in documents and continuation positions between calibration and test sets. Although absolute KL divergence values could differ between test and calibration datasets, the comparative advantage may remain stable. The background comparator $b = a^0$ has interaction remainder and $E(b)$ equal to zero. For other comparators, $E(a) + E(b) > |R_{\mathcal{C}}(a) - R_{\mathcal{C}}(b)|$ is possible because the benefit relies on the difference between interaction remainders.

The layer-wise allocation b remains feasible under state-retention expansion; therefore the optimal allocation a in the expanded candidate set must have an additive score no worse than the layer-wise one. The expansion introduces the potential for a gain, but the condition tells us when this potential can indeed materialize.

Interaction and Gain Measurements. In this section we examine gain relations for a fixed Ouro-1.4B allocation under the two-bit storage budget. This includes a comparison against the uniform two-bit residual background, whose interaction remainder vanishes. We evaluate the allocation and single-group changes one by one using the same 16 calibration documents and positions in FP32. This allocation has been selected according to BF16 costs; matched FP32 evaluations measure its actual interaction effects (Table 2).

Table 2: Finite-change decomposition of a fixed Ouro-1.4B two-bit allocation versus uniform residual two-bit. Post-hoc matched evaluation on 16 calibration documents in FP32; means and paired 95% intervals, in 10^{-3} nats. The net gain equals the single-group reduction minus the interaction penalty. No policy is selected using this diagnostic.

Component	Mean [95% CI]
Single-group reduction	10.75 [0.79, 24.91]
Interaction penalty	4.90 [−0.47, 12.91]
Joint net reduction	5.86 [1.09, 12.16]

The measured interaction cost partially diminishes the mean gain; about 55% of the single-group gain survives the allocation combination. The mean gain is large enough to compensate for the interaction loss on these calibration documents.

4 EVALUATION ACROSS MODELS, BUDGETS AND TASKS

Ouro-1.4B and Ouro-2.6B are calibrated individually over the same set of 16 documents, then tested on a disjoint set of 256 documents. Each set consists of half WikiText-2 articles (Merity et al., 2017) and half CPython files (Python Software Foundation, 2025). The test set includes 20 library files and 108 test files in its code portion. Prompts are 384 tokens long, except where noted otherwise.

Each prompt is fed through without compression before compression of the completed cache blocks. Calibration evaluates the continuation positions from 1 to 32. The held-out fidelity tests evaluate on the final 32 positions out of 160, while also exposing states compressed during the continuation. All methods receive the exact same ground-truth continuation tokens. We compare the averaged next-token KL divergence of $\mathrm{KL}(p_{\mathrm{exact}} \| p_{\mathrm{method}})$ (Dutta et al., 2024). All KL evaluations have been done against the corresponding uncompressed model, with a ratio of the methods' average KL divergences below one implying preference for the compression method in the numerator; NLL differences for true continuations have also been recorded. Data storage cost is measured as quantized data plus metadata and retained uncompressed state storage. The Holm correction procedure (Holm, 1979) is applied to the main significance tests in this paper; paired bootstrap intervals (Efron & Tibshirani, 1993) quantify uncertainty in the estimated effect sizes.

4.1 COMPRESSION BENEFITS AND RATIONALE

The following explores the impact of different candidate plans in the algorithm. Three candidate sets are considered in the experiments for Algorithm 1 using the same cost table, optimization algorithm and storage budget. In particular, let L denote the layer-wise set such that each layer group selects residual coding at two, three or four bits while retaining all recurrent states. The retention set S consists of all candidates in L as well as additional plans that retain only the last state or both the first and last states, using direct coding. Further allowing mixed precision across recurrences gives the full set H.

Our budget cap is $U_2 + \lambda(U_3 - U_2)$, in which U_b is the storage amount required by uniform residual b-bit coding at the evaluated sequence length. Tests include $\lambda = 0, 1/4, 1/2$. At $\lambda = 0$, the layer-wise set can only perform residual two-bit coding throughout. Intermediate budget values let the layer-wise set adjust precision upwards within some groups while retaining all recurrent states. At $\lambda = 1/4$, it admits 45 feasible allocations.

