FROM HISTORY TO STATE: REVISABLE MEMORY FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers reuse one layer stack across several passes, and each pass hands only its final hidden state to the next pass. The outputs of the blocks inside a pass are not kept, so a later pass cannot revise what an earlier block computed. Keeping these outputs in an additive memory does not allow revision either: the value retrieved for a key is a decayed sum of every value written under that key, so a block that contradicts an earlier block can only add to the earlier value. We propose Revisable Loop Memory, RLM, which gives each token a fixed-size associative memory that persists across passes. At each block, the memory predicts the value for the block's key, and RLM writes only the gated difference between the block's value and this prediction. The size of each write therefore depends on what the memory already holds: a block that agrees with the memory writes little, and a block that disagrees can replace the stored value for its key while leaving associations under orthogonal keys unchanged, which an additive write can do only by erasing the whole memory. The next pass reads the memory at the loop boundary. On Ouro-1.4B under matched fine-tuning, RLM improves GSM8K by 2.3 points and MATH500 by 2.2 points over native fine-tuning with 0.2% more computation per token, and it outperforms additive writing with the same memory size.

1 INTRODUCTION

A looped Transformer applies one shared stack of layers several times to the same tokens (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025), so the number of passes sets the computational depth while the parameter count stays fixed. In the standard design, each pass hands only its final hidden state to the next pass. The blocks inside a pass also produce intermediate outputs, but these outputs reach later passes only through the final hidden state, so a later pass cannot read or correct what a particular block of an earlier pass computed.

Table 1: **Change per additional pass in native Ouro-1.4B** on 10,000 held-out math samples. The NLL reduction is the drop in negative log-likelihood from one pass to the next. The cosine similarity compares the final hidden states of adjacent passes.

Loop Transition	$\mathbf{1 \to 2}$	$\mathbf{2 \to 3}$	$\mathbf{3 \to 4}$
NLL reduction (nats/token)	0.3751	0.0583	0.0068
Adjacent cosine similarity	0.8480	0.9246	0.9615

This interface limits what an extra pass can add. Table 1 measures native Ouro-1.4B on held-out math samples: the reduction in negative log-likelihood from one more pass shrinks by more than fifty times from the second pass to the fourth, while the final hidden states of adjacent passes become almost identical. Ordinary convergence explains part of this pattern, but the measurement raises a design question: what should one pass hand to the next, and how should a later pass revise what earlier passes computed?

Several architectures give later passes more access to earlier computation. CoTFormer lets later applications of the shared layers attend over earlier representations (Mohtashami et al., 2025), Hyperloop Transformers expand the residual state passed between loops (Zeitoun et al., 2026), and

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.

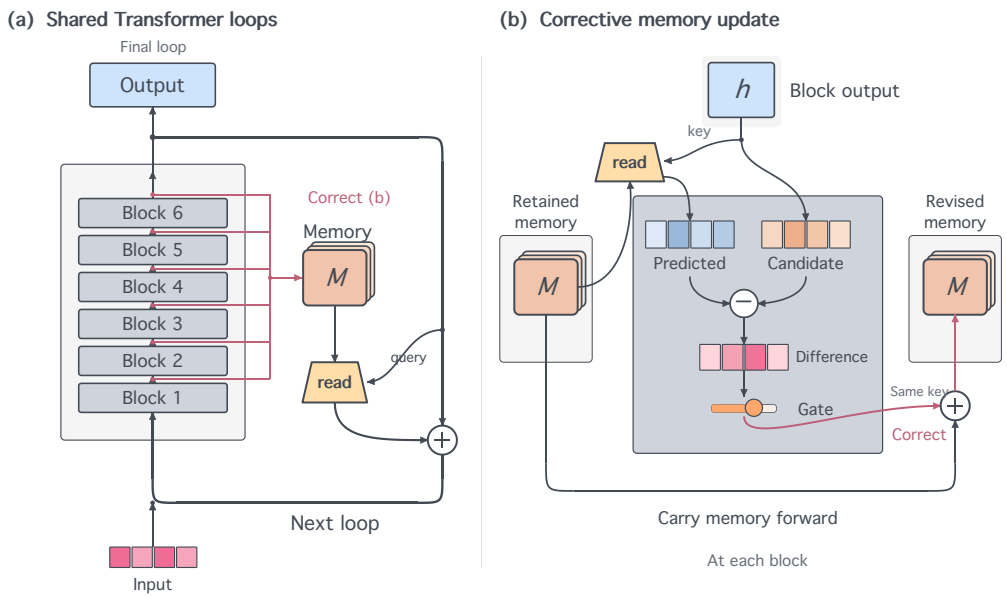

Figure 1: **Overview of Revisable Loop Memory, RLM.** Panel a: a shared Transformer stack runs several loops. Each token keeps a memory that receives a write at every block before the final loop. At each loop boundary, the memory is read and added to the input of the next loop. The final loop does not read the memory and uses the native prediction head. Panel b: at each block, the hidden state gives a key and a proposed value. The memory predicts the value for the same key. A gate that observes the memory sets how much of the difference is written back.

