LOOPLA: CAUSAL STATE REFINEMENT FOR LOOPED LINEAR ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers iteratively refine hidden representations at the cost of additional computation. Linear attention can reduce this cost through subquadratic scaling with sequence length. However, the role of its recurrent memory state in looped algorithms has not been systematically studied. We introduce LOOPLA, which refines this memory along both token and loop dimensions. To preserve causality without serial token-by-token computation, LOOPLA uses a Jacobi-style draft-and-consolidate procedure that approximates state-dependent coefficients for parallel scans. Chunk-wise state commits refresh the memory used by subsequent drafts. We train approximately 1B-parameter Gated DeltaNet models from scratch for 100B tokens. At matched parameter count, LOOPLA outperforms baselines without state carry overall and improves the average RULER score by 2.89 points over the no-loop baseline. Under matched FLOPs, it remains comparable, though slightly weaker overall, to the no-loop baseline, with better long-context performance on RULER 2k/4k. This work demonstrates the importance of state refinement along both token and loop dimensions for looped linear attention.

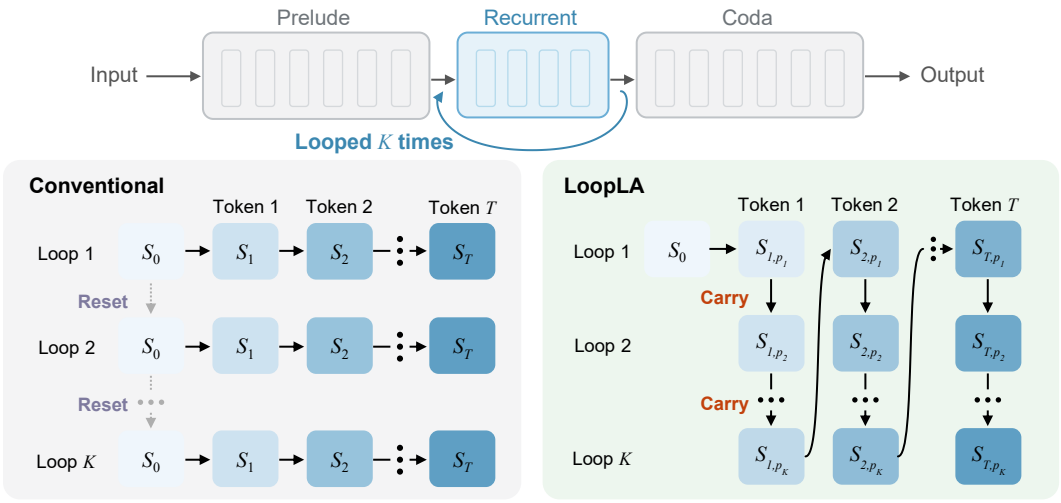

Figure 1: Overview of LOOPLA. Conventional looped linear attention resets its recurrent memory between passes, whereas LOOPLA explicitly carries and refines a causally aligned memory state.

1 INTRODUCTION

Looped Transformers have recently emerged as a promising approach to improving language modeling and reasoning through iterative refinement of hidden representations. Building on Universal Transformers (Dehghani et al., 2019), this approach has expanded from parameter sharing (Takase & Kiyono, 2023; Bae et al., 2025) to latent reasoning (Saunshi et al., 2025; Geiping et al., 2026; Koishekenov et al., 2025; Zhu et al., 2025). This perspective treats loop depth as the amount of computation allocated to iterative refinement, rather than as a fixed architectural depth defined by a stack of distinct layers.

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.

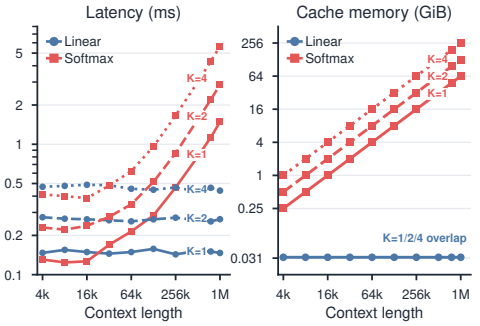

Figure 2: Decode-time scaling of looped softmax and linear attention. Recurrent linear attention keeps cache memory independent of context length and avoids the context-dependent latency growth of softmax attention.

However, looped computation in linear attention models has received limited systematic study. Linear attention (Katharopoulos et al., 2020) replaces quadratic attention with linear-time computation and efficient recurrent-state updates. Gated delta-rule mechanisms (Yang et al., 2025; Kimi Team et al., 2025) are now used in prominent production-scale models, including Kimi K3 (Kimi Team et al., 2026) and Qwen3.8-Flash-Next (Qiu et al., 2026). Their efficiency is particularly appealing for looped computation at long contexts: as illustrated in fig. 2, full-attention loops repeatedly process the entire sequence, incurring quadratic attention costs during training and prefill and context-dependent costs during decoding. Moreover, conventional K-pass loops often need K depth-specific KV caches to preserve their best performance (Bae et al., 2026; Schwethelm et al., 2026b; Nanbeige Lab et al., 2026). Looped linear attention, by contrast, can use its recurrent-state mechanism to reduce both computation and memory overhead, making it a promising option for scaling looped inference to long contexts.