At the allocation budget $\lambda = 1/4$, there are reductions of 36% and 24% in KL divergence relative to the layer-wise baseline allocation on Ouro-1.4B and Ouro-2.6B, respectively. Additionally, less storage space in bytes is needed compared with the layer-wise approach. Both KL reductions remain significant after Holm correction. Considering that both approaches use the same calibration data and optimizer configuration, the experimental evaluation shows the benefit brought by the extra candidate plans. Figure 3 in Appendix D shows the results across budgets.

The ablation study suggests a simpler way of gaining the advantage. Descriptively, the retention set offers comparable or greater gains than the full set at most budgets. No additional gain is consistently contributed by mixed precision across recurrences. When reaching $\lambda = 1/2$, gains remain significant against the layer-wise baseline on Ouro-1.4B and are unresolved on Ouro-2.6B. Independent tests with an additional set of 64 documents reveal gains against two-bit residual coding; and the comparison with uniform three-bit coding remains unresolved. This evidence pertains to the case of tight budget. By retaining fewer recurrent states we leave more freedom to increase coding precision elsewhere.

The placement control examines whether the gain is attributable to placing candidate plans at particular depths. Fixed candidate plans and storage costs are adopted while their assignment cycles among the eight layer groups. Using another common 32-document test set, the original placement yields a lower KL divergence than the mean over seven shifted placements in 31 out of 32 cases, for both models. Hence the value of an allocation is depth-dependent. One individual shift has lower mean KL than the original placement for each model.

Table 3: The six primary comparisons of nested candidate sets, a shared test set of 256 documents. L: layer-wise precision; S: adds retention and coding candidates; H: adds mixed precision across recurrences. Byte ratios are A over B for the coded elements (metadata included) and for total storage. An improvement (bold) requires a KL ratio below one and Holm $p < 0.05$. Intervals are unadjusted; ΔNLL is A minus B in 10^{-3} nats.

		Bytes					
A/B	λ	Coded	Total	KL ratio [95% CI]	Lower	Holm p	ΔNLL [95% CI]
Ouro-1.4B							
H/L	$1/4$	0.978	0.994	**0.64** [0.57, 0.72]	204/256	< 0.001	-2.2 [-6.2, 1.8]
H/L	$1/2$	0.992	0.998	**0.78** [0.69, 0.87]	174/256	0.006	1.2 [-2.5, 5.1]
H/S	0	1.002	1.000	1.062 [1.001, 1.124]	101/256	0.084	0.0 [-2.3, 2.3]
Ouro-2.6B							
H/L	$1/4$	0.988	0.997	**0.76** [0.66, 0.86]	177/256	< 0.001	-1.6 [-4.6, 1.4]
H/L	$1/2$	0.997	0.999	1.07 [0.99, 1.16]	113/256	0.085	-1.6 [-3.8, 0.7]
H/S	0	1.000	1.000	0.95 [0.91, 0.99]	145/256	0.052	-0.7 [-2.0, 0.5]

4.2 Reuse of fixed allocations

Allocations remain effective when increasing cache length with prompt size. We apply allocations made with $\lambda = 1/4$ to 2,048-token prompts. Using another shared test set of 32 documents, KL decreases against the layer-wise baseline by 57% and 33% on Ouro-1.4B and Ouro-2.6B, respectively. Both methods utilize allocations produced from calibration over 384-token prompts and satisfy the same budget condition at this longer prompt size.

Free generation evaluation applies the initial two-bit allocations to 1,287 previously unused GSM8K test questions (Cobbe et al., 2021). Task-specific fine-tuning and calibration are not conducted for any method. Differences in accuracy from uniform residual two-bit coding remain unresolved. Equivalence cannot be claimed.