MeSH writes recurrent outputs into a fixed number of memory slots and reads the slots to form the next state (Yu et al., 2026). Within one layer stack, DeepCrossAttention and Attention Residuals combine earlier layer outputs with learned attention weights (Heddes et al., 2025; Kimi Team et al., 2026). These designs decide which earlier outputs are kept and how the kept outputs are read, but they leave open how a new block output should change what is already kept. Block outputs along depth can confirm an association, refine it, or contradict it. An additive write adds the full value every time, so the value retrieved for a key is a decayed sum of all values written under that key, and a block that contradicts an earlier block can only add to the earlier value. Fast-weight models face the same issue along the token sequence and use the delta rule: the memory predicts a value for the key, and the write stores only the difference between the new value and the prediction (Schlag et al., 2021; Yang et al., 2025).

We propose Revisable Loop Memory, RLM, which applies the delta rule to the block outputs of one token across loops. Figure 1 shows the design. Each token keeps one matrix memory instead of a list of past outputs, so the memory has the same size for any number of loops. At each block, the block output gives a key and a proposed value, the memory predicts a value for that key, and RLM writes the gated difference between the two. We expect the useful size of a write to depend on what the memory already holds, so the write gate observes the proposed value, the prediction, and their difference. At each loop boundary, the final hidden state reads the memory, and the retrieved value enters the next pass through a scaled residual connection. Causal attention across tokens and the prediction head are unchanged, and a zero scale recovers the native model.

The paper makes three contributions. First, the delta write makes each write depend on what the memory already holds. One delta write can replace the value stored for a key and leave associations under orthogonal keys unchanged, while an additive write can reach the new value only by erasing the whole memory. Both writes affect other queries in proportion to key overlap, but the delta write passes only the residual, which is small for an association that the memory already returns. Section 2.2 proves these properties. Second, the memory has the same size for every loop count and adds almost no computation to a pass, as Section 4.2 counts. Third, on Ouro-1.4B under matched fine-tuning, RLM improves mathematical reasoning over native fine-tuning, and a comparison of four interfaces

with the same state size shows gains from both the corrective write and the memory-aware gate. Sections 3 and 4 report these results.

2 REVISABLE LOOP MEMORY

2.1 A STATE ALONG THE LOOP-DEPTH TRAJECTORY

RLM keeps the computation inside each loop and changes the information passed between loops. Let a backbone of width d contain L layers, partitioned into B consecutive blocks. For N token positions and T recurrent passes,

$$H_{t,0} = X_t, \qquad H_{t,b} = F_{\theta,b}(H_{t,b-1}), \quad b = 1, \dots, B. \tag{1}$$

Here $H_{t,b} \in \mathbb{R}^{N \times d}$. Block weights are shared across loops and differ across blocks. We write $U_t = G_{\text{nat}}(H_{t,B})$ for the native boundary map of the checkpoint, including its normalization. The native recurrence is $X_{t+1} = U_t$. RLM keeps the native prediction map $D_{\text{nat}}(H_{T,B})$ unchanged.

Each token i also holds J memory heads $M_i^j \in \mathbb{R}^{d_v \times d_k}$. Block outputs update the memory matrices in loop order, and each token position is updated independently. The memory axis indexes internal computation, not the autoregressive token sequence. Tokens interact only through the causal attention of the backbone. Below we drop i, j and index write events by $n = (t, b)$.

2.2 PREDICT AN ASSOCIATION, THEN CORRECT IT

A block state determines both the write address and the proposed value. Let h_n be the block output before boundary normalization, and let $a_b \in \mathbb{R}^{d_e}$ be a learned depth embedding. Each head computes

$$\bar{h}_n = \text{RMS}(h_n), \quad k_n = \text{Unit}(W_k \bar{h}_n + E_k a_b), \quad v_n = W_v \bar{h}_n. \tag{2}$$

Both normalizations are parameter-free: $\text{RMS}(x) = x/\sqrt{\|x\|_2^2 / \dim(x) + \epsilon}$ and $\text{Unit}(x) = x/\max(\|x\|_2, \epsilon)$, with $\epsilon = 10^{-6}$. A separate embedding a_0 initializes the memory from the input of each token as $M_0 = v_0 k_0^\mathsf{T}$. Later writes can revise M_0, so M_0 is not a protected slot.

Retention sets how much of the previous memory survives before the new candidate is written:

$$\alpha_n = \text{sigmoid}(w_\alpha^\mathsf{T}[v_n; a_b] + c_\alpha), \qquad M_n^- = \alpha_n M_{n-1}. \tag{3}$$

The retained memory predicts the candidate value as $\widehat{v}_n = M_n^- k_n$. The discrepancy $e_n = v_n - \widehat{v}_n$ is the correction that the memory needs to return v_n at key k_n. A two-layer write gate with hidden width r_g reads the candidate value, the predicted value, and the discrepancy:

$$\beta_n = \text{sigmoid}\big(w_2^\mathsf{T} \,\text{SiLU}(W_1[v_n; \widehat{v}_n; e_n; a_b] + c_1) + c_2\big). \tag{4}$$

The new state is a gated delta update:

$$\boxed{M_n = M_n^- + \beta_n\big(v_n - M_n^- k_n\big)k_n^\mathsf{T}.} \tag{5}$$

The delta rule and retention come from fast-weight and gated-delta models (Schlag et al., 2021; Yang et al., 2025). RLM applies them to the internal block states of one token. The added hypothesis is that the write gate benefits from reading the existing association. The discrepancy is not a confidence estimate. The final task loss decides which corrections are useful.

