REMEM: REUSABLE LATENT COMPUTATION FOR DEPTH EXTRAPOLATION IN LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Language Models (LoopLMs) provide a promising approach to test-time scaling by repeatedly applying parameter-shared Transformer blocks to increase latent computational depth without increasing model size. However, performance often deteriorates when recurrent depth extends beyond the regime encountered during training, limiting the effectiveness of recurrent-depth extrapolation. Several existing approaches seek to improve recurrent scaling through additional training or post-training, but require extra optimization. In this work, we revisit how information is propagated across recurrent iterations and find that previously computed recurrent states retain reusable computation that can support deeper recurrent processing. Based on this insight, we propose ReMem, a training-free recurrent memory mechanism that explicitly preserves and reuses historical computation during inference. ReMem maintains a compact, evolving recurrent history, allowing newly formed recurrent states to remain directly accessible to later iterations, and uses evidence from the frozen LM head to adaptively allocate historical contributions at the feature level. Importantly, this design requires no parameter updates or additional Transformer evaluations at matched recurrent depth. Experiments across language understanding, mathematical reasoning, and code generation benchmarks show that ReMem generally improves performance retention under recurrent-depth extrapolation, with gains becoming more pronounced as recurrent depth increases.

1 INTRODUCTION

Test-time scaling aims to improve the reasoning capabilities of large language models (LLMs) by allocating additional computation during inference (Snell et al., 2025; Wu et al., 2025). Token-space approaches use explicit reasoning traces (Wei et al., 2022), aggregate sampled reasoning paths (Wang et al., 2023), or search over intermediate thoughts (Yao et al., 2023). Looped Language Models (LoopLMs) provide a complementary alternative by repeatedly applying parameter-shared Transformer blocks to perform additional computation in latent space (Dehghani et al., 2019; Giannou et al., 2023). Recent studies further show that recurrent computation can increase effective computational depth for latent reasoning while reusing the same model parameters (Saunshi et al., 2025; Geiping et al., 2025). The number of iterations defines the model's recurrent depth. When inference extends beyond the recurrent-depth regime encountered during training, the model operates under recurrent-depth extrapolation (Jeddi et al., 2026). Making such extrapolation reliable is essential for turning recurrent depth into an effective axis of test-time scaling.

However, reliably improving LoopLM performance under recurrent-depth extrapolation remains challenging. One straightforward option is to further post-train the model, for example through supervised fine-tuning (SFT) (Zhu et al., 2025). As shown in Fig. 1, SFT improves the peak and overall performance, yet the degradation trend beyond the trained recurrent regime persists. Recent approaches improve the flexibility of parameter sharing (Bae et al., 2025) or stabilize recurrent dynamics through additional training (Yang et al., 2026). Training-free alternatives also explore damped repeated block updates in pretrained Transformers (Chen et al., 2026). Nevertheless, reliably extending an already-trained LoopLM beyond its training depth remains challenging, partic-

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.

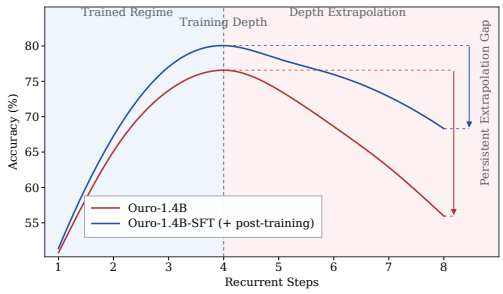

Figure 1: GSM8K performance of Ouro-1.4B before and after SFT across recurrent depths. SFT improves overall performance, while degradation beyond the trained recurrent regime persists.

ularly without further optimization. This motivates a practical question: can additional recurrent computation be utilized more reliably beyond the trained regime, without retraining the model?

To investigate this question, we revisit how information is propagated across recurrent iterations. Although increasing recurrent depth introduces additional latent computation, standard single-state recurrence provides each new iteration with direct access only to the latest hidden representation (Saunshi et al., 2025). As recurrence proceeds, earlier representations are no longer directly accessible as independent recurrent states, despite encoding computation performed at earlier stages of the trajectory. This creates an inherent mismatch during depth extrapolation: computational depth continues to increase, while direct access to previously computed latent information remains restricted to a single evolving state. Consequently, additional recurrence does not necessarily translate into additional usable computation. We therefore hypothesize that previously computed recurrent states may retain reusable latent computation, and that restoring direct access to such computation can support subsequent recurrent processing at greater depths.

Our empirical study in Sec. 2 supports this hypothesis, showing through controlled state re-entry that restoring access to earlier recurrent states can benefit subsequent recurrent processing at extrapolated depths. However, this insight also presents two practical challenges: how should recurrent history evolve as computation progresses, and how should useful historical information be incorporated into the current computation? To address these challenges, we introduce **ReMem**, a training-free mechanism with two complementary components: (1) **Compact Temporal Memory**, which maintains a rolling recurrent history that evolves with ongoing computation, allowing newly formed states to become directly reusable by later iterations; and (2) **State-Contrastive Historical Fusion**, which uses contrastive evidence from the frozen LM head to adaptively allocate historical contributions at the feature level. The fused recurrent state subsequently becomes part of the evolving history, allowing newly completed computation to remain available to later iterations. In contrast to training-based approaches, ReMem directly augments the recurrent-state interface of a pretrained LoopLM, requiring neither parameter updates nor additional Transformer evaluations at matched recurrent depth.

The key contributions of this paper are summarized as follows:

- We show that historical recurrent states remain reusable during recurrent-depth extrapolation, motivating a reusable latent computation perspective on LoopLM inference.
- We propose ReMem, a training-free recurrent memory mechanism that maintains evolving history and adaptively reuses it at the feature level using state-contrastive readout evidence.
- Experiments across language understanding, mathematical reasoning, and code generation show that ReMem generally improves performance retention under recurrent-depth extrapolation without parameter updates or additional Transformer evaluations at matched recurrent depth.

2 Reusable Recurrent History

Preliminaries. Let $\mathbf{h}_0 \in \mathbb{R}^{T\times d}$ denote the initial representation of a sequence of length T with hidden dimension d. We regard $\mathbf{h}_0$ as the recurrent state at depth zero. A Looped Language Model

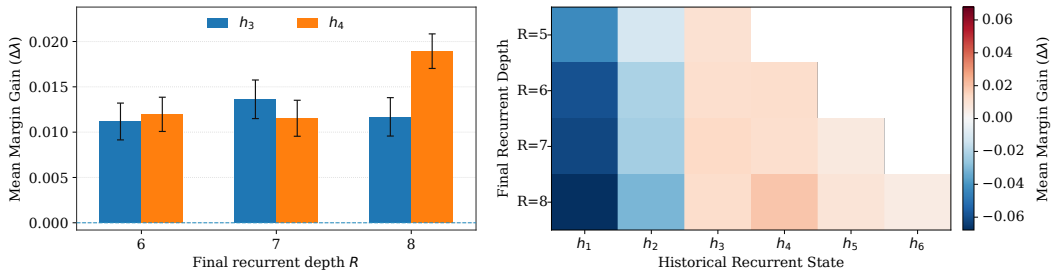

Figure 2: **Recurrent history remains reusable. Left:** Mean correct-answer margin gains from reintroducing $\mathbf{h}_3$ or $\mathbf{h}_4$ at $R = 6, 7, 8$; error bars denote 95% bootstrap confidence intervals. **Right:** Margin gains across historical source positions and target recurrent depths. Positive values indicate improvement over the original model; blank cells denote unavailable source–depth pairs.

repeatedly applies a parameter-shared Transformer stack $\mathcal{F}_\theta$ (Saunshi et al., 2025; Zhu et al., 2025):

$$\mathbf{h}_r = \mathcal{F}_\theta(\mathbf{h}_{r-1}), \qquad r = 1, \ldots, R, \tag{1}$$

where θ is shared across recurrent iterations and R denotes the recurrent depth. Let R_{tr} denote the recurrent depth used during training. Inference with $R > R_{\text{tr}}$ is referred to as **recurrent-depth extrapolation**. Since the same $\mathcal{F}_\theta$ is reused across iterations, increasing R increases effective computational depth without increasing the number of model parameters.