Beyond efficiency, looped linear attention offers a more promising carrier of cross-pass information than hidden representations alone: the recurrent linear state. Conventional looped Transformers rely solely on hidden representations across passes, making what the loop refines difficult to interpret. Although LT2 (Deng et al., 2026) and MixerLoop (Lin et al., 2026) use linear attention, neither explicitly carries its recurrent state across passes; their loop-level information flow therefore remains hidden-state-only. As summarized in fig. 1, explicit state carry instead allows repeated computation on the same token to refine its memory directly. This view complements the online-learning interpretation of the delta rule (Schlag et al., 2021), Gated DeltaNet (Yang et al., 2025), Test-Time Training (TTT) (Sun et al., 2025), and Titans (Behrouz et al., 2026), which treats recurrent memory updates as online optimization. These methods refine memory across successive tokens; cross-pass state carry introduces a second refinement axis over the same token.

Although cross-pass state carry is appealing, it introduces two coupled challenges in practice. **First, it must remain causal.** The recurrent state S_t summarizes the prefix through position t in a single matrix, so it has no token-level entries to which a causal mask can be directly applied. If the final state S_T from one pass simply initializes the next, early tokens can access information from later positions in the previous pass. To prevent this leakage, token t in every pass may receive only a state built from positions no later than t. **Second, causal state carry makes training more sequential.** The state at each position depends on both earlier tokens and the previous pass. Moreover, each token's affine update coefficients are generated from hidden representations that themselves depend on the carried state. These coefficients therefore cannot be prepared independently before the recurrence begins, as required by the standard parallel affine scan. Exact state carry would therefore require serial token-by-token computation across passes, which is impractical for training.

We develop LoopLA to address both challenges: it carries recurrent state causally across loop passes while avoiding the resulting serial training bottleneck. At every pass, token t may read only a state formed by itself and preceding tokens. Because successive passes refine the same token, temporal decay is applied only once and convolutional history is committed only after the final pass. To enable parallel training, LoopLA uses a **Jacobi-style draft-and-consolidate procedure**. The first pass produces a draft hidden trajectory. Each subsequent pass uses the preceding draft to approximate its state-dependent affine coefficients, allowing memory updates across tokens and loop passes to be consolidated in causal order with a parallel prefix scan. This retains the parallel scan structure used in Flash Linear Attention (FLA) (Yang et al., 2024b; Yang & Zhang, 2024). We optionally divide the sequence into Jacobi chunks, committing the consolidated state at each boundary to provide a fresher starting state for the next chunk's drafts. This improves the approximation at the cost of additional sequential dependencies.

Using publicly available training data and an open-source training framework, we train an approximately 1B-parameter Gated DeltaNet model from scratch for 100B tokens. At matched parameter

count, chunked LOOPLA outperforms comparable recurrent-computation baselines without explicit cross-pass state carry overall. It improves the average RULER score across 1k, 2k, and 4k contexts by 2.89 points over the iso-parameter no-loop baseline. Under matched FLOPs, LOOPLA remains comparable, though slightly weaker overall, to the no-loop baseline, with better long-context performance on RULER 2k/4k. Comprehensive ablations further support the value of state refinement along both dimensions, validate LOOPLA's algorithmic choices for the loop dimension, and demonstrate that the additional loop computation indeed refines the recurrent linear state.

2 BACKGROUND AND MOTIVATION

Looped Architecture. Looped Transformers repeatedly apply the same model layers to an evolving hidden state. For a sequence of T tokens, let $h_t^{(k)} \in \mathbb{R}^d$ denote the hidden state of token $t \in \{1, \dots, T\}$ after loop pass $k \in \{1, \dots, K\}$. For a recurrent core with L shared layers $\mathcal{F}_1, \dots, \mathcal{F}_L$, the parameters θ of all L layers are reused at every pass. One loop pass is

$$\begin{aligned} H^{(k)} &= \mathcal{R}_\theta\Big(H^{(k-1)}\Big) = (\mathcal{F}_L \circ \cdots \circ \mathcal{F}_1)\Big(H^{(k-1)}\Big), \quad k = 1, \dots, K, \\ H^{(k)} &= \big[h_1^{(k)}; \dots; h_T^{(k)}\big] \in \mathbb{R}^{T \times d}, \qquad\qquad\qquad\quad t = 1, \dots, T. \end{aligned} \tag{1}$$

Rather than repeating the entire model, we adopt the prelude–recurrent–coda design (Geiping et al., 2026; Koishekenov et al., 2025). Only the middle recurrent core is looped, with each pass receiving the preceding pass's output; the prelude and coda run once to map token embeddings into the hidden residual stream and refined representations to output logits, respectively. The prelude output $H^{(0)}$ serves as the input to the first recurrent pass.

Linear Attention and the Gated Delta Rule. Softmax attention explicitly compares every query with all preceding keys, which requires quadratic computation in the sequence length. Linear attention instead uses a decomposable similarity function and aggregates key–value products before applying each query (Katharopoulos et al., 2020). Omitting normalization for clarity,

$$o_t = \sum_{i=1}^{t} \big(\phi(q_t)^\top \phi(k_i)\big)\, v_i = \underbrace{\left(\sum_{i=1}^{t} v_i \phi(k_i)^\top\right)}_{S_t} \phi(q_t), \qquad S_t = S_{t-1} + v_t \phi(k_t)^\top. \tag{2}$$