The update changes the addressed association in a fixed way. When $\|k_n\|_2 = 1$,

$$M_n k_n = (1 - \beta_n)\widehat{v}_n + \beta_n v_n, \qquad v_n - M_n k_n = (1 - \beta_n) e_n. \tag{6}$$

Proposition 1 states two properties of one write event.

Proposition 1 (Local correction)**.** *Consider one write event with $M^- = \alpha M$, $\|k\|_2 = 1$, and a gate value $\beta \in [0, 1]$. The update $M^+ = M^- + \beta(v - M^- k)k^\mathsf{T}$ satisfies*

$$v - M^+ k = (1 - \beta)(v - M^- k). \tag{7}$$

For any query q, the write changes the retrieved value by

$$(M^+ - M^-)q = \beta(v - M^- k)(k^\mathsf{T} q). \tag{8}$$

Proof. Multiplying the update by k on the right and using $k^\mathsf{T}k = 1$ gives $M^+k = M^-k + \beta(v - M^-k)$. Subtracting both sides from v gives the first identity. Multiplying the update by q on the right gives the second identity. □

The proof uses only the value of β. The state-aware gate in Equation (4) reads M^-, and the first identity still holds for this gate. By Equation (8), the change seen by another query is proportional to the key overlap $k^\mathsf{T}q$. A query orthogonal to k receives no change from the write. The additive write in Equation (13) has the same proportionality, with the full value v in place of the residual $v - M^-k$. With $\beta = 1$, the delta write sets $M^+k = v$ and leaves every query orthogonal to k at αMq. When the stored value Mk is not parallel to v, the additive write reaches $M^+k = v$ only with $\alpha = 0$, which erases every other association. Retention scales every association equally, so retention alone cannot replace one association. Retention still scales the value retrieved by that query by α. The unit normalization in Equation (2) gives $\|k\|_2 < 1$ only when the vector before normalization has norm below ϵ. The residual factor is then $1 - \beta\,\|k\|_2^2$ instead of $1 - \beta$. The state-aware gate makes β a function of M, so Proposition 1 does not imply stability of the whole recurrence.

Corollary 1 (Repeated confirmation without decay)**.** *Let $\alpha = 1$, and let the same unit-key association (k, v) be written at events $n + 1, \ldots, n + m$. Then*

$$v - M_{n+m}k = \left[\prod_{s=1}^{m}(1 - \beta_{n+s})\right](v - M_nk). \tag{9}$$

If $M_nk = v$, the later writes of (k, v) leave the memory unchanged. With gates in $[0, 1]$, the residual norm of a repeated association does not increase. A nonzero residual converges to zero if and only if the product of the factors $1 - \beta_{n+s}$ converges to zero.

The update in Equation (5) is one gradient step on $\ell(M) = \frac{1}{2}\,\|Mk_n - v_n\|_2^2$ from M_n^-, with step size β_n. The task loss in Equation (12) does not include ℓ. An auxiliary loss on ℓ could reward values that the memory already predicts, with no gain on the task.

2.3 Read revised state into the next pass

The current working state decides what the next pass retrieves. At boundary $t < T$, each token and head computes

$$q_{t,i}^j = \mathrm{Unit}(W_q^j\,\mathrm{RMS}(u_{t,i})), \qquad r_{t,i}^j = M_{t,B,i}^j q_{t,i}^j, \tag{10}$$

$$x_{t+1,i} = u_{t,i} + \gamma W_o\,\mathrm{RMS}\Big(\mathrm{Concat}_{j=1}^{J}\, r_{t,i}^j\Big). \tag{11}$$

The shared scalar γ starts at 10^{-3}, and W_o starts nonzero so that the memory branch receives gradients. The small initial scale lets fine-tuning start from a model that behaves like the native one. The RMS normalization in Equation (11) makes the size of the injected vector independent of the norm of the memory, so γ and W_o alone set that size. No memory read enters the language-model head after the final pass, and writes in the final pass are skipped. The forward pass at $T = 1$ is therefore identical to the native forward pass. Setting $\gamma = 0$ recovers the native computation at every loop budget. Both identities hold for fixed backbone weights and serve as implementation checks. Joint fine-tuning changes those weights.

Algorithm 1 gives the forward computation at loop budget T.