Empirical Study. We examine whether earlier recurrent states remain useful beyond the trained depth using a controlled one-shot historical re-entry diagnostic. We evaluate the frozen Ouro-1.4B model (Zhu et al., 2025) ($R_{\text{tr}} = 4$) on the full MMLU test set (Hendrycks et al., 2021), comprising 14,042 examples across 57 subjects. For each target depth $R \in \{5, 6, 7, 8\}$, we run the original recurrence to $j = R - 1$ and reintroduce an earlier state $\mathbf{h}_k$, where $k < j$:

$$\bar{\mathbf{h}}_j^{(k)} = (1 - \alpha)\mathbf{h}_j + \alpha\mathbf{h}_k. \tag{2}$$

We set $\alpha = 0.10$, match the token-wise RMS of $\bar{\mathbf{h}}_j^{(k)}$ to that of $\mathbf{h}_j$, and then apply the final shared transformation:

$$\widetilde{\mathbf{h}}_R^{(k)} = \mathcal{F}_\theta\left(\mathcal{R}_{\mathbf{h}_j}\left(\bar{\mathbf{h}}_j^{(k)}\right)\right), \tag{3}$$

where $\mathcal{R}_{\mathbf{h}_j}(\cdot)$ denotes token-wise RMS matching. This intervention changes only the state entering the final recurrence while preserving the target depth and number of Transformer evaluations.

We measure its effect using the correct-answer margin

$$\lambda_i^{(R)} = s_i^{(R)}(y_i) - \max_{a \neq y_i} s_i^{(R)}(a), \tag{4}$$

where $s_i^{(R)}(a)$ is the summed continuation log-likelihood of candidate answer a. The re-entry gain is defined as

$$\Delta\lambda_i^{(k,R)} = \widetilde{\lambda}_i^{(k,R)} - \lambda_i^{(R)}, \tag{5}$$

where a positive $\Delta\lambda$ indicates an increased correct-answer margin, while a negative $\Delta\lambda$ indicates a decreased margin.

Finding 1. Historical recurrent states remain reusable during depth extrapolation. Figure 2 (left) shows that reintroducing $\mathbf{h}_3$ or $\mathbf{h}_4$ yields positive mean correct-answer margin gains across $R \in \{6, 7, 8\}$. Despite being produced several iterations earlier, these states can still improve the final margin when directly reintroduced, indicating that useful intermediate computation is not necessarily fully captured by the latest recurrent state. This suggests that task-relevant information can persist in historical states even after several additional recurrent transformations. We therefore view recurrent history as a source of reusable latent computation rather than merely as transient intermediate representations.

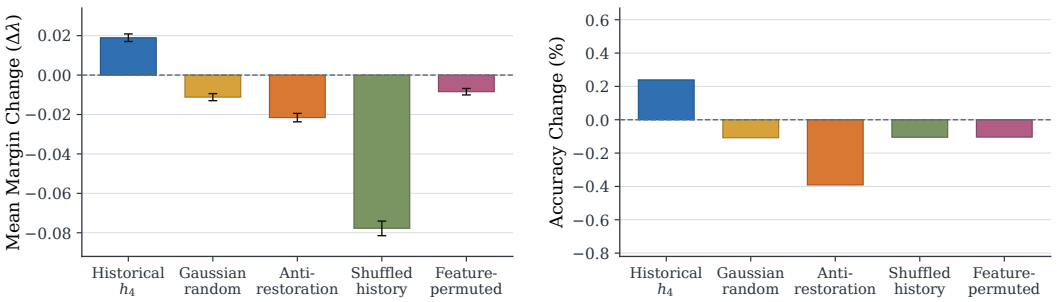

Figure 3: **Historical re-entry versus controls. Left:** Mean margin change; error bars denote 95% bootstrap confidence intervals. **Right:** Mean accuracy change. Intact same-sample $\mathbf{h}_4$ is compared with Gaussian random, anti-restoration, shuffled-history, and feature-permuted controls.

Finding 2. Historical utility evolves with recurrent computation. Figure 2 (right) shows that the effect of historical re-entry depends jointly on the source position and the target recurrent depth. At a fixed depth, different historical states yield different margin gains, while the utility of the same state also changes as recurrence progresses. Very early states, such as $\mathbf{h}_1$ and $\mathbf{h}_2$, are generally less useful, whereas states formed later in the recurrent trajectory tend to provide more favorable gains. These results suggest that the region of useful historical computation tends to shift toward later states as the recurrent trajectory evolves, rather than remaining anchored to a fixed set of earlier states. Such evolving utility motivates maintaining a recurrent history that advances with the computation, keeping newly formed states available for subsequent reuse.

Finding 3. Effective reuse depends on the structure of the historical representation. A possible alternative explanation is that the gains from historical re-entry arise simply from perturbing the current representation. To test this possibility, we compare intact $\mathbf{h}_4$ re-entry with several controlled perturbations at $R = 8$. As shown in Fig. 3, genuine same-sample history is the only tested condition that yields positive average changes in both correct-answer margin and accuracy, whereas Gaussian random, anti-restoration, shuffled-history, and feature-permuted controls produce negative average changes. In particular, the shuffled-history control indicates that sample alignment is important for effective reuse, while the feature-permuted control shows that preserving feature correspondence also matters. These results suggest that the benefit of historical reuse depends on the structured, sample-consistent information carried by recurrent states, rather than on perturbation alone. This structural dependence further motivates selectively integrating historical information rather than treating retained states as interchangeable. Detailed constructions of the control interventions are provided in Appendix C.

To summarize, the above analysis demonstrates that historical recurrent states can retain reusable computation during depth extrapolation, while also revealing two key practical challenges: (1) **how to maintain an evolving recurrent history as computation progresses?** and (2) **how to adaptively integrate historical information into the current computation?**

3 ReMem: Recurrent Memory for Depth Extrapolation

To address the challenges identified in Sec. 2, we propose ReMem, a training-free mechanism that augments the recurrent-state interface with (1) Compact Temporal Memory, which maintains an evolving window of recent recurrent states, and (2) State-Contrastive Historical Fusion, which adaptively integrates historical information at the feature level using evidence from the frozen LM head. ReMem is applied at each recurrent iteration, while our primary evaluation focuses on recurrent-depth extrapolation, where $R > R_{\text{tr}}$. All pretrained parameters remain frozen.

3.1 Compact Temporal Memory

The analysis in Sec. 2 shows that historical utility changes as recurrent computation progresses. Rather than anchoring reuse to fixed earlier states, ReMem maintains a rolling memory so that newly completed computation remains accessible to later iterations.

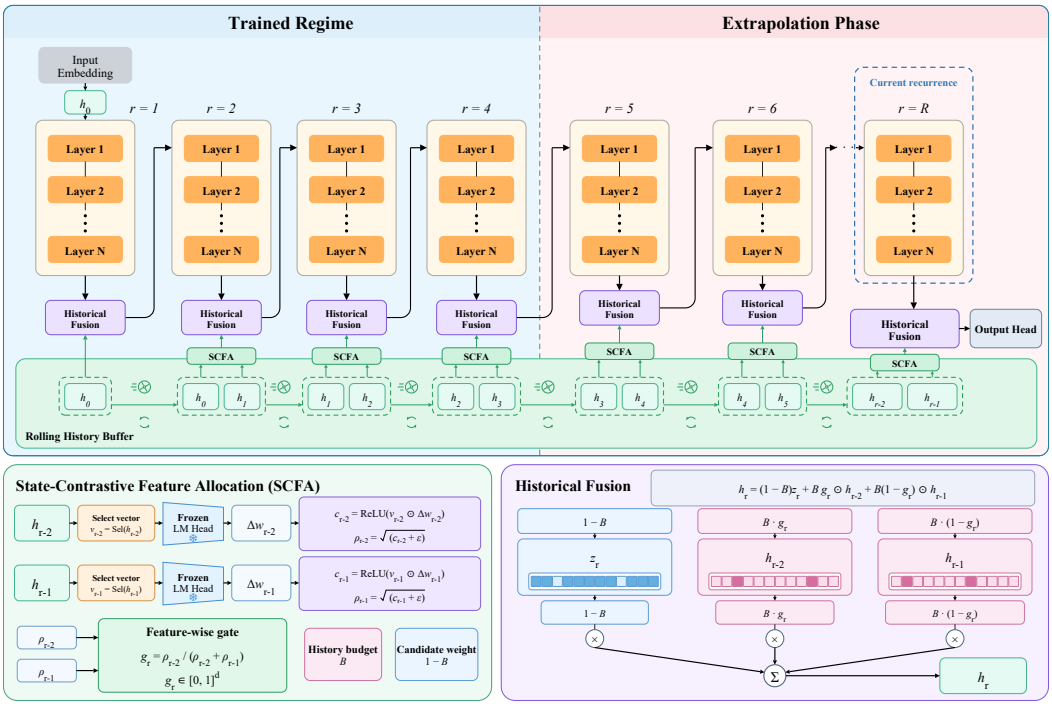

Figure 4: **Overview of ReMem.** A rolling memory preserves recent recurrent states, whose feature-wise contributions are allocated using frozen readout evidence and fused with the newly computed candidate. The illustration shows $K = 2$.