Here, ϕ is a feature map applied to queries and keys, and S_t is a fixed-size memory state that summarizes the key–value associations in the prefix. The gated delta rule replaces this additive update with adaptive forgetting and error-correcting writes (Schlag et al., 2021; Yang et al., 2025). Let q_t, k_t, and v_t denote the query, key, and value vectors used by GDN at token t. Its update is

$$\begin{gathered} \widetilde{S}_{t-1} = \alpha_t S_{t-1}, \qquad S_t = \widetilde{S}_{t-1} + \beta_t\big(v_t - \widetilde{S}_{t-1} k_t\big) k_t^\top = \widetilde{S}_{t-1} - \beta_t \nabla_S \ell_t(S)\big|_{S=\widetilde{S}_{t-1}}, \\ \ell_t(S) = \tfrac{1}{2}\|S k_t - v_t\|_2^2, \qquad o_t = S_t q_t. \end{gathered} \tag{3}$$

The gate $\alpha_t \in (0, 1)$ controls how much of the previous memory is retained. The delta term then compares the state's prediction $\widetilde{S}_{t-1} k_t$ with the target v_t and writes the resulting error along k_t, with step size β_t. This correction can be interpreted as one step of online SGD on the memory state for each new token (Sun et al., 2025).

Motivation: State Refinement Along Two Dimensions. Conventional looped Transformers carry $h_t^{(k)}$ across passes, but this hidden-state trajectory does not identify what information is retained or corrected. Linear attention provides another carrier with a clearer role: S_t is the model's explicit memory of preceding key–value associations, and eq. (3) already refines it once whenever a new token arrives. This motivates extending memory refinement across loop passes.

Existing looped linear-attention models such as LT2 (Deng et al., 2026) and MixerLoop (Lin et al., 2026) do not realize this state-level recurrence. LOOPLA instead carries the recurrent state across loop passes. This gives a single memory state **two complementary refinement axes**: successive tokens incorporate new evidence from the sequence, while successive loop passes reconsider how the same token should modify that memory using its refined hidden representation. The next section formalizes this coupled token–loop recurrence and develops a causal, parallel procedure for computing it.

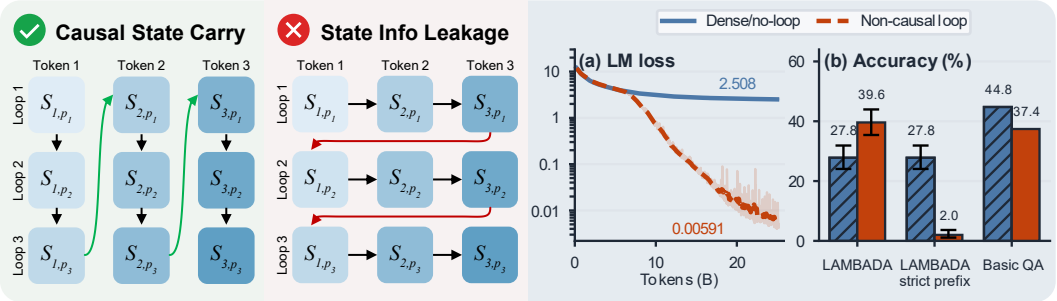

Figure 3: Why cross-pass state carry must remain causal. Carrying a state that contains later-token information into an earlier position allows the model to exploit future context. The non-causal control attains an extremely low training loss and an inflated LAMBADA score under ordinary full-sequence scoring, but collapses under strict-prefix evaluation and also degrades on BasicQA tasks.

3 METHOD

In this section, we describe LOOPLA, a causal state-carry mechanism for looped linear attention. The goal is to let loop passes refine the recurrent memory associated with each token while preserving the left-to-right causal contract and retaining the efficient parallel training structure.

3.1 CAUSAL STATE REFINEMENT

LOOPLA treats looping as refinement of the same token's readable memory rather than as repeated advancement through new tokens. Let p_k denote the k-th loop pass, for $k = 1, \dots, K$. For a given GDN layer, let x_{t,p_k} denote its input representation at token t during pass p_k, and S_{t,p_k} its memory state after the corresponding update, omitting the layer index for clarity. We initialize the memory state of a new sequence as $S_{0,p_K} = 0$. We write the gated delta update with coefficients generated from x_{t,p_k} as $\mathcal{U}_{t,p_k}(S) = SA_{t,p_k} + B_{t,p_k}$, where

$$A_{t,p_k} = \alpha_{t,p_k}\left(I - \beta_{t,p_k} k_{t,p_k} k_{t,p_k}^\top\right), \qquad B_{t,p_k} = \beta_{t,p_k} v_{t,p_k} k_{t,p_k}^\top. \tag{4}$$

The exact causal recurrence is

$$S_{t,p_k} = \begin{cases} \mathcal{U}_{t,p_1}(S_{t-1,p_K}) = S_{t-1,p_K} A_{t,p_1} + B_{t,p_1}, & k = 1, \\ \mathcal{U}_{t,p_k}(S_{t,p_{k-1}}) = S_{t,p_{k-1}} A_{t,p_k} + B_{t,p_k}, & k = 2, \dots, K. \end{cases} \tag{5}$$

The token-first, pass-second ordering is essential to the intended causal state-refinement semantics: $(1, p_1) \to \cdots \to (1, p_K) \to (2, p_1) \to \cdots \to (T, p_K)$. All loop updates for a token precede those for the next token, allowing token t to use its earlier loop-pass writes without accessing future positions. This is not merely a formal constraint. As shown in fig. 3, carrying future-token information across passes produces a misleadingly low training loss and inflated scores on language modeling tasks under ordinary full-sequence scoring, but performance collapses under strict-prefix evaluation and also deteriorates on basic question answering. The apparent gain thus reflects causal leakage rather than improved model quality. We also tested several ways to reduce leakage, but none ensured causality while preserving useful cross-pass communication (details in appendix D).