2.4 Learning and resource cost

Writes that a later pass reads are trained through final-loop response-token cross-entropy:

$$\mathcal{L} = -\frac{\sum_a \sum_{i \in \mathcal{P}_a} \log p_{\theta,\phi}^{(T)}(y_{a,i} \mid y_{a,<i})}{\sum_a |\mathcal{P}_a|}. \tag{12}$$

Here $\mathcal{P}_a$ holds the response prediction positions of example a, and ϕ denotes the interface parameters. The primary setting uses $T = 4$, adapts the backbone and the interface jointly, and adds no auxiliary discrepancy loss or intermediate-loop loss. The memory resets between examples. Memory arithmetic

Algorithm 1 RLM forward pass at loop budget T

Require: Token sequence y, native input map, shared blocks $F_{\theta,1}, \ldots, F_{\theta,B}$

$X \leftarrow \mathrm{Input}_{\mathrm{nat}}(y)$
$M \leftarrow v_0 k_0^{\mathsf{T}}$ from X and a_0 ▷ per token and head
for $t = 1, \ldots, T$ **do**
  $H \leftarrow X$
  for $b = 1, \ldots, B$ **do**
    $H \leftarrow F_{\theta,b}(H)$ with the native causal attention mask
    if $t < T$ **then**
      Compute k, v, α from H and a_b ▷ Eqs. 2, 3
      $M^- \leftarrow \alpha M, \quad \widehat{v} \leftarrow M^- k, \quad e \leftarrow v - \widehat{v}$
      $\beta \leftarrow \mathrm{WriteGate}(v, \widehat{v}, e, a_b)$ ▷ Eq. 4
      $M \leftarrow M^- + \beta e k^{\mathsf{T}}$ ▷ Eq. 5
    end if
  end for
  if $t < T$ **then**
    $U \leftarrow G_{\mathrm{nat}}(H)$
    $X \leftarrow U + \gamma\, \mathrm{Read}(M, U)$ ▷ Eqs. 10, 11
  end if
end for
return $D_{\mathrm{nat}}(H)$

runs in FP32, and backbone operations and feature projections run in BF16. Full-prefix causal recomputation is the decoding reference, and incremental decoding must preserve the loop and layer cache identities of the checkpoint.

The size of the memory state does not depend on the number of loops. With $S = Jd_k d_v$, the persistent matrices hold NS elements, and $W = 1 + B(T-1)$ write events reach a later read. Each write event costs $O(dJ(d_k + d_v) + S + Jr_g(3d_v + d_e))$ extra operations. The Q, K, V, O projections contain $2dJ(d_k + d_v)$ parameters. For $d = 2048$ and $(J, d_k, d_v) = (4, 32, 32)$, the interface holds 4,096 matrix elements per token and has approximately 1.07M parameters, including the gates. At $N = 2048$, the FP32 memory occupies 32 MiB. Section 4.2 counts full-model FLOPs and storage.

3 Experiments

3.1 Backbone and adaptation

We start from `ByteDance/Ouro-1.4B`, a base model with 24 physical layers, hidden width 2048, and four recurrent passes (ByteDance, 2026). Six groups of four layers provide the block states. All interfaces preserve the tokenizer, attention operators, and final output path. Adaptive exit is disabled. We use $J = 4$ heads, $d_k = d_v = 32$, $d_e = 16$, and $r_g = 32$.

Native–FT fine-tunes the backbone with its native interface. RLM fine-tunes the backbone and the memory interface jointly. Both use the same GSM8K and MATH training data, processed-token budget, optimizer settings, response-only final-loop supervision, and checkpoint selection by development NLL. Ouro denotes the unadapted checkpoint. Table 2 evaluates the three models under one protocol, next to published Qwen scores (Zhu et al., 2025). Table 3 compares write rules and gate inputs under the same adaptation.

3.2 Evaluation protocol

Table 2 evaluates Ouro, Native–FT, and RLM under one few-shot protocol that follows the Ouro report (Zhu et al., 2025). The three models share prompts, scoring code, answer extraction, and the MATH500 evaluator, and each adapted model uses its checkpoint from training seed 17. Table 3 uses zero-shot greedy decoding on GSM8K and MATH500 and averages three training seeds, so its scores differ from those in Table 2. Mathematical generation always uses greedy decoding with a limit of 2,048 tokens, and a failed extraction or a truncated answer counts as incorrect. The Reproducibility Statement gives the shot counts, the code evaluation, and the bootstrap procedure.

Table 2: **General, mathematical, and coding performance in percent.** Qwen scores are published references from Zhu et al. (2025) and give context. The Ouro-family columns use the protocol of Section 3.2, with training seed 17 for Native–FT and RLM. RLM adds about 1.07M interface parameters. Interface effects are measured against Native–FT.

	Published references				Common evaluation protocol		
Model	Qwen2.5	Qwen3	Qwen2.5	Qwen3	Ouro	Native–FT	**RLM**
Scale	1.5B	1.7B	3B	4B	1.4B R4	1.4B R4	1.4B R4
Architecture	Dense	Dense	Dense	Dense	LoopLM	LoopLM	LoopLM
# Params	1.5B	1.7B	3.0B	4.0B	1.4B	1.4B	1.4B +1.07M
General Tasks (% ↑)							
MMLU	60.99	62.46	65.62	73.19	67.28	68.93	69.48
MMLU-Pro	29.11	37.27	37.87	51.40	48.55	49.90	50.88
BBH	43.66	53.51	55.37	70.95	70.88	72.55	73.24
ARC-C	54.44	55.72	55.46	63.65	60.84	62.12	62.80
HellaSwag	67.73	67.09	74.54	75.66	74.28	75.32	75.91
Winogrande	66.77	66.30	70.17	71.19	72.22	73.64	74.19
Math & Coding Tasks (% ↑)							
GSM8K	60.73	70.28	74.60	72.86	78.62	80.89	82.64
MATH500	17.60	25.80	42.60	59.60	81.80	83.40	85.60
HumanEval	52.40	66.50	68.90	77.40	74.39	75.61	76.83
HumanEval+	46.30	59.80	62.20	70.70	67.07	68.90	69.51
MBPP	60.30	68.00	63.00	78.80	73.02	73.81	75.13
MBPP+	50.00	58.50	54.20	65.90	62.70	61.38	60.85