At iteration r, the shared Transformer stack produces $\mathbf{z}_r = \mathcal{F}_\theta(\mathbf{h}_{r-1})$, where $\mathbf{h}_{r-1}, \mathbf{z}_r \in \mathbb{R}^{T\times d}$. Here, $\mathbf{h}_0$ is the initial representation and $\mathbf{h}_r$ for $r \geq 1$ denotes the state after historical fusion. ReMem retains up to $K \in \mathbb{N}_+$ states:

$$\mathcal{M}_r = (\mathbf{h}_{r-m_r}, \ldots, \mathbf{h}_{r-1}), \qquad m_r = \min(K, r). \tag{6}$$

After fusion, $\mathbf{h}_r$ enters the memory and states outside the most recent K positions are discarded. Thus, K controls the span of accessible history while keeping memory bounded as recurrence advances. The contributions of retained states are determined adaptively below.

3.2 State-Contrastive Historical Fusion

We separate the overall strength of historical reuse from its allocation among retained states. A history budget $B \in [0, 1)$ controls the total historical mixing weight, while State-Contrastive Feature Allocation (SCFA) distributes this weight independently across hidden features. Let $\mathcal{S}_r = \{r - m_r, \ldots, r - 1\}$ denote the indices in $\mathcal{M}_r$.

Contrastive readout evidence. For each retained state, we extract a boundary representation $\mathbf{v}_s = \mathrm{Sel}(\mathbf{h}_s) \in \mathbb{R}^d$ as a compact summary at the prediction boundary. For multiple-choice evaluation, the selector is the last token of the shared prompt prefix and the allocation is shared across candidate continuations; for autoregressive generation, it is the current last token.

Let $\mathcal{O}_\theta$ denote the frozen readout and $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times d}$ its output projection. For each state s, let $\tau_s^{(1)}$ and $\tau_s^{(2)}$ denote the top-1 and top-2 token indices predicted from $\mathbf{v}_s$, and define

$$\Delta\mathbf{w}_s = \mathbf{W}_{\tau_s^{(1)}} - \mathbf{W}_{\tau_s^{(2)}}. \tag{7}$$

We construct non-negative feature evidence as

$$\mathbf{c}_s = \mathrm{ReLU}\left(\mathbf{v}_s \odot \Delta\mathbf{w}_s\right), \qquad \boldsymbol{\rho}_s = \sqrt{\mathbf{c}_s + \epsilon\mathbf{1}}, \tag{8}$$

where $\epsilon > 0$ and all operations are element-wise. The top-1/top-2 contrast captures the state's local predictive preference without labels; the element-wise product identifies positively aligned features, while the square root moderates overly concentrated evidence. Each state uses its own predicted token pair, without candidate-answer scores or future recurrent states. We use $\boldsymbol{\rho}_s$ only as a readout-aligned proxy for historical allocation, not as an estimate of answer correctness.

Feature-wise allocation. SCFA normalizes the evidence across retained states for each hidden feature:

$$\mathbf{g}_{r,s} = \frac{\boldsymbol{\rho}_s}{\sum_{u \in \mathcal{S}_r} \boldsymbol{\rho}_u}, \qquad s \in \mathcal{S}_r, \tag{9}$$

where division is element-wise, $\mathbf{g}_{r,s} \in [0,1]^d$, and $\sum_{s \in \mathcal{S}_r} \mathbf{g}_{r,s} = \mathbf{1}$. Different features can therefore favor different historical states, with allocations recomputed as the memory evolves. If all retained states have zero positive score for a feature, the numerical floor yields equal allocation.

Historical fusion. ReMem combines the candidate with allocated historical information as

$$\mathbf{h}_r = (1 - B)\mathbf{z}_r + B \sum_{s \in \mathcal{S}_r} \mathbf{g}_{r,s} \odot \mathbf{h}_s. \tag{10}$$

Each allocation vector is broadcast across positions in the currently processed block, with incremental decoding detailed in Appendix F.3. For every hidden feature, historical coefficients sum to B, leaving coefficient $1 - B$ for the candidate regardless of K. Thus, B controls historical strength independently of memory size, and $B = 0$ recovers the original recurrence.

At $r = 1$, the update reduces to $\mathbf{h}_1 = (1 - B)\mathbf{z}_1 + B\mathbf{h}_0$. For $K = 2$ and $r \geq 2$, $\mathbf{g}_r := \mathbf{g}_{r,r-2} = \boldsymbol{\rho}_{r-2}/(\boldsymbol{\rho}_{r-2} + \boldsymbol{\rho}_{r-1})$, while the recent state receives $\mathbf{1} - \mathbf{g}_r$. At intermediate iterations, $\mathbf{h}_r$ is propagated to the next shared transformation and added to memory. At $r = R$, $\mathbf{h}_R$ is passed directly to the original readout without another Transformer evaluation. Algorithm 1 in Appendix A summarizes the inference procedure.

3.3 Training-Free Integration and Overhead

ReMem keeps both the shared Transformer stack and readout frozen, introducing neither trainable parameters nor optimization steps. At recurrent depth R, it performs exactly R shared Transformer-stack evaluations, matching original inference at the same depth. For a fixed input prefix and selector boundary, $\boldsymbol{\rho}_s$ depends only on the retained state and frozen readout and can therefore be cached while the state remains in memory; this cache is not reused across arbitrary generated prefixes.

For states of size $T \times d$, retained history requires $O(KTd)$ storage, with $O(Kd)$ additional storage for cached evidence. Computing evidence for a newly retained state requires a frozen vocabulary readout with $O(|\mathcal{V}|d)$ computation, after which feature allocation and historical fusion require $O(Kd)$ and $O(KTd)$ operations per iteration, respectively. For fixed K and T, the history-storage bound does not grow with R. These auxiliary operations incur real computation but no additional Transformer evaluations.

4 Experiments

4.1 Experimental Setup

Models and comparisons. We evaluate ReMem on Ouro-1.4B and Ouro-2.6B (Zhu et al., 2025), which perform recurrent latent computation through parameter-shared Transformer blocks. For both backbones, we evaluate recurrent depths $R \in \{5, 6, 7, 8\}$ beyond the training depth $R_{\text{tr}} = 4$. Each ReMem configuration is compared with original inference on the same backbone and at the same recurrent depth, with pretrained parameters and the number of shared Transformer-stack evaluations held fixed. The main experiments use the Full-Trajectory schedule, with ReMem applied from the first recurrent iteration. We additionally include Rec.-Llama-3.2 and Rec.-OLMo-2-0425 (McLeish et al., 2026), which introduce recurrent computation into pretrained language models, as additional recurrent baselines.

Table 1: Benchmark performance (%) across recurrent depths, with ReMem applied to Ouro-1.4B. Best results at each depth are shown in **bold**.