Beyond preserving causality, cross-pass state carry must distinguish refinement of the same token from advancement to a new token. In LOOPLA, all loop passes refine the same token, so the temporal decay gate is applied only on the first pass, with $\alpha_{t,p_k} = 1$ for $k > 1$ in eq. (4). Likewise, convolutional history is committed only after the final pass, while intermediate passes may still correct the matrix state. The ablations in table 3 support these choices: repeating decay or convolutional-state advancement across passes degrades language modeling and long-context performance.

Composing these K refinement updates gives the complete state transition for token t:

$$S_{t,p_K} = S_{t-1,p_K} \overline{A}_t + \overline{B}_t, \qquad \overline{A}_t = \prod_{k=1}^{K} A_{t,p_k}, \qquad \overline{B}_t = \sum_{k=1}^{K} B_{t,p_k} \prod_{j=k+1}^{K} A_{t,p_j}. \tag{6}$$

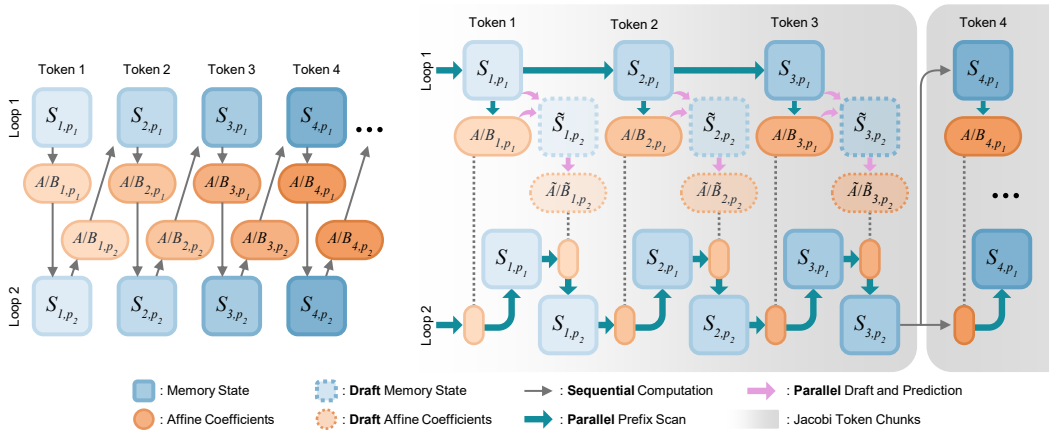

Figure 4: **Left:** Sequential looping completes all loop-pass updates for each token before advancing to the next; **Right:** LOOPLA's chunk-wise Jacobi iteration algorithm uses parallel scans within each chunk and passes the final consolidated memory state to the next chunk. The affine coefficients A/B are generated from hidden states x computed using S, not directly from S; this intermediate computation is omitted for visual clarity.

Ordinary gated delta attention is the special case $K = 1$. For $K > 1$, although this composition has an affine form, **its coefficients depend on the carried state**, preventing direct evaluation with a standard parallel scan. We next describe how to address this dependency and enable parallel training.

3.2 JACOBI ITERATION ALGORITHM

In a conventional linear-attention layer without cross-pass state carry, every pair (A_t, B_t) is computed from the input hidden states before the recurrence begins. A parallel prefix scan computes all token states without stepping through the sequence one token at a time, using the associative composition of the corresponding affine maps:

$$(A_1, B_1) \otimes (A_2, B_2) = \big(A_1 A_2,\ B_1 A_2 + B_2\big). \tag{7}$$

With cross-pass state carry, however, (A_{t,p_k}, B_{t,p_k}) is generated from the layer input x_{t,p_k}, which depends on carried memory states through the preceding computations in the recurrent core. These inputs, and hence the coefficients required by the scan, are therefore unavailable until the relevant preceding token–pass states have been computed. Computing this recurrence directly would require sequential computation across both tokens and loop passes, removing the main training advantage of linear attention.

LOOPLA breaks this dependency with a **Jacobi-style draft-and-consolidate procedure**. The first pass runs an ordinary parallel scan to produce a draft hidden trajectory. This draft does not yet include state corrections made by earlier tokens in later loop passes, but it can be generated in parallel across all tokens using the scan. At each subsequent pass p_k, the preceding draft supplies an approximation to the layer inputs x_{t,p_k} needed to generate (A_{t,p_k}, B_{t,p_k}) for all tokens in parallel.

This procedure is Jacobi-style because each stage constructs its coefficients from the preceding draft rather than from newly updated states within that stage. Once these approximate coefficients are fixed, the updates from passes p_1 through p_k are composed in the causal order of eq. (5). A parallel prefix scan computes the recurrence exactly for these coefficients. The consolidated memory states are used to compute the next hidden trajectory. The procedure repeats until pass p_K.

The approximation lies only in the draft layer inputs used to construct the coefficients; consolidation exactly computes the recurrence defined by those coefficients. The procedure preserves prefix causality while approximating the hidden and memory trajectories of exact sequential computation. More importantly, the parallel scans at each stage can be accelerated using Flash Linear Attention (FLA) kernels (Yang et al., 2024b; Yang & Zhang, 2024).

Table 1: Main comparison after 100B training tokens. Bold indicates the best result within the iso-parameter group.