5 Conclusion and future directions

Retention decisions and coding choices incur different KL costs at different depths. Joint allocation of retained states and their coding outperforms layer-wise precision allocation significantly at the tested tight budget on both Ouro models. Gains made with our selected allocations persist on independent test sets and hold up at increased prompt length.

Finite-change analysis yields rigorous relations between calibration performance and joint allocation results. Specifically speaking, a gain is guaranteed when the calibrated additive gain exceeds the sum of bounds on interaction costs and the calibration-to-test gap. As for the specific two-bit allocation of interest, the matched calibration analysis shows that approximately 55% of the sum of single-group gains remains after joint compression across layer groups.

Future research may explore modeling interactions between layer groups when assigning multiple groups simultaneously. Evaluations should extend to long-range free generation, multiple tasks and alternative recurrent architectures (Bae et al., 2025; Wu et al., 2025).

AI USE STATEMENT

Generative AI was used to aid or polish writing and search for related work. Examples of related work found with AI assistance include KIVI (Liu et al., 2024b), KVQuant (Hooper et al., 2024), KVarN (Muller et al., 2026), KVTuner (Li et al., 2025), MiniCache (Liu et al., 2024a), Cross-Layer Attention (Brandon et al., 2024), AQUA-KV (Shutova et al., 2025), XQuant (Tomar et al., 2025), Mixture-of-Recursions (Bae et al., 2025) and Parallel Loop Transformer (Wu et al., 2025).

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, volume 38, pp. 96572–96617, 2025.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, volume 37, pp. 86927–86957, 2024.

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

Victor Conchello Vendrell, Arnau Padres Masdemont, Niccolò Grillo, Jordi Ros-Giralt, Arash Behboodi, and Fabio Valerio Massoli. Memory-efficient looped transformer: Decoupling compute from memory in looped language models. *arXiv preprint arXiv:2605.07721*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Zihao Deng, Sayeh Sharify, Xin Wang, and Michael Orshansky. Mixed-precision quantization for deep vision models with integer quadratic programming. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 293–302, 2019.

Abhinav Dutta, Sanjeev Krishnan, Nipun Kwatra, and Ramachandran Ramjee. Accuracy is not all you need. In *Advances in Neural Information Processing Systems*, volume 37, pp. 124347–124390, 2024.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

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

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 41340–41391, 2025.

Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, 1992.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

N. S. Jayant and Peter Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Yiwu Yao, Wulong Liu, Sinno Jialin Pan, and Mingxuan Yuan. KVTuner: Sensitivity-aware layer-wise mixed-precision KV cache quantization for efficient and nearly lossless LLM inference. In *International Conference on Machine Learning*, pp. 36451–36485, 2025.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 139997–140031, 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, 2024.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Lorenz K. Muller, Philippe Bich, Chiara Boretti, Hyun-Min Chang, Jiawei Zhuang, and Lukas Cavigelli. KVarN: Variance-normalized KV-cache quantization mitigates error accumulation in reasoning tasks. *arXiv preprint arXiv:2606.03458*, 2026.

James O'Neill and Fergal Reid. Looped latent attention: Cross-loop KV compression for looped transformers. *arXiv preprint arXiv:2607.15456*, 2026. Version 2.

Python Software Foundation. CPython 3.11.14 source code, commit cd1c3a6, 2025.

Kannan Ramchandran, Antonio Ortega, and Martin Vetterli. Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders. *IEEE Transactions on Image Processing*, 3(5):533–545, 1994.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025.

Alina Shutova, Vladimir Malinovskii, Vage Egiazarian, Denis Kuznedelev, Denis Mazur, Nikita Surkov, Ivan Ermakov, and Dan Alistarh. Cache me if you must: Adaptive key-value quantization for large language models. In *International Conference on Machine Learning*, pp. 55451–55473, 2025.