Rec.	Method	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
$R=5$	Rec.-Llama-3.2	55.80	68.44	45.24	61.20	40.44	37.27	39.73	14.63	21.16	17.20	10.00	51.40	20.54	37.37
	Rec.-OLMo-2-0425	58.01	69.48	49.59	63.17	41.81	38.83	38.82	12.20	17.20	15.34	12.50	53.48	19.21	37.90
	Ouro-1.4B	65.04	**78.89**	**72.61**	75.42	58.79	66.53	73.77	59.76	70.37	59.79	35.00	69.55	59.74	65.09
	Ouro-1.4B-ReMem (Ours)	**66.14**	78.62	71.99	**76.68**	**59.04**	**66.60**	**77.03**	**64.63**	**70.90**	**60.85**	**42.50**	**69.85**	**63.18**	**66.82**
$R=6$	Rec.-Llama-3.2	55.72	68.23	45.26	61.24	40.36	37.42	39.95	14.63	22.49	19.05	22.50	51.37	23.72	38.80
	Rec.-OLMo-2-0425	57.93	69.48	49.66	63.22	41.98	39.05	39.80	12.80	17.46	15.08	12.50	53.55	19.53	38.09
	Ouro-1.4B	64.72	**78.89**	71.63	75.76	58.11	65.80	68.61	59.76	66.40	58.20	32.50	69.15	57.09	63.67
	Ouro-1.4B-ReMem (Ours)	**66.38**	**78.89**	**71.94**	**77.15**	**59.04**	**66.54**	**76.27**	**63.41**	**69.31**	**59.52**	**40.00**	**69.99**	**61.70**	**66.22**
$R=7$	Rec.-Llama-3.2	55.72	68.17	45.37	61.20	40.27	37.20	38.51	14.63	23.54	20.11	17.50	51.32	22.86	38.38
	Rec.-OLMo-2-0425	57.85	69.53	49.66	62.88	41.55	38.93	39.42	13.41	16.67	14.81	20.00	53.40	20.86	38.61
	Ouro-1.4B	64.48	**78.73**	71.16	75.04	57.94	65.33	62.85	57.32	65.61	56.88	35.00	68.78	55.53	62.76
	Ouro-1.4B-ReMem (Ours)	**65.59**	78.45	**71.98**	**77.27**	**59.04**	**65.40**	**75.13**	**62.20**	**68.78**	**58.20**	**40.00**	**69.62**	**60.86**	**65.64**
$R=8$	Rec.-Llama-3.2	56.20	68.23	45.35	61.28	40.53	37.35	38.89	14.63	24.34	20.63	15.00	51.49	22.70	38.40
	Rec.-OLMo-2-0425	57.70	69.48	49.70	62.96	41.72	39.02	39.58	12.80	16.93	15.08	12.50	53.43	19.38	37.95
	Ouro-1.4B	64.56	78.18	70.55	75.46	57.34	64.66	55.95	54.88	60.85	52.12	32.50	68.46	51.26	60.64
	Ouro-1.4B-ReMem (Ours)	**65.04**	**78.24**	**71.67**	**77.10**	**58.36**	**65.05**	**73.84**	**60.98**	**67.46**	**57.41**	**37.50**	**69.24**	**59.44**	**64.79**

Benchmarks and metrics. We evaluate on eleven benchmarks covering general language understanding, mathematical reasoning, and code generation. The general tasks comprise WinoGrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), ARC-Easy and ARC-Challenge (Clark et al., 2018), and MMLU (Hendrycks et al., 2021). The math and coding tasks include GSM8K (Cobbe et al., 2021), AMC23 (Mathematical Association of America, 2023), HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and MBPP+ (Liu et al., 2023). MC Avg and Gen Avg denote the unweighted means over the six multiple-choice tasks and five math/coding tasks, respectively; All Avg denotes the unweighted mean over all eleven benchmarks. Additional implementation and evaluation details are provided in Appendix B.

4.2 Overall Performance

Tables 1 and 2 show that ReMem slows aggregate performance degradation on both Ouro backbones. On Ouro-1.4B, the advantage becomes more pronounced as recurrence deepens, particularly on math and coding tasks, where original inference degrades more rapidly than on general language-understanding tasks. This pattern suggests that historical reuse is especially beneficial on tasks that are more sensitive to recurrent-depth extension. On Ouro-2.6B, ReMem is slightly weaker in aggregate at $R = 5$ but becomes advantageous at deeper settings, indicating a depth-dependent benefit rather than a uniform accuracy offset. The higher-depth evaluations further show that improved retention persists on both model scales (Appendix D.1): even the larger backbone loses substantial capability under extended recurrence, while ReMem preserves more of its performance. At $R = 8$, ReMem also achieves aggregate performance comparable to Ouro-1.4B-SFT without additional parameter optimization (Appendix D.4). Together, these comparisons support historical-state reuse as an effective training-free approach to preserving capability during depth extrapolation.

4.3 Ablation Study

We evaluate four variants on Ouro-1.4B at $R = 8$ to examine the roles of historical access and allocation. Table 3 reports the aggregate results.

Compact Temporal Memory. w/o Multi-state retains only the immediately preceding state ($K = 1$), reducing historical reuse to a damped recurrent update. Its lower performance indicates that moderating the candidate update with the latest state does not recover the full benefit of directly accessing multiple recurrent stages. w/o Rolling instead retains the fixed pair $(\mathbf{h}_3, \mathbf{h}_4)$. Although this variant preserves two historical states, it also underperforms ReMem, supporting memory renewal in addition to multi-state access. A fixed history preserves the same earlier computation, whereas rolling updates make newly completed computation available for later reuse. This distinction is consistent with the evolving historical utility observed in Sec. 2.

State-Contrastive Historical Fusion. w/o Adaptive assigns equal weights under the same rolling-memory policy, while w/o Feature-wise uses one adaptive scalar per historical state. Equal allocation

Table 2: Aggregate performance (%) on Ouro-2.6B across recurrent depths.

Rec.	Method	MC Avg	Gen Avg	All Avg
$R = 5$	Original	74.74	**67.55**	**71.47**
	ReMem	**74.85**	65.88	70.77
$R = 6$	Original	74.50	63.45	69.48
	ReMem	**74.92**	**64.36**	**70.12**
$R = 7$	Original	74.29	61.02	68.26
	ReMem	**74.90**	**61.82**	**68.95**
$R = 8$	Original	73.92	58.89	67.08
	ReMem	**74.59**	**60.03**	**67.97**

Table 3: Ablation study on Ouro-1.4B at $R = 8$.

Method	MC Avg	Gen Avg	All Avg
ReMem	**69.24**	**59.44**	**64.79**
w/o Multi-state	69.07	58.55	64.29
w/o Rolling	69.12	58.48	64.29
w/o Adaptive	69.04	57.54	63.81
w/o Feature-wise	69.02	58.49	64.23

causes the largest reduction in All Avg and Gen Avg among the four ablations. Because the total history budget is unchanged, this comparison concerns how historical contributions are distributed rather than their overall mixing strength. Scalar weighting recovers part of the loss in All Avg and Gen Avg, but assigns the same historical preference to every hidden dimension. Feature-wise allocation relaxes this constraint by allowing different dimensions to favor different states. ReMem's higher scores support integrating historical contributions at this finer granularity rather than relying only on state-level weighting.

The higher-depth comparisons reinforce the contribution of adaptive reuse. On Ouro-1.4B at $R = 10, 12, 16$, ReMem outperforms Single-State Damping and Uniform Historical Fusion on all three aggregate metrics under the same fixed $B = 0.20$, with the clearest gains on math and coding tasks (Appendix D.2). The activation study further shows that reuse introduced only after $r = 4$ is effective, while Full-Trajectory activation provides stronger high-depth retention (Appendix D.3). Early fusion changes both the state entering deeper recurrence and the history available to subsequent updates, suggesting that the benefits of reuse also depend on how the preceding trajectory is formed. Allocation dynamics and sensitivity to B and K are analyzed in Appendices D.5 and E, respectively.

4.4 Efficiency Analysis

Figure 5 compares accuracy against measured per-token latency on MMLU and GSM8K across $R = 5$–8. On MMLU, ReMem improves matched-depth accuracy with little change in latency, providing additional performance with limited measured cost for this scoring workload. On GSM8K, the decoding overhead becomes more visible at deeper recurrence, but accuracy declines much less sharply than under original inference. The additional cost is therefore accompanied by better capability retention, particularly where original inference spends more computation while losing performance. Cross-depth comparisons further show that some shallower ReMem configurations outperform deeper original configurations at comparable or lower per-token latency. This indicates that reusing prior recurrent computation can offer a more favorable accuracy–latency trade-off than simply adding iterations to original inference. Across the evaluated workloads, ReMem combines improved performance retention with modest, workload-dependent auxiliary overhead. Detailed latency, memory, and runtime measurements are provided in Appendix F.

5 Related Work

Recurrent and Latent Computation. Looped Transformers increase latent computational depth by repeatedly applying parameter-shared transformations (Dehghani et al., 2019). Studies of computational expressivity (Giannou et al., 2023) and iterative learning (Yang et al., 2024; Gatmiry et al., 2024) provide foundations for recurrent computation. Recent work extends this paradigm to latent reasoning and large-scale language models (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025), as well as compositional generalization (Kohli et al., 2026). Complementary approaches introduce additional computation through pause tokens (Goyal et al., 2024), token-level rationales (Zelikman et al., 2024), or continuous latent states (Hao et al., 2025). However, increasing recurrent computation does not by itself ensure reliable performance beyond the training depth.