	Iso-parameter comparison						Reference
Task	No-loop (iso-param)	LT2-style loop	MixerLoop	DeltaProduct	LoopLA (chunk)	LoopLA (sequence)	No-loop (iso-FLOPs)
Language modeling ↓							
LAMBADA (PPL)	10.93	**9.99**	10.64	11.13	10.19	10.04	9.39
WikiText (PPL)	19.34	**18.94**	19.40	19.87	19.03	19.07	18.48
BasicQA ↑ (0-shot)							
ARC-Easy^n	58.33	61.07	60.31	60.02	60.27	**61.49**	59.81
OpenBookQA^n	33.00	34.40	33.00	32.80	32.60	**34.80**	33.60
SciQ^n	78.60	81.90	82.30	81.40	**83.50**	**83.50**	80.20
BoolQ	50.09	55.72	56.02	53.27	51.44	**56.91**	60.46
PIQA^n	70.40	70.57	70.24	70.08	**71.98**	70.29	70.35
HellaSwag^n	47.68	48.85	47.45	46.40	**49.01**	48.25	49.43
WinoGrande	54.78	55.17	53.59	53.12	53.83	**55.25**	54.46
BasicQA (mean)	56.13	58.24	57.56	56.73	57.52	**58.64**	58.33
Reasoning ↑ (5-shot)							
MMLU	26.26	25.48	26.59	26.53	**26.65**	25.62	26.23
ARC-Challenge^n	34.64	34.64	33.62	32.17	**34.73**	32.94	36.77
Reasoning (mean)	30.45	30.06	30.10	29.35	**30.69**	29.28	31.50
Long context (0-shot) ↑							
LongBench-E	**19.21**	18.04	18.62	18.74	19.13	17.08	19.39
RULER (1k)	46.92	47.17	46.42	41.92	**48.90**	46.31	49.48
RULER (2k)	34.87	35.75	38.70	33.07	**39.60**	34.84	36.99
RULER (4k)	29.02	29.40	29.66	24.97	**30.98**	29.01	30.05

3.3 Jacobi Chunks for Memory Refresh

The Jacobi procedure refreshes coefficients between passes, but each stage uses a preceding draft that cannot incorporate state corrections produced by that stage's scan. When each pass spans the full sequence, these corrections become available for coefficient generation only in a subsequent pass.

To provide this feedback to later tokens sooner, LoopLA optionally partitions the sequence into contiguous *Jacobi chunks*. Within each chunk, it performs the draft-and-consolidate procedure from section 3.2 with parallel scans. After completing all K passes, it commits the final consolidated memory state at the chunk boundary to initialize the first draft of the next chunk. This lets later chunks use memory refined through all K passes of preceding chunks, shortening the feedback interval. As illustrated in fig. 4, coefficient computation and scans remain parallel across tokens within each chunk, while chunks are processed sequentially. Based on our chunk-size ablations (table 5) and efficiency measurements (section 4.3), we use $B = 1024$ for sequences of length $4{,}096$ in our experiments. This setting provides a suitable state-refresh frequency across the sequence without introducing excessive sequential dependencies.

4 Experiments

We compare recurrent-computation and state-refinement mechanisms on a common $\sim$1B-parameter pure linear attention model, containing 32 Gated DeltaNet (GDN) layers (Yang et al., 2025). We train all main models from scratch on Qwen3.5-tokenized Dolma 3 Stage A data (Allen Institute for AI, 2025; Qwen Team, 2026), using a context length of 4,096 and a global batch size of 1,024. Looped models use a 12–8×2–12 schedule: the first and last 12 layers are applied once, while the middle eight layers are applied twice, yielding 40 effective layer applications. The evaluation covers language modeling, commonsense question answering, knowledge and reasoning, and long-context

Table 2: Ablation for information carriers. Carrying both hidden representations and recurrent memory yields the strongest overall long-context performance.

Variant	Hidden state	Linear state	Loss ↓	Lamb. ↓	Wiki ↓	BasicQA ↑	LongBench-E ↑	RULER 1k/2k/4k ↑
no-loop	×	×	2.665	31.55	31.44	50.03	13.21	**31.44** / 27.18 / 17.08
LT2-style loop	✓	×	2.653	27.57	**30.76**	50.43	13.82	29.86 / 26.11 / 18.64
LoopLA state only	×	✓	2.666	31.15	31.55	49.93	14.39	28.21 / 17.12 / 11.45
LoopLA	✓	✓	**2.650**	**24.78**	30.80	**50.60**	**14.81**	29.63 / **27.29** / **19.40**

Table 3: Ablation for state advancement. Repeating decay or convolution append across passes weakens language modeling and long-context performance.

Variant	Loss ↓	Lamb. ↓	Wiki ↓	BasicQA ↑	LongBench-E ↑	RULER 1k/2k/4k ↑
LoopLA	**2.650**	**24.78**	**30.80**	50.60	**14.81**	**29.63/27.29/19.40**
+ per-pass decay	2.660	30.31	31.16	**50.74**	13.06	28.83/24.63/16.35
+ per-pass conv. append	2.653	30.74	30.91	50.26	13.41	26.88/24.28/14.97

performance on LongBench-E (Bai et al., 2024) and RULER (Hsieh et al., 2024). Full architectural, data, optimization, checkpoint-conversion, and evaluation details are provided in appendices A and B. Training loss curves for the 100B main comparison and all four 20B ablations are provided in fig. 6.