Aditya Tomar, Coleman Hooper, Minjae Lee, Haocheng Xi, Rishabh Tiwari, Wonjun Kang, Luca Manolache, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. XQuant: Breaking the memory wall for LLM inference with KV cache rematerialization. *arXiv preprint arXiv:2508.10395*, 2025.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, Siyuan Qiao, and Xingyan Bin. Parallel loop transformer for efficient test-time computation scaling. *arXiv preprint arXiv:2510.24824*, 2025.

Junchen Zhao, Ali Derakhshan, Jayden Hyman, Junhao Dong, Sangeetha Abdu Jyothi, and Ian Harris. CoopQ: Cooperative game inspired layerwise mixed precision quantization for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

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. *arXiv preprint arXiv:2510.25741*, 2025.

A QUADRATIC SCORING OF COMPLETE-HISTORY PLANS

As introduced in Section 3.2, the allocator estimates joint KL costs using measured single-group costs. A quadratic score can be defined using the uncompressed next-token distribution p. Let us write the cache reconstruction error at recurrence r as e_r and its Jacobian to logits at the uncompressed execution as J_r, then linearize the logit change to $Je = \sum_r J_r e_r$ and expand KL to the second order:

$$Q(e) = \tfrac{1}{2}\operatorname{Var}_p(Je) = \tfrac{1}{2}\sum\nolimits_{r,s} e_r^{\mathsf{T}} J_r^{\mathsf{T}} F_p J_s e_s, \qquad F_p = \operatorname{diag}(p) - pp^{\mathsf{T}}. \tag{10}$$

where F_p is the Fisher information matrix of the categorical distribution (Martens, 2020). Here two approximations are made: logits are treated as affine in cache entries, and KL is considered quadratically dependent on the logit change. According to Section 3.1, in a particular layer group the candidate plan makes decisions about all its recurrences. Under the conditions discussed below, the complete reconstruction error of the candidate plan can actually serve as input into quadratic scoring.

Proposition A.1 (Complete Histories Give a Pairwise Objective)**.** *Organize the layers into layer groups indexed by g, where each group selects a_g among K_g complete-history plans. Assume that the cache target states derive from a common uncompressed prompt while each group reconstructs using only its chosen candidate plan. Denote by $e_g(a_g)$ the error vector that takes nonzero values only within group g, such that $e(a) = \sum_g e_g(a_g)$. With the model Jacobian J and next-token distribution p fixed, define $u_g(a_g) = F_p^{1/2} J e_g(a_g)$. As a consequence, we obtain the following expression:*

$$Q(a) = \tfrac{1}{2}\sum_g \|u_g(a_g)\|^2 + \sum_{g<h} \langle u_g(a_g), u_h(a_h)\rangle\,, \tag{11}$$

The objective function has a pairwise structure with respect to the one-hot encoded candidate selection vector for each group. Based upon a baseline candidate a_g^0 for each group, $1 + \sum_g (K_g - 1)$ logit directional derivatives suffice to determine $Q(a)$ for all $\prod_g K_g$ allocations.

Proof. Picking a baseline candidate a_g^0 for each group gives rise to baseline error $e^0 = \sum_g e_g(a_g^0)$ and deviation $\delta e_g = e_g(a_g) - e_g(a_g^0)$. Since each group acts independently for reconstruction of its cache, we obtain $e(a) = e^0 + \sum_g \delta e_g$. Expanding $\frac{1}{2}\|\sum_g \delta u_g + u^0\|^2$ with $u^0 = F_p^{1/2} J e^0$ and $\delta u_g = F_p^{1/2} J \delta e_g$ introduces a constant term, single-group terms and interaction terms between two groups g, h. Computing these terms involves one directional derivative along the baseline error e^0 and one along each δe_g for nonbaseline plans. With multiple prediction positions and weights $w_t \geq 0$ satisfying $\sum_t w_t = 1$, one stacks $\sqrt{w_t} F_{p_t}^{1/2} J_t$ before computing the squared norm.