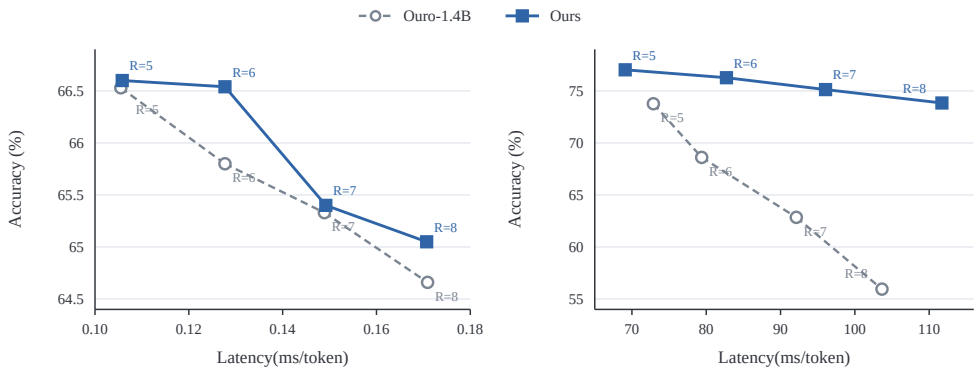

Figure 5: **Accuracy–latency trade-off on Ouro-1.4B.** Original inference and ReMem are compared at $R = 5$–8. **Left:** MMLU accuracy versus per-token latency. **Right:** GSM8K accuracy versus per-token latency. Lines connect configurations in increasing recurrent depth.

Recent work addresses recurrent-depth scaling through more flexible parameter sharing (Bae et al., 2025), retrofitted recurrence with additional training (McLeish et al., 2026), variable-depth trajectory alignment (Jeddi et al., 2026), and recurrent-dynamics regularization (Yang et al., 2026). Related implicit-depth models seek stable fixed points (Bai et al., 2019), with Jacobian regularization improving convergence stability (Bai et al., 2021). Training-Free Looped Transformers instead apply damped updates to repeatedly used frozen blocks (Chen et al., 2026).

Memory and Representation Reuse. Memory-based architectures preserve earlier information through cached hidden states (Dai et al., 2019), compressed memories (Rae et al., 2020), or learned recurrent memory tokens (Bulatov et al., 2022). These methods primarily extend access to information along the sequence dimension. Related work also investigates how intermediate representations contribute to model predictions. Vocabulary-space analyses interpret intermediate representations through the output vocabulary (Geva et al., 2022), while DoLa contrasts intermediate-layer readouts to modify decoding distributions (Chuang et al., 2024). Amnesic probing evaluates the functional role of encoded information through removal-based interventions (Elazar et al., 2021). Distinct from these lines of work, ReMem preserves and reuses intermediate states along the recurrent-depth axis. It leaves the pretrained recurrent transformation unchanged and integrates retained recurrent history directly into subsequent computation. This design requires neither parameter updates nor additional Transformer evaluations at matched recurrent depth.

6 Conclusion and Limitations

In this work, we study recurrent-depth extrapolation in Looped Language Models from the perspective of reusable latent computation. Controlled interventions show that historical recurrent states retain useful computation that can support subsequent recurrent processing. Motivated by this finding, we propose ReMem, a training-free mechanism that maintains an evolving recurrent history and adaptively reuses it with frozen readout evidence. Experiments across two Ouro model scales and diverse benchmarks demonstrate that ReMem improves performance retention as recurrent depth extends beyond the training regime, without parameter updates or additional Transformer evaluations at matched depth. Our results suggest that reliable recurrent-depth scaling depends not only on adding computation, but also on preserving access to computation already performed.

Limitations. ReMem improves the useful range of recurrent computation but does not make performance indefinitely monotonic, and degradation still emerges under sufficiently deep extrapolation. Achieving fully reliable recurrent-depth scaling therefore remains an open challenge. Moreover, ReMem introduces historical access into an existing single-state LoopLM, suggesting that future recurrent architectures may benefit from natively preserving and reusing intermediate computation. Exploring such history-aware designs and more adaptive mechanisms for controlling state reuse is an important direction for future work.

AI USE STATEMENT

Generative AI tools were used only for language polishing and editorial refinement of the manuscript.

ETHICS STATEMENT

We have read the ICLR Code of Ethics and ensure that our paper strictly adheres to these guidelines.

REPRODUCIBILITY STATEMENT

The recurrent memory and historical allocation rules are specified in Sec. 3, with inference pseudocode provided in Algorithm 1 in Appendix A. Benchmark protocols and inference settings are described in Sec. 4.1 and Appendix B. The historical re-entry diagnostic and its control interventions are detailed in Sec. 2 and Appendix C. Additional experiments, ablations, and hyperparameter sensitivity analyses are provided in Appendices D and E. Latency and memory measurement protocols, autoregressive execution semantics, and implementation checks are documented in Appendix F.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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*, 2025.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:690–701, 2019.

Shaojie Bai, Vladlen Koltun, and Zico Kolter. Stabilizing equilibrium models by Jacobian regularization. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 554–565. PMLR, 2021.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, 2020.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026.

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

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. DoLa: Decoding by contrasting layers improves factuality in large language models. In *International Conference on Learning Representations*, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988. Association for Computational Linguistics, 2019.

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

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021.

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.

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. *Advances in Neural Information Processing Systems*, 38: 46242–46293, 2025.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 30–45. Association for Computational Linguistics, 2022.

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.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E. Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

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.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. In *Conference on Language Modeling*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

Mathematical Association of America. American mathematics competitions, 2023. 2023 competition problems.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. In *Conference on Language Modeling*, 2026.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, 2020.

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.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *International Conference on Learning Representations*, 2025.

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.

Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. Stabilizing recurrent dynamics for test-time scalable latent reasoning in looped language models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36:11809–11822, 2023.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. In *Conference on Language Modeling*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019.

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.

APPENDIX

A INFERENCE ALGORITHM DETAILS

Algorithm 1 summarizes ReMem for arbitrary memory size K, with historical fusion applied at every recurrent iteration.

Algorithm 1 Training-Free ReMem Inference

Input: Initial recurrent state $\mathbf{h}_0$; shared Transformer stack $\mathcal{F}_\theta$; frozen readout $\mathcal{O}_\theta$ with output matrix $\mathbf{W}$; recurrent depth $R \in \mathbb{N}_+$; memory size $K \in \mathbb{N}_+$; history budget $B \in [0, 1)$; selector $\mathrm{Sel}(\cdot)$; numerical constant $\epsilon > 0$

Initialize memory $\mathcal{M}_1 \leftarrow (\mathbf{h}_0)$ and an empty evidence cache

for $r = 1, \ldots, R$ **do**

  $\mathbf{z}_r \leftarrow \mathcal{F}_\theta(\mathbf{h}_{r-1})$

  $m_r \leftarrow \min(K, r)$, $\mathcal{S}_r \leftarrow \{r - m_r, \ldots, r - 1\}$

  if $m_r = 1$ **then**

    $\mathbf{g}_{r,r-1} \leftarrow \mathbf{1}$

  else

    for $s \in \mathcal{S}_r$ **do**

      if evidence $\boldsymbol{\rho}_s$ is not cached **then**

        $\mathbf{v}_s \leftarrow \mathrm{Sel}(\mathbf{h}_s)$

        $\boldsymbol{\ell}_s \leftarrow \mathcal{O}_\theta(\mathbf{v}_s)$

        $(\tau_s^{(1)}, \tau_s^{(2)}) \leftarrow \mathrm{TopTwoIndices}(\boldsymbol{\ell}_s)$

        $\Delta\mathbf{w}_s \leftarrow \mathbf{W}_{\tau_s^{(1)}} - \mathbf{W}_{\tau_s^{(2)}}$

        $\boldsymbol{\rho}_s \leftarrow \sqrt{\mathrm{ReLU}\left(\mathbf{v}_s \odot \Delta\mathbf{w}_s\right) + \epsilon\mathbf{1}}$

        Cache $\boldsymbol{\rho}_s$

      end if

    end for

    $\mathbf{q}_r \leftarrow \sum_{u \in \mathcal{S}_r} \boldsymbol{\rho}_u$

    for $s \in \mathcal{S}_r$ **do**

      $\mathbf{g}_{r,s} \leftarrow \boldsymbol{\rho}_s / \mathbf{q}_r$

    end for

  end if

  $\mathbf{h}_r \leftarrow (1 - B)\mathbf{z}_r + B \sum_{s \in \mathcal{S}_r} \mathbf{g}_{r,s} \odot \mathbf{h}_s$

  $\mathcal{M}_{r+1} \leftarrow \mathrm{Last}_K\left(\mathcal{M}_r \parallel (\mathbf{h}_r)\right)$

  Discard states outside $\mathcal{M}_{r+1}$ and their cached evidence

end for

return $\mathbf{h}_R$

Here, $\parallel$ denotes sequence concatenation and Last_K retains the most recent at most K states. Feature-wise operations are element-wise, and allocation vectors are broadcast across positions in the currently processed block. Evidence caching is restricted to unchanged retained states within a fixed input block and selector boundary; it is distinct from autoregressive KV caching, whose execution semantics are described in Appendix F.3.