Table 3: **Write-rule and gate-input ablations under the matched zero-shot greedy protocol.** All interfaces share the Ouro-1.4B backbone, adaptation, and four-pass budget. Input and State indicate whether the write gate observes the retrieved association. Each token holds $S = Jd_kd_v = 4{,}096$ matrix elements. Entries are mean ± sample standard deviation over training seeds 17, 29, and 43. Paired bootstrap 95% intervals for RLM minus Add–State and RLM minus Delta–Input are $[0.53, 3.01]$ and $[0.13, 2.63]$ on GSM8K, and $[-0.20, 3.47]$ and $[-0.73, 3.20]$ on MATH500.

Interface	Forward state/token	GSM8K ↑	MATH500 ↑
Native–FT	No added memory	79.53 ±0.30	83.00 ±0.53
Add–Input	S	79.68 ±0.35	83.20 ±0.60
Add–State	S	80.06 ±0.20	83.60 ±0.40
Delta–Input	S	80.44 ±0.27	84.00 ±0.53
RLM (Delta–State)	S	81.80 ±0.23	85.20 ±0.40
Normalized–Add	$S + J$	80.21 ±0.27	83.80 ±0.53

3.3 General and reasoning capabilities

Table 2 separates the effect of adaptation from the effect of the interface. Native fine-tuning on mathematics data already raises most scores of Ouro, including the general benchmarks, so the comparison that isolates the interface is RLM against Native–FT. RLM is higher on eleven of the twelve benchmarks. The largest gains are on the two mathematics benchmarks, which match the adaptation data: on MATH500, RLM reaches 85.6 and Native–FT reaches 83.4. RLM also stays above Native–FT on all six general benchmarks, by half a point to one point, although the adaptation data contain only mathematics. The only loss is MBPP+, where RLM passes two fewer tasks.

3.4 Correction and memory-aware gating

Table 3 crosses the write rule with the gate input. Add–Input and Add–State write the full proposed value. Delta–Input and RLM write the discrepancy from the memory's prediction. The State gates also observe the retrieved association, and the Input gates do not. All variants share projections, retention, state dimensions, and boundary reads. Normalized–Add controls for accumulated magnitude. The

Table 4: **Performance across recurrent depths.** Accuracy in percent at $T = 1$ to 8. $T > 4$ is extrapolation beyond the training depth. Ouro entries are published scores of the unadapted model (Zhu et al., 2025) and differ slightly from the common-protocol scores in Table 2. RLM uses training seed 17 and the few-shot settings of the Ouro report. C-QA is CommonsenseQA.

Benchmark	$T = 1$		$T = 2$		$T = 3$		$T = 4$	
	Ouro	RLM	Ouro	RLM	Ouro	RLM	Ouro	RLM
ARC-C	37.63	37.80	54.86	55.72	59.47	60.92	60.92	62.80
ARC-E	63.85	64.06	80.30	81.02	83.33	84.43	83.96	85.40
C-QA	44.64	44.88	67.98	68.96	74.37	75.68	75.43	77.23
HellaSwag	55.24	55.18	71.15	71.89	74.07	75.20	74.29	75.91
MMLU	41.21	41.48	60.43	61.52	66.71	68.16	67.45	69.48
Winogrande	56.99	56.91	66.69	67.56	71.35	72.69	72.30	74.19
Benchmark	$T = 5$		$T = 6$		$T = 7$		$T = 8$	
	Ouro	RLM	Ouro	RLM	Ouro	RLM	Ouro	RLM
ARC-C	58.96	63.23	59.73	63.05	58.96	62.88	58.19	62.63
ARC-E	82.91	85.69	82.58	85.77	81.99	85.48	82.07	85.27
C-QA	75.35	77.64	74.94	77.56	74.28	77.31	73.55	77.07
HellaSwag	73.72	76.08	72.77	75.99	72.35	75.76	71.60	75.48
MMLU	66.64	69.81	65.77	69.94	65.28	69.72	64.49	69.36
Winogrande	70.32	74.51	71.03	74.74	70.09	74.35	69.30	74.03

additive write is

$$M_n = \alpha_n M_{n-1} + \beta_n v_n k_n^{\mathsf{T}}. \tag{13}$$

The input-only gate removes $\widehat{v}_n$ and e_n from Equation (4):

$$\beta_n = \mathrm{sigmoid}\Big(\widetilde{w}_2^{\mathsf{T}}\,\mathrm{SiLU}(\widetilde{W}_1[v_n; a_b] + \widetilde{c}_1) + \widetilde{c}_2\Big). \tag{14}$$

The input-only gate uses hidden width 73 instead of 32, which matches the parameter count of the state-aware gate to within two parameters per head. All four factorial variants share the retention gate. Normalized–Add keeps an additive numerator A_n and a scalar mass c_n per head:

$$A_n = \alpha_n A_{n-1} + \beta_n v_n k_n^{\mathsf{T}}, \qquad c_n = \alpha_n c_{n-1} + \beta_n, \qquad M_n = \frac{A_n}{\max(1, c_n)}, \tag{15}$$

with $A_0 = v_0 k_0^{\mathsf{T}}$ and $c_0 = 1$. In Normalized–Add, a state-aware gate reads the normalized pre-write memory $\alpha_n A_{n-1}/\max(1, \alpha_n c_{n-1})$, and the boundary read uses M_n. The J mass scalars are the extra J entries in the state budget $S + J$. Dividing only the final readout by a scalar is not an independent control: the RMS normalization in Equation (11) largely removes such a scalar.