Fixing the shared uncompressed prompt fixes the target cache states. Under group-local reconstruction, all errors in a nonbaseline candidate plan are determined by its baseline error and deviation. Through the Jacobian, these errors correspond to first-order changes in the model output logit vector, hence the dot product captures the pairwise interaction effect of different cache reconstruction changes in the quadratic score. In this manner one can obtain the overall quadratic score without evaluation of each joint allocation, whose usefulness depends on accurately estimating measured KL.

B CALIBRATION AND ALLOCATION ALGORITHM

The procedure uses $\mathrm{Eval}(a; \mathcal{C}) = \mathcal{L}_{\mathcal{C}}(a)$ for candidate evaluation. Remaining tie cases use the fixed candidate order across groups.

This multiple-choice knapsack problem can be solved via dynamic programming (Kellerer et al., 2004). One starts with no groups and no storage used, subsequently incorporating a new group. Denote the minimal cost summation achieved by using exactly q bytes with the first g groups as $D_g(q)$. For group g and a chosen candidate o, the previous $g-1$ groups must consume $q - B_g(o)$ bytes, whose optimal cost is already known from D_{g-1}. Then

$$D_g(q) = \min_{\substack{o \in \mathcal{O}_g \\ B_g(o) \le q}} \left[D_{g-1}\big(q - B_g(o)\big) + c_g(o) \right], \quad D_0(0) = 0, \quad D_0(q > 0) = +\infty. \tag{12}$$

One keeps track of only reachable byte totals no greater than $B - B_0$. After considering all groups, one picks the lowest-cost byte total within the budget; and traces its history to recover the chosen candidates.

Algorithm 1 Calibration and allocation of recurrent KV compression

Require: Calibration documents $\mathcal{C}$; groups $g = 1, \ldots, G$; ordered candidate sets $\mathcal{O}_g$ containing background plans a_g^0; total budget $B \ge B_0$; common bytes B_0 and candidate bytes $B_g(o)$.
Ensure: One compression plan $a_g^\star$ per group within budget, or infeasibility.

Record uncompressed prompt states and reference predictions on $\mathcal{C}$.
$\ell_0 \leftarrow \mathrm{Eval}(a^0; \mathcal{C})$.
for $g = 1, \ldots, G$ **do**
    $c_g(a_g^0) \leftarrow 0$.
    for $o \in \mathcal{O}_g \setminus \{a_g^0\}$ **do**
        $c_g(o) \leftarrow \mathrm{Eval}(a^0_{g \leftarrow o}; \mathcal{C}) - \ell_0$.
    end for
end for
Initialize $D_0(0) \leftarrow 0$; all other entries have cost $+\infty$.
for $g = 1, \ldots, G$ **do**
    Apply (12) to reachable byte totals $q \le B - B_0$.
    Store a minimizing candidate and its predecessor for each finite entry.
end for
if no finite entry remains **then**
    return infeasible.
end if
Choose $q^\star$ minimizing $(D_G(q), q)$ over finite entries. ▷ cost, then bytes
Backtrack from $(G, q^\star)$ to obtain $a^\star = (a_1^\star, \ldots, a_G^\star)$.
return $a^\star$.

An empty minimum has cost $+\infty$.

C RELATED WORK

KV compression techniques vary both in the precision they assign and the states they represent. KIVI, KVQuant and KVarN compress the cache vectors (Liu et al., 2024b; Hooper et al., 2024; Muller et al., 2026). KVTuner determines KV precision for each layer based on sensitivity (Li et al., 2025). MiniCache merges similar states across layers (Liu et al., 2024a). Cross-Layer Attention trains adjacent layers to share states (Brandon et al., 2024). AQUA-KV learns cross-layer predictors and codes prediction residuals (Shutova et al., 2025). XQuant-CL quantizes differences between layer inputs and rematerializes KV states accordingly (Tomar et al., 2025). Our candidate plans reconstruct the recurrent states within each layer and are selected according to depth. The allocator compares joint allocations under fixed calibration routines and storage budgets. The nested-set ablation separates the effects of adding retention and coding choices from those of adding mixed precision across recurrences.