4.1 Main Results at 100B Tokens

Table 1 compares models trained for 100B tokens. Our iso-parameter baselines include no-loop GDN, LT2-style looping (Deng et al., 2026), MixerLoop (Lin et al., 2026), and DeltaProduct (Siems et al., 2026); none explicitly carries recurrent memory across network-level loop passes. Architecture and parameter-matching details are provided in appendix A.4. We report two LoopLA variants: *chunk* uses $B = 1024$ with memory commits between chunks, while *sequence* applies the Jacobi procedure over the full sequence ($B = 4096$ during training), without intermediate chunk commits. Sequence-level LoopLA achieves the highest BasicQA score in the iso-parameter group, while chunked LoopLA leads on Reasoning and RULER at all three context lengths. Chunking also improves LongBench-E over the sequence variant. Compared with the iso-parameter no-loop baseline, chunked LoopLA improves the average RULER score across 1k, 2k, and 4k contexts by 2.89 points. Complete long-context results are provided in appendix B.2.

To complement the iso-parameter comparison, we use a non-loop 40-layer GDN trained for the same 100B-token budget as an approximate iso-FLOPs control with more independent parameters, matching the 40 full-layer applications of LoopLA's 12–8×2–12 schedule. This matching excludes the additional state-consolidation work, but at $B = 1024$ and $K = 2$, LoopLA requires only $1.07\times$ the training latency and $1.00\times$ the model-body decoding latency of this 40-layer no-loop reference (section 4.3). The 40-layer baseline is stronger in language modeling and Reasoning, whereas chunked LoopLA remains better on RULER at 2k and 4k, showing that its long-context advantages can persist under this compute comparison.

4.2 Ablation Studies

All experiments in this subsection use the same 20B-token budget, data trajectory, model architecture, and optimization setup. We omit Reasoning results from the main ablation tables because performance on MMLU and ARC-Challenge remains near chance at this budget.

Information Carrier. We ablate the two cross-pass information carriers to test the motivation in section 2 that linear-state carry complements hidden-state feedback. As shown in table 2, removing hidden-state feedback while retaining linear-state carry degrades performance, particularly on long-context tasks. Retaining both carriers yields the strongest overall results, supporting their complementary roles. Complete task results are provided in appendix B.3.

Why Repeating the Full GDN Update Fails. We test the state-handling rules in section 3.1 by independently restoring per-pass decay or convolution append while keeping all other settings fixed. As shown in table 3, both changes worsen training loss and perplexity and degrade long-

Table 4: Ablation for state refinement. Enabling both refined reads and cross-chunk state commits improves overall model performance.

Variant	Refined read	Refined-state commit	Loss ↓	Lamb. ↓	Wiki ↓	BasicQA ↑	LongBench-E ↑	RULER 1k/2k/4k ↑
No Pass-2 Write	×	×	2.660	32.66	31.27	49.03	11.92	29.43/24.60/15.49
No Pass-2 Commit	✓	×	2.652	28.86	30.84	49.75	12.85	**31.66**/26.62/16.37
LoopLA	✓	✓	**2.650**	**24.78**	**30.80**	**50.60**	**14.81**	29.63/**27.29**/**19.40**

Table 5: Ablation for Jacobi chunk size. $B = 1024$ provides the strongest overall performance.

Chunk size B	Loss ↓	Lamb. ↓	Wiki ↓	BasicQA ↑	LongBench-E ↑	RULER 1k/2k/4k ↑
$B = 4096$	2.654	25.08	30.93	49.56	12.46	**29.69**/24.57/16.18
$B = 2048$	2.653	30.67	30.99	49.52	12.42	29.59/26.13/15.19
$B = 1024$	**2.650**	24.78	30.80	**50.60**	**14.81**	29.63/**27.29**/**19.40**
$B = 512$	2.654	29.03	30.93	49.61	13.64	26.75/19.66/12.08
$B = 256$	**2.650**	**24.43**	**30.75**	49.45	13.89	26.58/16.36/10.92

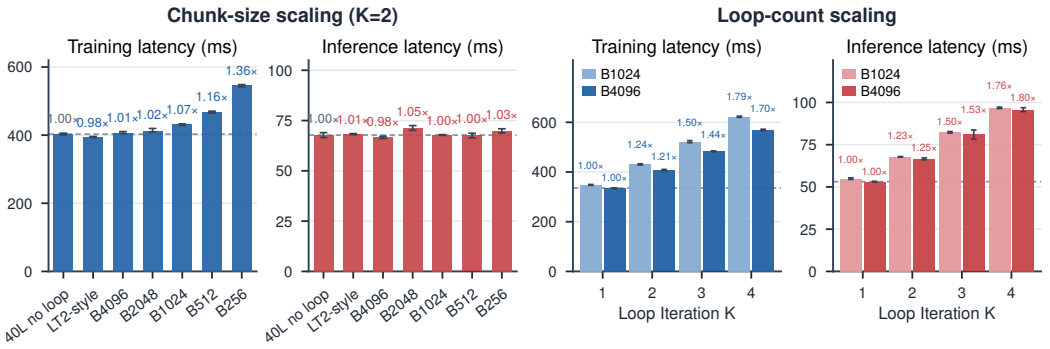

Figure 5: Training and decoding latency scaling with Jacobi chunk size and loop count.

context performance: RULER at 4k falls from 19.40 to 16.35 and 14.97, respectively. These results support treating loop passes as refinements of the same token rather than additional token-time steps. Complete task results are provided in appendix B.4.