B IMPLEMENTATION DETAILS

ReMem is applied from the first recurrent iteration. Unless otherwise specified, all experiments use the fixed configuration $K = 2$ and $B = 0.20$ across tasks, recurrent depths, and model scales, without setting-specific tuning. We set the numerical constant in Eq. 8 to $\epsilon = 10^{-8}$ in all experiments.

At each target depth, ReMem and original inference use the same checkpoint and execute exactly R evaluations of the shared Transformer stack. We additionally report Rec.-Llama-3.2 and Rec.-OLMo-2-0425 (McLeish et al., 2026) as cross-model recurrent references. Because their architectures and training procedures differ from Ouro, they are not treated as compute-matched comparisons.

For general language understanding, we evaluate WinoGrande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), ARC-Easy and ARC-Challenge (Clark et al., 2018), and MMLU (Hendrycks et al., 2021). WinoGrande, PIQA, and ARC-Easy use zero-shot continuation scoring, while HellaSwag, ARC-Challenge, and MMLU use 10-shot, 25-shot, and 5-shot evaluation, respectively. For mathematical reasoning, GSM8K (Cobbe et al., 2021) uses 3-shot greedy decoding with a maximum of 256 generated tokens and strict exact-match scoring, while AMC23 (Mathematical Association of America, 2023) is evaluated by exact match. For code generation, HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and MBPP+ (Liu et al., 2023) use greedy decoding and execution-based pass@1. HumanEval uses zero-shot prompting with a maximum of 1,024 generated tokens. Within each matched comparison, ReMem and original inference use identical prompts and scoring or decoding settings.

All performance evaluations are conducted on NVIDIA L40S GPUs. Detailed efficiency measurements and implementation checks are provided in Appendix F.

C Historical Re-entry Control Details

We detail the control interventions used in Fig. 3. All conditions use states from the original Ouro-1.4B recurrent trajectory, without ReMem fusion. We fix the target depth to $R = 8$, the intervention boundary to $j = R - 1 = 7$, the historical source to $\mathbf{h}_4$, and the intervention strength to $\alpha = 0.10$. Let $\boldsymbol{\delta} = \mathbf{h}_4 - \mathbf{h}_j$ denote the historical displacement. Below, $\bar{\mathbf{h}}_j$ denotes the intervened state before the common token-wise RMS matching.

Historical state. The intact-history condition reintroduces the historical state from the same example:

$$\bar{\mathbf{h}}_j^{\text{hist}} = \mathbf{h}_j + \alpha\boldsymbol{\delta} = (1-\alpha)\mathbf{h}_j + \alpha\mathbf{h}_4. \tag{11}$$

This is the reference intervention for assessing whether earlier recurrent computation remains useful to subsequent processing.

Gaussian random direction. We sample an isotropic Gaussian tensor $\boldsymbol{\epsilon}$ of the same shape as $\boldsymbol{\delta}$ and rescale it independently for each token:

$$\mathbf{g}_t = \frac{\|\boldsymbol{\delta}_t\|_2}{\max(\|\boldsymbol{\epsilon}_t\|_2, \varepsilon_{\text{norm}})}\boldsymbol{\epsilon}_t, \qquad \bar{\mathbf{h}}_j^{\text{rand}} = \mathbf{h}_j + \alpha\mathbf{g}, \tag{12}$$

where $\varepsilon_{\text{norm}} > 0$ is a numerical safeguard. This control replaces the historical direction with a random direction while matching its token-wise displacement norm whenever the numerical floor is inactive.

Anti-restoration. We reverse the historical displacement:

$$\bar{\mathbf{h}}_j^{\text{anti}} = \mathbf{h}_j - \alpha\boldsymbol{\delta} = (1+\alpha)\mathbf{h}_j - \alpha\mathbf{h}_4. \tag{13}$$

This deterministic control has the same pre-normalization displacement norm as intact-history reentry but moves in the opposite direction, testing whether the effect depends on moving toward the historical state.

Feature-permuted historical displacement. We apply a seed-specific permutation P of the hidden dimensions to the historical displacement:

$$\bar{\mathbf{h}}_j^{\text{perm}} = \mathbf{h}_j + \alpha P(\boldsymbol{\delta}). \tag{14}$$

The same permutation is used across examples, candidate continuations, and sequence positions within each seeded run. Permuting the displacement preserves its token-wise norm while disrupting its alignment with the original hidden-feature coordinates.

Shuffled history. For each seed, we randomly reorder the examples, pair adjacent examples, and exchange their complete historical-state tensors. Let $\pi(i)$ denote the paired donor for example i, with $\pi(i) \neq i$. Candidate trajectories are exchanged at corresponding option indices and padded sequence positions:

$$\bar{\mathbf{h}}_{j,i}^{\text{shuffle}} = (1-\alpha)\mathbf{h}_{j,i} + \alpha\mathbf{h}_{4,\pi(i)}. \tag{15}$$

Table 4: High-depth performance (%) on Ouro-1.4B.

Rec.	Method	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
$R = 10$	Original	62.59	77.04	68.56	73.19	55.29	62.29	27.90	41.46	47.35	40.48	17.50	66.49	34.94	52.15
	ReMem	**64.56**	**78.29**	**70.89**	**76.64**	**57.08**	**63.66**	**70.36**	**54.88**	**60.85**	**51.59**	**35.00**	**68.52**	**54.54**	**62.16**
$R = 12$	Original	61.64	76.61	67.67	72.98	53.07	59.06	17.74	19.51	35.71	30.16	7.50	65.17	22.12	45.60
	ReMem	**63.54**	**77.69**	**70.15**	**76.56**	**56.83**	**62.40**	**64.37**	**47.56**	**53.97**	**44.97**	**32.50**	**67.86**	**48.67**	**59.14**
$R = 16$	Original	59.91	75.14	65.33	70.12	45.05	45.83	1.67	2.44	4.23	3.70	**5.00**	60.23	3.41	34.40
	ReMem	**61.48**	**77.48**	**68.85**	**75.13**	**56.23**	**58.83**	**35.94**	**24.39**	**31.75**	**26.46**	2.50	**66.33**	**24.21**	**47.18**

Table 5: High-depth performance (%) on Ouro-2.6B.

Rec.	Method	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
$R = 10$	Original	**70.24**	79.76	78.07	77.74	64.85	71.34	63.84	49.39	60.58	50.26	**40.00**	73.67	52.81	64.19
	ReMem	69.30	**80.09**	**78.33**	**79.46**	**65.27**	**73.00**	**76.88**	**53.66**	**65.61**	**53.70**	30.00	**74.24**	**55.97**	**65.94**
$R = 12$	Original	68.90	79.60	77.92	77.36	**65.10**	70.37	48.67	25.00	46.03	39.15	22.50	73.21	36.27	56.42
	ReMem	**69.06**	**80.14**	**78.13**	**79.21**	64.93	**72.23**	**72.40**	**47.56**	**60.85**	**50.79**	**32.50**	**73.95**	**52.82**	**64.35**
$R = 16$	Original	66.69	78.45	77.47	76.05	62.37	62.44	14.71	6.10	17.72	15.87	2.50	70.58	11.38	43.67
	ReMem	**68.67**	**80.09**	**77.87**	**78.75**	**64.59**	**70.40**	**53.98**	**23.78**	**45.50**	**38.36**	**22.50**	**73.40**	**36.82**	**56.77**

This preserves the historical source depth while breaking example correspondence. Unlike the directional controls above, it does not additionally match the donor displacement norm to that of intact-history re-entry. Its effect therefore cannot be attributed solely to the loss of sample correspondence.

Shared evaluation protocol. Each intervention is applied once to the complete prompt–continuation hidden-state tensors at $j = 7$. For condition ν, the final state is computed as

$$\widetilde{\mathbf{h}}_R^{\nu} = \mathcal{F}_{\theta}\left(\mathcal{R}_{\mathbf{h}_j}\left(\bar{\mathbf{h}}_j^{\nu}\right)\right), \tag{16}$$

where $\mathcal{R}_{\mathbf{h}_j}$ is the token-wise RMS-matching operator used in Sec. 2. Each resulting trajectory uses exactly eight shared-stack evaluations, with identical 5-shot MMLU prompts and summed continuation log-likelihood scoring. The common RMS matching controls the scale of the intervened state but does not guarantee equal displacement norms after normalization.