In looped architectures, Mixture-of-Recursions adopts KV state sharing from the first recurrence across subsequent recurrences (Bae et al., 2025). PLT combines first-recurrence KV reuse and

gated local attention (Wu et al., 2025). LLA fits a latent cache codec over a fixed looped model (O'Neill & Reid, 2026). MELT fine-tunes Ouro and introduces a gated KV state shared across loops (Conchello Vendrell et al., 2026). Our method freezes the pretrained weights and allocates between retained states and reconstructed ones with different precision levels according to measured KL costs and a storage budget. The finite-change analysis gives conditions for these joint allocations to improve KL despite interactions between layer groups. Another study examines post-training weight and activation quantization of looped models (LoopQ) and accumulated errors across recurrences (Fang et al., 2026).

Residual coding uses the classical DPCM technique (Jayant & Noll, 1984; Gersho & Gray, 1992). Dependent quantization motivates choosing complete reconstruction histories (Ramchandran et al., 1994). Here we take residuals relative to the immediately preceding reconstruction within the same layer. Second-order sensitivity analysis guides mixed precision in HAWQ (Dong et al., 2019). Interactions are accounted for in CLADO and CoopQ across layers (Deng et al., 2025; Zhao et al., 2026). Our calibration cost estimator evaluates every candidate on a small calibration set. Dynamic programming solves the additive allocation objective under a storage budget based on these estimates. The analysis addresses the transfer of single-group gains to joint allocations.

D ADDITIONAL FIGURES

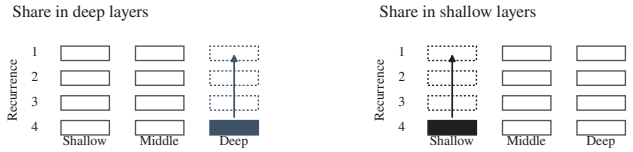


Figure 2: Last-state substitution changes which recurrent KV states are stored. Columns indicate depth and rows recurrences. Dotted states use the reconstructed final state of the same layer instead of being stored separately; other layers retain all recurrences. Figure 1 compares the resulting output distortion.

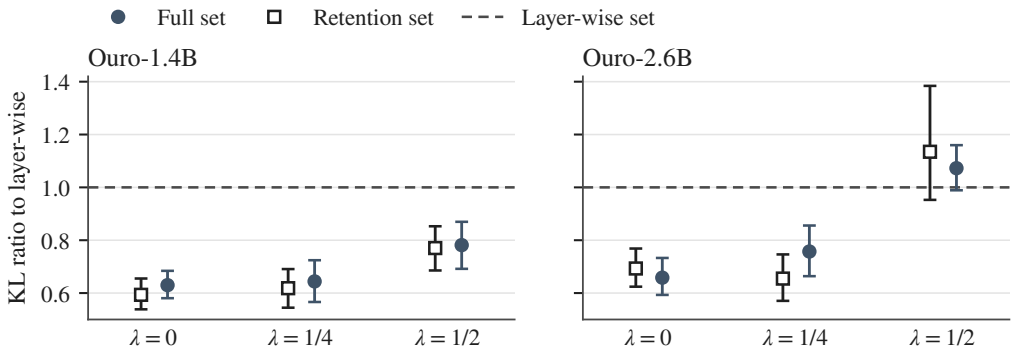


Figure 3: Compression fidelity at matched storage budgets on the shared 256-document test set. Ratios compare each allocation with layer-wise precision allocation (L), which retains all recurrent states. The retention set (S, squares) adds last-only and first-plus-last options; the full set (H, circles) also permits mixed precision across recurrences. Budgets are $U_2 + \lambda(U_3 - U_2)$, where U_b is uniform residual b-bit storage. Bars are unadjusted paired 95% intervals. Corrected primary tests appear in Table 3; other comparisons are descriptive.