Does looping refine the memory state? We test the claim in section 3.1 that loop passes make useful corrections to the recurrent memory with two ablations. Pass 1 and Pass 2 denote the two loop passes (p_1 and p_2) used in these experiments. *No Pass-2 Write* sets the second-pass matrix-write strength to zero while retaining its hidden-state and projection computations. Both current reads and cross-chunk carry therefore use only the state produced by Pass 1, testing whether memory corrections contribute beyond the second pass's other computations. *No Pass-2 Commit* allows Pass 2 to update and read the memory within each chunk, but discards its corrections at the chunk boundary and carries only the Pass 1 state into the next chunk. This retains the immediate benefit of refined reads while removing the benefit of carrying those corrections into subsequent chunks. As shown in table 4, retaining refined reads partially recovers performance, while also committing the refined state yields the strongest overall results. These results support both immediate state refinement and persistent memory carry. Complete task results are provided in appendix B.5.

Jacobi chunk size. We evaluate the chunk-based memory refresh described in section 3.3 by varying $B \in \{256, 512, 1024, 2048, 4096\}$ for 4,096-token sequences. As shown in table 5, more frequent refreshes do not consistently improve model quality, and smaller chunks can degrade long-context performance. Among the tested settings, $B = 1024$ provides the strongest overall long-context results, supporting its use as our default. Complete task results are provided in appendix B.6.

4.3 Training and Inference Efficiency

Looping is particularly attractive for linear-attention models: their recurrent updates and fixed-size memory avoid the context-dependent growth in attention-mixer decoding cost and cache storage seen with softmax attention (fig. 2). Although LoopLA introduces additional state-refinement computation, parallel scans, fused recurrent kernels, and training optimizations such as CUDA

Graphs keep the measured overhead modest (fig. 5). At $B = 1024$ and $K = 2$, LOOPLA requires only $1.07\times$ the training latency and $1.00\times$ the model-body decoding latency of the 40-layer no-loop baseline. As chunk size decreases, decoding latency remains nearly constant, while training latency grows slowly despite the increasing number of serial chunk boundaries. As the loop count increases, both training and decoding latency grow approximately linearly with K over the tested range, despite the additional writes required for state consolidation. Optimization details and measurement protocols are provided in appendix C.

5 RELATED WORK

Looped Transformers. Looped Transformers increase effective depth by repeatedly applying shared layers to evolving hidden representations (Dehghani et al., 2019). Beyond parameter sharing (Takase & Kiyono, 2023; Bae et al., 2025), this repeated computation supports latent reasoning (Saunshi et al., 2025; Geiping et al., 2026) through hidden-state (Hao et al., 2025; Zeng et al., 2026a) or vocabulary-mediated feedback (Zeng et al., 2026b). To control where and how much computation occurs, designs introduce layer-local loops (Gao et al., 2026b) and token-adaptive computation (Bae et al., 2026; Li et al., 2026). Scaling and stability studies (Schwethelm et al., 2026a; Wang et al., 2026; Prairie et al., 2026) further examine this repeated computation, complemented by trillion-token training in Ouro (Zhu et al., 2025) and Nanbeige4.2-3B (Nanbeige Lab et al., 2026). Conventional loops typically retain depth-specific KV caches (Bae et al., 2026; Nanbeige Lab et al., 2026), motivating cache sharing in MELT (Vendrell et al., 2026) and compression in Looped Latent Attention (O'Neill & Reid, 2026).

Linear-attention models. Linear attention provides a complementary form of recurrence, compressing key–value associations into a fixed-size memory updated along the token dimension (Katharopoulos et al., 2020). To regulate retention, RetNet (Sun et al., 2023) introduces fixed forgetting, and Gated Linear Attention (Yang et al., 2024a) makes it input-dependent. Beyond forgetting, DeltaNet (Schlag et al., 2021; Yang et al., 2024b) corrects memory predictions through delta updates and supports parallel training. Gated DeltaNet (Yang et al., 2025) combines delta updates with adaptive forgetting, with Gated DeltaNet-2 (Hatamizadeh et al., 2026) further decoupling erase and write. Viewing these updates as online learning connects memory refinement to TTT (Sun et al., 2025), which updates an inner learner's weights as recurrent memory. In this view, Titans (Behrouz et al., 2026) incorporates momentum and adaptive forgetting.

Recurrent memory refinement. DeltaProduct (Siems et al., 2026) shares the idea of intra-token memory refinement through multiple state updates with step-specific projections. However, these updates occur within a single layer and do not involve a looped Transformer architecture. Related SSM work explores latent refinement (Wang & Reid, 2026), depth recurrence for time series (Farsang et al., 2026), and skip-aware caching (Yu et al., 2026), without explicitly carrying one recurrent state across loops. Among looped linear-attention models, LT2 (Deng et al., 2026) and MixerLoop (Lin et al., 2026) repeat computation without specifying explicit cross-loop recurrent-state carry. To connect network repetition with memory refinement, LOOPLA carries and refines linear memory across shared-network passes under prefix causality, coupling token-wise updates with loop-depth refinement.

6 CONCLUSION

We presented LOOPLA, a looped linear-attention architecture that refines recurrent memory states rather than only hidden token representations. The key challenge is to carry state across loop passes without leaking future-token information or giving up parallel training. LOOPLA addresses this with causal state refinement, Jacobi-style draft-and-consolidate scans, and chunk-wise state commits. At matched parameter count, 100B-token experiments show stronger overall downstream performance than looped baselines without explicit state carry, with the clearest gains on long-context RULER evaluations. Under matched FLOPs, the method is not uniformly stronger than a deeper no-loop model, but remains competitive overall and improves longer-context RULER performance. These results suggest that the recurrent memory exposed by linear attention is a useful refinement axis for looped computation.