Table 3 shows that both design choices raise accuracy and that the two together give the best interface. With a state-aware gate, a corrective write beats an additive write on both benchmarks. With a corrective write, a gate that sees the memory beats a gate that sees only the block output. Normalized–Add stays close to the additive variants, so the gain of RLM does not come from bounding the magnitude of the memory. The gain is consistent with the property that only the corrective write has: the corrective write can replace the value stored for one key, and the write leaves the other associations unchanged. RLM improves on Native–FT by 2.3 points on GSM8K and by 2.2 points on MATH500. On GSM8K, the paired bootstrap intervals of both contrasts exclude zero. On MATH500, the differences point the same way, and the intervals include zero.

The pattern is consistent with the role of the prediction in Equation (5). The corrective write needs the memory's prediction to form the correction, and the state-aware gate uses the same prediction to decide how much of the correction to write. An additive write never forms the correction, and the memory-aware gate adds less to it.

Table 5: **Counted cost per token at $N = 2{,}048$.** FLOPs count the dominant operations of a full-prefix forward pass, using Equation (16). Storage counts weights, per-loop KV caches, and interface state in BF16 and FP32.

Interface	T	GFLOPs per token	Counted storage GiB
Native–FT	4	10.8720	4.172
Native–FT	5	13.5397	4.547
RLM	4	10.8961	4.206

4 Analysis

4.1 Recurrent depth and extrapolation

Table 4 places RLM beyond the four passes used in training, next to the published depth profile of Ouro. The six-benchmark average of RLM stays almost unchanged from four to eight passes, and RLM loses less than half a point on every benchmark over this range. Every RLM score peaks at five or six passes, beyond the training depth. The published Ouro scores peak at four passes and fall by about 2.5 points on average from four to eight passes.

4.2 Compute and storage accounting

The backbone has $L = 24$, $d = 2{,}048$, feed-forward width $h = 5{,}632$, vocabulary size $V = 49{,}152$, and 16 KV heads of width 128. With a multiply-add counted as two FLOPs and only causal attention pairs counted, the native forward cost per token is

$$F_{\mathrm{nat}}(T, N) \simeq TL\left(8d^2 + 6dh + 2(N+1)d\right) + 2dV. \tag{16}$$

The count excludes normalization, activations, softmax, and other elementwise operations. The memory adds about 0.2% to the arithmetic of a four-pass forward pass, while one more native pass adds about a quarter. The memory state itself is small next to the KV caches. The KV caches grow with the number of passes in both models, so the memory does not change how total decoding memory scales with depth. An interface that kept every block output would add a state that also grows with depth.

5 Related Work

Parameter sharing and recurrent depth. Reusing parameters across depth adds sequential computation without a proportional increase in parameter count. Universal Transformers and ALBERT study shared Transformer parameters (Dehghani et al., 2019; Lan et al., 2020). Recurrent stacking and later studies of sharing strategies examine the trade-off between parameter count and translation quality (Dabre & Fujita, 2019; Takase & Kiyono, 2023). Deep equilibrium models define weight-tied computation through an implicit fixed point (Bai et al., 2019). Work on explicit looping studies programmable computation, in-context learning, iterative optimization, and length generalization (Giannou et al., 2023; Yang et al., 2024; Gatmiry et al., 2024; Fan et al., 2025). In language modeling, latent-thought studies, Huginn, and Ouro study reasoning and inference-time scaling through recurrent depth (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025). Relaxed Recursive Transformers convert pretrained models into shared recursive blocks with depth-specific low-rank adaptations (Bae et al., 2025a). RLM keeps this looped setting and changes the state passed between recurrent applications: intermediate block outputs revise a token-local memory.

Latent reasoning and adaptive computation. Continuous-space reasoning methods differ in what they feed back. Coconut feeds the final hidden state back as the next input embedding (Hao et al., 2024). PonderLM feeds back an embedding averaged under the predicted vocabulary distribution (Zeng et al., 2025). PHD-Transformer adds hidden decoding steps with selective KV-cache retention (Wu et al., 2025). Other methods adapt the amount of computation across tokens or iterations. Ouro learns recurrent exit probabilities (Zhu et al., 2025). Inner Thinking Transformer and Mixture-of-Recursions allocate recurrent depth through token-dependent routing (Chen et al., 2025; Bae et al., 2025b). LoopFormer conditions computation on time and step size to support variable budgets (Jeddi

et al., 2026), and MoDr routes among low-rank recurrent branches (Zhang et al., 2026). At a given loop budget, RLM changes how the intermediate computations are consolidated.