The Gaussian, feature-permuted, and shuffled-history controls use three seeds. For the plotted results, outcomes are averaged across seeds within each example before subject-macro aggregation. Historical re-entry and anti-restoration are deterministic.

D Additional Experiments and Ablations

D.1 Extrapolation to Higher Recurrent Depths

We extend the matched-backbone evaluation to $R \in \{10, 12, 16\}$ on Ouro-1.4B and Ouro-2.6B, using the settings in Appendix B. Tables 4 and 5 report the results for the two model scales.

ReMem preserves higher aggregate performance across both model scales, with the clearest benefits on mathematical reasoning and code generation. The larger separation on these depth-sensitive tasks highlights the value of retaining useful computation as recurrence is extended. This pattern persists on Ouro-2.6B, indicating that historical-state reuse remains useful even when the underlying model has greater capacity.

D.2 Comparison with Simpler Historical Reuse

To examine whether the high-depth gains can be explained by simpler recurrent updates, we compare ReMem with two historical-reuse variants. Single-State Damping retains only the immediately preceding state:

$$\mathbf{h}_r = (1 - B)\mathbf{z}_r + B\mathbf{h}_{r-1}. \tag{17}$$

Table 6: Comparison with simplified historical reuse on Ouro-1.4B (%).

Rec.	Method	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
$R = 10$	Original	62.59	77.04	68.56	73.19	55.29	62.29	27.90	41.46	47.35	40.48	17.50	66.49	34.94	52.15
	Single-State Damping	64.01	77.97	70.35	75.17	56.23	**63.79**	58.68	46.95	54.76	46.83	20.00	67.92	45.44	57.70
	Uniform Historical Fusion	64.25	77.69	70.49	76.01	**57.59**	63.62	63.61	50.61	55.56	47.62	30.00	68.27	49.48	59.73
	ReMem	**64.56**	**78.29**	**70.89**	**76.64**	57.08	63.66	**70.36**	**54.88**	**60.85**	**51.59**	**35.00**	**68.52**	**54.54**	**62.16**
$R = 12$	Original	61.64	76.61	67.67	72.98	53.07	59.06	17.74	19.51	35.71	30.16	7.50	65.17	22.12	45.60
	Single-State Damping	62.19	77.37	69.20	74.87	56.83	60.94	46.25	26.22	41.80	35.19	7.50	66.90	31.39	50.76
	Uniform Historical Fusion	62.83	**77.80**	69.56	75.00	**57.00**	61.15	52.01	38.41	44.71	37.57	12.50	67.22	37.04	53.50
	ReMem	**63.54**	77.69	**70.15**	**76.56**	56.83	**62.40**	**64.37**	**47.56**	**53.97**	**44.97**	**32.50**	**67.86**	**48.67**	**59.14**
$R = 16$	Original	59.91	75.14	65.33	70.12	45.05	45.83	1.67	2.44	4.23	3.70	**5.00**	60.23	3.41	34.40
	Single-State Damping	**61.64**	77.20	66.81	73.27	53.84	51.30	11.30	5.49	8.20	7.94	2.50	64.01	7.09	38.14
	Uniform Historical Fusion	**61.64**	76.77	67.45	73.91	54.69	54.10	21.00	9.15	21.16	19.05	**5.00**	64.76	15.07	42.17
	ReMem	61.48	**77.48**	**68.85**	**75.13**	**56.23**	**58.83**	**35.94**	**24.39**	**31.75**	**26.46**	2.50	**66.33**	**24.21**	**47.18**

Table 7: Effect of ReMem activation schedule on Ouro-1.4B (%).

Rec.	Schedule	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
$R = 8$	Full-Trajectory	**65.04**	78.24	**71.67**	**77.10**	**58.36**	65.05	**73.84**	**60.98**	**67.46**	**57.41**	37.50	**69.24**	**59.44**	**64.79**
	Extrapolation-Only	64.48	**78.67**	71.51	75.67	58.11	**65.97**	68.92	60.37	65.34	56.08	**45.00**	69.07	59.14	64.56
$R = 10$	Full-Trajectory	**64.56**	**78.29**	**70.89**	**76.64**	57.08	63.66	**70.36**	**54.88**	**60.85**	51.59	**35.00**	**68.52**	**54.54**	**62.16**
	Extrapolation-Only	63.30	77.69	70.70	75.21	**57.51**	**64.64**	57.54	**54.88**	60.05	**51.85**	30.00	68.18	50.86	60.31
$R = 12$	Full-Trajectory	**63.54**	**77.69**	**70.15**	**76.56**	**56.83**	62.40	**64.37**	**47.56**	**53.97**	**44.97**	**32.50**	**67.86**	**48.67**	**59.14**
	Extrapolation-Only	62.51	77.31	69.52	74.54	55.29	**63.25**	42.91	39.02	47.88	41.53	27.50	67.07	39.77	54.66
$R = 16$	Full-Trajectory	61.48	**77.48**	**68.85**	**75.13**	**56.23**	**58.83**	**35.94**	**24.39**	**31.75**	**26.46**	**2.50**	**66.33**	**24.21**	**47.18**
	Extrapolation-Only	**62.12**	75.95	67.57	72.90	48.12	58.18	15.62	13.41	23.81	19.31	0.00	64.14	14.43	41.54

Uniform Historical Fusion retains two recent states and allocates the history budget uniformly:

$$\mathbf{h}_r = (1 - B)\mathbf{z}_r + \frac{B}{2}\left(\mathbf{h}_{r-2} + \mathbf{h}_{r-1}\right), \qquad r \geq 2. \tag{18}$$

At $r = 1$, all historical-reuse variants use $\mathbf{h}_1 = (1 - B)\mathbf{z}_1 + B\mathbf{h}_0$. Uniform Historical Fusion and ReMem use the same two-state rolling-memory policy, with each method retaining states from its own trajectory.

All variants use frozen Ouro-1.4B with identical evaluation settings and matched shared Transformer-stack evaluation counts. All historical-reuse variants use $B = 0.20$ without variant-specific tuning. Table 6 reports results at $R \in \{10, 12, 16\}$; the $R = 8$ component ablations are reported in Table 3.

The simplified variants improve over original inference, with Uniform Historical Fusion consistently outperforming Single-State Damping in aggregate. ReMem further improves all three aggregate metrics across the evaluated depths, with the largest advantage on math and coding tasks. Because Uniform Historical Fusion uses the same memory policy and history budget, the additional gains support the importance of adaptive integration alongside multi-state access.

D.3 Effect of ReMem Activation Schedule

Because the main configuration applies ReMem from the first recurrent iteration, we examine whether its high-depth benefit depends on modifying the entire recurrent trajectory. We compare **Full-Trajectory**, which applies ReMem from $r = 1$, with **Extrapolation-Only**, which follows original recurrence through $R_{\text{tr}} = 4$ and activates ReMem only for $r > 4$. For Extrapolation-Only, the initial recurrent memory is formed from $(\mathbf{h}_3, \mathbf{h}_4)$ produced by the original trajectory, after which fused states enter the rolling memory normally. Both schedules use $K = 2$, $B = 0.20$, and identical evaluation settings.

Extrapolation-Only improves aggregate performance over original inference at every evaluated depth, showing that historical reuse remains effective even when the initial recurrent trajectory is left unchanged. At $R = 8$, the two schedules achieve similar aggregate performance, whereas Full-Trajectory becomes increasingly advantageous as recurrence extends further beyond the training depth.

Table 8: Comparison with post-trained Ouro-1.4B at $R = 8$ (%).

Method	Param. updates	General Tasks						Math & Coding Tasks					MC Avg	Gen Avg	All Avg
		Wino.	PIQA	Hella.	ARC-E	ARC-C	MMLU	GSM8K	HumanEval	MBPP	MBPP+	AMC23			
Ouro-1.4B	No	64.56	78.18	70.55	75.46	57.34	64.66	55.95	54.88	60.85	52.12	32.50	68.46	51.26	60.64
Ouro-1.4B-SFT	Yes	**65.27**	77.20	68.46	76.77	**59.73**	**65.52**	68.31	**64.63**	**67.99**	**60.05**	**37.50**	68.83	**59.70**	64.68
ReMem	No	65.04	**78.24**	**71.67**	**77.10**	58.36	65.05	**73.84**	60.98	67.46	57.41	**37.50**	**69.24**	59.44	**64.79**

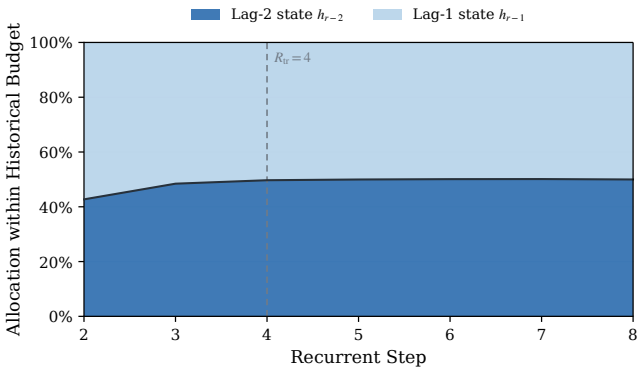

Figure 6: Mean historical allocation on Ouro-1.4B. The dashed line marks $R_{\text{tr}} = 4$.