Recurrent dynamics and state propagation. Studies of recurrent networks document overthinking on algorithmic tasks (Bansal et al., 2022). Mechanistic analyses of looped language models examine cyclic convergence and the stabilization of attention across recurrences (Blayney et al., 2026). These observations motivate the study of state propagation, but the observations do not show that additional recurrence fails in all settings. Several architectures already give later recurrences richer access to earlier recurrent information. CoTFormer interleaves earlier and newly computed representations, so later applications of the shared layers attend over the augmented sequence (Mohtashami et al., 2025). Fully Looped Transformers distribute recurrent signals across layers and inject them through attention (Fu et al., 2026). Hyperloop Transformers place hyper-connections between recurrent applications and expand the recurrent residual state (Zeitoun et al., 2026). MeSH, the closest design, keeps a fixed number of memory slots, writes recurrent-core outputs into the slots through learned routers, and reads the updated memory to form the next state (Yu et al., 2026). RLM also keeps a fixed-size state but uses a different write rule: each block addresses a token-local matrix with a key, predicts its proposed value from the matrix, and writes only the gated discrepancy. The matrix is revised in place and read at loop boundaries.

Attention-based residual and cross-depth aggregation. DeepCrossAttention uses input-dependent combinations of preceding layer outputs and depth-wise cross-attention (Heddes et al., 2025). Attention Residuals replaces uniform residual accumulation with source-wise softmax attention driven by learned pseudo-queries (Kimi Team et al., 2026). Block AttnRes sums transformation outputs within blocks to reduce history storage and attends over the resulting block representations. XAttnRes extends history aggregation across encoder and decoder stages for medical image segmentation (Liu et al., 2026). These methods aggregate representations that remain separately accessible. RLM instead keeps one associative state per token, predicts each candidate value from that state, and writes the gated discrepancy. The attention, feed-forward, normalization, and residual operators inside each loop are unchanged.

Fast weights and the delta rule. Linear attention writes the outer product of each key and value into a matrix state that later queries read (Katharopoulos et al., 2020). Schlag et al. (2021) interpret this state as a fast-weight memory and replace the additive write with the delta rule, which writes only the difference between the new value and the value that the memory returns for the key. Gated DeltaNet combines the delta rule with a decay gate (Yang et al., 2025). These models write along the token sequence and use the memory in place of softmax attention. RLM applies the same write along the depth of one token, keeps causal attention unchanged, and uses the memory only to pass information between loops. RLM also gives the write gate the memory's prediction, which these models do not use.

6 Conclusion

A looped Transformer passes only its final hidden state from one pass to the next, and an additive memory can keep earlier block outputs but cannot replace one stored value alone. RLM gives each token a fixed-size memory that later blocks correct instead of extend. Each block writes the gated difference between its proposed value and the memory's prediction, so a block that agrees with the memory writes little, and a block that disagrees can replace the value stored for its key. The next pass reads the memory at the loop boundary, and causal attention and the prediction head stay unchanged.

Under matched fine-tuning on Ouro-1.4B, the corrective write improves GSM8K by 2.3 points over native fine-tuning, and it outperforms additive writing and an input-only gate with the same memory size at about 0.2% extra computation per token. Bounding the magnitude of an additive memory does not reproduce this gain, which is consistent with revision, not size control, as the property that matters. RLM also keeps its accuracy when it runs more passes than it was trained with.

The results suggest a design rule for the interface between loops. The question is not only which earlier computation to keep, but also how a new block output should change what is already kept. Writing the correction answers the second question at almost no cost. The write rule does not depend on Ouro and can be added to any looped model that passes a state between loops.

AI USE STATEMENT

The authors used a large language model to edit the wording and layout of this manuscript and to recheck the arithmetic in its tables.

REPRODUCIBILITY STATEMENT

Section 2 defines every interface component. Algorithm 1 gives the forward computation. Section 3.4 defines the baselines. Table 6 lists the adaptation settings. Adaptation uses the GSM8K and MATH training splits (Cobbe et al., 2021; Hendrycks et al., 2021). Table 7 lists the benchmark settings for Table 2. A failed answer extraction or a truncated completion counts as incorrect. Code generation uses EvalPlus with base-model prompts and one greedy completion per task. HumanEval and HumanEval+ share the 164 tasks of HumanEval+ v0.1.10, MBPP and MBPP+ share the 378 tasks of MBPP+ v0.2.0, and base and augmented tests score the same completion. The intervals in Table 3 come from 10,000 paired bootstrap replicates. Each replicate resamples three training seeds and all problems of a benchmark with replacement, applies the same draws to every interface, and averages each difference over the sampled seed and problem pairs. Intervals are the 2.5th and 97.5th percentiles without correction for multiple comparisons. The native diagnostic in Table 1 uses 10,000 held-out math samples from UltraData-SFT-2605. NLL is averaged over response tokens. Adjacent-loop cosine similarity compares the final-block states of consecutive loops after the boundary normalization. The similarity is first averaged over the response positions of each sample, then averaged over samples with equal weight.

Table 6: **Adaptation settings.**

Setting	Value
Backbone	Ouro-1.4B Base, original tokenizer and output head
Loops and blocks	$T = 4$, block endpoints at layers 4, 8, 12, 16, 20, 24
Memory dimensions	$J = 4$, $d_k = d_v = 32$, $d_e = 16$, $r_g = 32$
Precision	BF16 backbone, FP32 memory and gate arithmetic
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$
Peak learning rate	Backbone 2×10^{-5}, interface 2×10^{-4}
Schedule	3% token warmup, cosine decay to 10% of peak
Weight decay	0.1 on matrix weights, none on biases, norms, depth embeddings, and γ
Gradient clipping	Global norm 1.0
Context and batch	2,048 tokens per sequence, about 32,768 non-padding tokens per step
Token budget	32M processed non-padding tokens
Seeds	17, 29, 43
Checkpoint selection	Minimum development response NLL at common token intervals
Initialization	Xavier projections, depth embedding std 0.02, $\gamma_0 = 10^{-3}$, $\alpha_0 = 0.99$, $\beta_0 = 0.1$

Table 7: **Evaluation settings for Table 2**, following Zhu et al. (2025).

Benchmark	Scoring and prompting	Framework
MMLU	Log probabilities, 5-shot	lm-eval-harness
MMLU-Pro	Strict match, 5-shot CoT	lm-eval-harness
BBH	Strict match, 3-shot CoT	lm-eval-harness
ARC-C	Log probabilities, 25-shot	lm-eval-harness
HellaSwag	Log probabilities, 10-shot	lm-eval-harness
Winogrande	Log probabilities, 5-shot	lm-eval-harness
GSM8K	Strict match, 3-shot CoT	lm-eval-harness
MATH500	Strict match, 5-shot CoT	Ouro in-house evaluator
HumanEval, HumanEval+	pass@1	EvalPlus
MBPP, MBPP+	pass@1	EvalPlus

REFERENCES

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*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/54d6a55225cebbdc16fbb0e45c5bdf2b-Abstract-Conference.html.

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, 2025b. URL https://neurips.cc/virtual/2025/poster/118085.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/7f70331dbe58ad59d83941dfa7d975aa-Abstract-Conference.html.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, Michael M. Bronstein, and Xiaowen Dong. A mechanistic analysis of looped reasoning language models. *arXiv preprint arXiv:2604.11791*, 2026. URL https://arxiv.org/abs/2604.11791.

ByteDance. Ouro-1.4B: Official model card, 2026. URL https://huggingface.co/ByteDance/Ouro-1.4B.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1369. URL https://aclanthology.org/2025.acl-long.1369/.

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.

Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6292–6299, 2019. doi: 10.1609/aaai.v33i01.33016292. URL https://ojs.aaai.org/index.php/AAAI/article/view/4590.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. In *International Conference on Learning Representations*, 2019. URL https://iclr.cc/virtual/2019/poster/1068.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/25cc3adf8c85f7c70989cb8a97a691a7-Abstract-Conference.html.

Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. Simply stabilizing the loop via fully looped transformer. *arXiv preprint arXiv:2605.18797*, 2026. URL https://arxiv.org/abs/2605.18797.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15130–15152. PMLR, 2024. URL https://proceedings.mlr.press/v235/gatmiry24b.html.

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, 2025. URL https://arxiv.org/abs/2502.05171.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2024. URL https://arxiv.org/abs/2412.06769.

Mike Heddes, Adel Javanmard, Kyriakos Axiotis, Gang Fu, Mohammadhossein Bateni, and Vahab Mirrokni. DeepCrossAttention: Supercharging transformer residual connections. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 22881–22903. PMLR, 2025. URL https://proceedings.mlr.press/v267/heddes25a.html.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026. URL https://arxiv.org/abs/2603.15031.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Xinyu Liu, Qing Xu, and Zhen Chen. XAttnRes: Cross-stage attention residuals for medical image segmentation. *arXiv preprint arXiv:2604.03297*, 2026. URL https://arxiv.org/abs/2604.03297.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. CoTFormer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1eaa5146756be028ad6fff1efcc8e6bd-Paper-Conference.pdf.

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. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2676109d49d1eb26d6bc584a8f556305-Abstract-Conference.html.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In *Proceedings of the Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pp. 78–90. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.sustainlp-1.5. URL https://aclanthology.org/2023.sustainlp-1.5/.

Bohong Wu, Shen Yan, Sijun Zhang, Jianqiao Lu, Yutao Zeng, Ya Wang, and Xun Zhou. Efficient pretraining length scaling, 2025. URL https://arxiv.org/abs/2504.14992.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025. URL https://arxiv.org/abs/2412.06464.

Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, Jiaang Li, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. MeSH: Memory-as-state-highways for recursive transformers. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ef12378d988ef4dac37b200d1906056a-Paper-Conference.pdf.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026. URL https://arxiv.org/abs/2604.21254.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. PonderLM: Pretraining language models to ponder in continuous space, 2025. URL https://arxiv.org/abs/2505.20674.

Xiaojing Zhang, Haifeng Wu, Gang He, Jiyang Shen, Bochen Lyu, and Zhanxing Zhu. MoDr: Mixture-of-depth-recurrent transformers for test-time reasoning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a81180649011d1690e05c68bfeaa57c-Abstract-Conference.html.

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