The separation is especially pronounced on math and coding tasks. Applying ReMem from the beginning changes both the state entering the extrapolation regime and the recurrent history available to later iterations, suggesting that early history shaping becomes increasingly important as the trajectory grows longer.

D.4 Comparison with Post-Trained Inference at High Recurrent Depth

To further assess high-depth performance, we compare original Ouro-1.4B, Ouro-1.4B-SFT, and ReMem at $R = 8$. Ouro-1.4B-SFT denotes the released Ouro-1.4B-Thinking checkpoint, obtained through additional supervised fine-tuning (Zhu et al., 2025). The parameter-update column in Table 8 refers to this additional post-training stage.

ReMem reaches aggregate performance comparable to Ouro-1.4B-SFT, with a higher multiple-choice average and a slightly lower math-and-coding average. The two approaches incur additional costs at different stages: SFT optimizes model parameters before deployment, whereas ReMem maintains and fuses recurrent history during inference. Achieving this level of high-depth performance with the frozen base checkpoint further supports the effectiveness of training-free state reuse.

D.5 Historical Allocation Dynamics

We examine how SCFA distributes the history budget across recurrent steps. For $K = 2$, $\mathbf{g}_r$ denotes the feature-wise allocation to the older state $\mathbf{h}_{r-2}$, while $\mathbf{1} - \mathbf{g}_r$ is assigned to $\mathbf{h}_{r-1}$. Figure 6 averages these allocations over examples and hidden features, excluding the candidate contribution $1 - B$.

The mean allocation shifts from favoring the more recent state toward a balanced split as recurrence progresses. The global budget distribution and feature-wise fusion capture different aspects of reuse: the fused representation depends on how allocation weights align with the retained states, so similar mean shares can produce different updates. The comparison with Uniform Historical Fusion in Table 6 complements this analysis by assessing the performance benefit of adaptive allocation under the same memory policy and total budget.

Table 9: Sensitivity to memory size K on Ouro-1.4B with $B = 0.20$. We report All Avg (%) at each recurrent depth and its mean over $R = 5$–8.

K	$R = 5$	$R = 6$	$R = 7$	$R = 8$	Mean
2	**66.82**	**66.22**	**65.64**	**64.79**	**65.87**
3	63.23	63.34	64.76	64.02	63.84

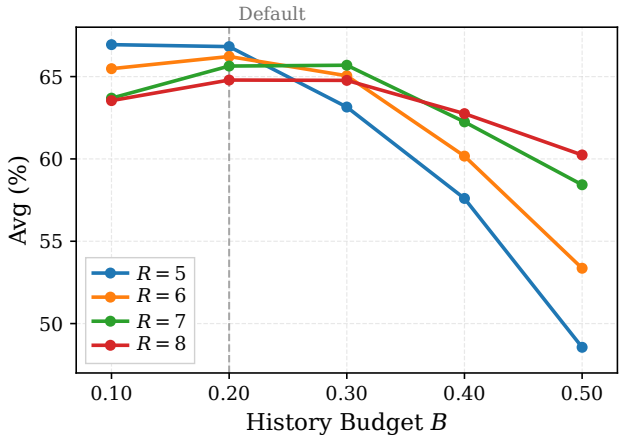

Figure 7: Sensitivity to history budget B on Ouro-1.4B with $K = 2$. Each curve reports All Avg (%) at a different recurrent depth $R \in \{5, 6, 7, 8\}$.

E Hyperparameter Sensitivity

We examine the sensitivity of ReMem to memory size K and history budget B on Ouro-1.4B across $R \in \{5, 6, 7, 8\}$. The main experiments use the fixed configuration $K = 2$ and $B = 0.20$ described in Appendix B. The following analyses characterize sensitivity to these design choices rather than select task- or depth-specific configurations.

Memory Size. Table 9 compares $K = 2$ and $K = 3$ at the fixed history budget $B = 0.20$. Increasing the memory size does not improve aggregate performance at any evaluated depth. Because the total historical coefficient remains B, a larger window changes the available states and their allocation rather than increasing the overall mixing strength. The results therefore suggest that retaining more states does not by itself ensure more effective reuse, supporting a compact memory under the evaluated configuration.

History Budget. Figure 7 examines the history budget with $K = 2$ fixed. Moderate historical mixing performs well across depths, whereas larger budgets reduce aggregate performance. Since increasing B also reduces the candidate coefficient $1-B$, this pattern highlights the balance between reusing previous computation and incorporating the current recurrent update. Despite small depth-dependent shifts in the best-performing budget, the fixed default remains competitive across the evaluated range.

F Detailed Efficiency Evaluation

We further examine the computational cost of historical-state reuse. Measurements use frozen Ouro-1.4B in BF16 on a single NVIDIA L40S GPU with batch size one. Original inference and ReMem use identical evaluation settings and execute R shared Transformer-stack evaluations at depth R. ReMem uses $K = 2$ and $B = 0.20$.

Table 10: Matched-depth latency differences and peak-memory ratios on Ouro-1.4B, relative to original inference.

Task	R	Δ_{lat}	Peak-memory ratio
MMLU	5	$+0.23\%$	$1.000005\times$
	6	-0.00%	$1.000005\times$
	7	$+0.18\%$	$1.000005\times$
	8	-0.13%	$1.000005\times$
GSM8K	5	-5.20%	$1.000004\times$
	6	$+4.20\%$	$1.000004\times$
	7	$+4.27\%$	$1.000004\times$
	8	$+7.78\%$	$0.999169\times$

Table 11: Runtime composition of ReMem on MMLU at $R = 8$ and $K = 2$.

Component	Runtime share
Transformer stack	98.25%
Evidence construction	0.72%
Feature-wise allocation	0.03%
Historical fusion	0.42%
Output head / other	0.58%

F.1 Measurement Protocol

Latency is measured after warm-up using synchronized CUDA events, including Transformer computation, output-head scoring, and ReMem's auxiliary operations, but excluding loading, preprocessing, and disk I/O. Peak memory denotes the maximum CUDA memory allocated during inference. MMLU uses 512 fixed, subject-balanced examples with 20 warm-up iterations per configuration. GSM8K uses 200 fixed prompts with exactly 128 generated tokens, and decoding latency is normalized by the generated-token count. This controlled protocol separates per-token computational cost from differences in output length.

Relative latency differences are reported as $\Delta_{\text{lat}} = (L_{\text{ReMem}}/L_{\text{Original}} - 1) \times 100\%$. Negative differences are not interpreted as evidence of algorithmic acceleration.

F.2 Matched-Depth Latency and Memory

Table 10 shows that historical reuse incurs workload-dependent costs: measured latency changes are small for MMLU, while overhead becomes more visible during deeper GSM8K decoding. Thus, matching Transformer evaluations does not imply equal total inference cost.

Peak allocated memory remains close to that of original inference, although ReMem still requires an $O(KTd)$ historical-state buffer. For fixed K and active-block length T, deeper recurrence repeats evidence construction and fusion without increasing the number of retained states. This distinguishes the recurring computational cost of historical reuse from its bounded history-storage requirement.

F.3 Runtime Breakdown and Implementation Checks

Table 11 shows that Transformer computation dominates ReMem's runtime. Within the auxiliary operations, evidence construction and historical fusion cost more than feature-wise normalization. These shares characterize ReMem's runtime composition rather than its relative overhead against original inference.

During incremental decoding, historical fusion acts only on the current active block, typically one token; previously cached prefix positions are not re-fused. Evidence is recomputed when the prediction boundary advances and reused only for unchanged retained states and selector boundaries. This evidence cache is distinct from the autoregressive KV cache.

In the recorded checks, cached execution agrees with an uncached reference using the same active-block semantics in recurrent states and logits up to numerical precision, with identical predictions and generated tokens. This agreement is limited to the checked settings and does not establish equivalence to re-fusing the entire prefix with each new boundary-dependent gate.