AI USE STATEMENT

We used generative AI tools for auxiliary tasks, including polishing the writing, organizing loss and accuracy results into figures and tables, and organizing code for engineering optimizations. All technical ideas, algorithms, mathematical derivations and proofs, experiments, and the narrative and logical structure of the manuscript were designed and written by the authors. The training code, background research, literature citations, training data, and evaluation data used in this work are supported by traceable source artifacts and records. All reported results were produced by actual experiments and verified by the authors. We have reviewed all AI-assisted work, verified all data against the original records, and checked the correctness of the code and written content. We take full responsibility for the final content of this paper.

ETHICS STATEMENT

This work studies a general-purpose large language model architecture and does not introduce an application-specific deployment, additional data collection, or human-subjects research. Its ethical risks are therefore those inherent to general-purpose large language models, and we identify no additional ethical considerations specific to the proposed method.

REPRODUCIBILITY STATEMENT

We provide source code for the complete training pipeline, from data preparation to checkpoint conversion. Appendix A documents the model architectures, data and tokenizer processing, optimization settings, and training dynamics. Appendix B describes checkpoint-conversion checks and evaluation protocols, and provides detailed task results for the main comparison and ablations. Appendix C records the software environment and pinned versions, implementation optimizations, and efficiency measurement procedures. It also describes the supplementary ZIP archive attached to the OpenReview submission, which contains executable code and supporting materials as a detailed reference.

REFERENCES

Allen Institute for AI. Dolma 3 Mix 150B. Hugging Face dataset card, 2025. URL `https://huggingface.co/datasets/allenai/dolma3_mix-150B-1025`.

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

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617, 2026.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026.

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

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

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.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Chunyuan Deng, Yizhe Zhang, Rui-jie Zhu, Yuanyuan Xu, Jiarui Liu, T. S. Eugene Ng, and Hanjie Chen. LT2: Linear-time looped transformers. *arXiv preprint arXiv:2605.20670*, 2026.

Mónika Farsang, Ramin Hasani, Daniela Rus, and Radu Grosu. Looped SSMs: Depth-recurrence and input reshaping for time series classification. *arXiv preprint arXiv:2605.16048*, 2026.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Software, version 0.4.12, 2026a. URL https://github.com/EleutherAI/lm-evaluation-harness/releases/tag/v0.4.12.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the loopies! *arXiv preprint arXiv:2607.16051*, 2026b.

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

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 *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026.

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. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu

Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Kimi Team et al. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. *arXiv preprint arXiv:2510.07358*, 2025.

He Li, Feichen Song, Boyi Zeng, Shixiang Song, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. PonderLM-3: Adaptive token-wise pondering with differentiable masking. *arXiv preprint arXiv:2603.02023*, 2026.

Ruhai Lin, Yiyang Guo, Rui-Jie Zhu, Hao Ye, and Jason K Eshraghian. Allocating recurrent compute in looped language models. *arXiv preprint arXiv:2608.18230*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Nanbeige Lab, Chen Yang, Chengrui Huang, Fufeng Lan, Hanhui Chen, Hao Zhou, Huatong Song, Jiaqi Cao, Jiaying Zhu, Jinlin Niu, Kai Wang, Lisheng Huang, Qiliang Liang, Ran Le, Ruixiang Feng, Shuang Sun, Tao Gu, Tao Zhang, Tianyu Luo, Yang Song, Yun Xing, Yuntao Wen, Ziyao Xu, Zongchao Chen, and Zongqiang Li. Nanbeige4.2-3B: Unlocking agentic capabilities in a compact model. *arXiv preprint arXiv:2607.22083*, 2026.

NVIDIA Corporation. NVIDIA PyTorch Container Release 25.11. NVIDIA NGC container catalog, 2025. URL https://catalog.ngc.nvidia.com/orgs/nvidia/-/containers/pytorch/25.11-py3/tags.

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

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

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

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Qwen Team. Qwen3.5-0.8B-Base. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-0.8B-Base.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.

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*, volume 2025, pp. 14855–14881, 2025.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9355–9366, 2021.

Kristian Schwethelm, Daniel Rückert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026a.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. Depth-adaptive inference of looped language models via continuous depth batching. *arXiv preprint arXiv:2608.09444*, 2026b.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. *Advances in Neural Information Processing Systems*, 38:153738–153782, 2026.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 57503–57522, 2025.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

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, 2023. doi: 10.18653/v1/2023.sustainlp-1.5.

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

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. SMELT: Scaling laws for compute-matched MoE looped transformers. *arXiv preprint arXiv:2609.01343*, 2026.

Wenlong Wang and Fergal Reid. Tiny recursive reasoning with mamba-2 attention hybrid. In *Workshop on Latent & Implicit Thinking – Going Beyond CoT Reasoning*, 2026. URL https://openreview.net/forum?id=AEz0zbLuzg.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Songlin Yang and Yu Zhang. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism. GitHub repository, 2024. URL https://github.com/fla-org/flash-linear-attention.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 56501–56523, 2024a.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37: 115491–115522, 2024b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Zhenxuan Yu, Takeshi Kojima, Yutaka Matsuo, and Yusuke Iwasawa. CHASE: Cache-hole-adapted skip exit for looped state-space language models. *arXiv preprint arXiv:2607.10110*, 2026.

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

Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Zitong Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. PonderLM-2: Pretraining LLM with latent thoughts in continuous space. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL `https://openreview.net/forum?id=yVFxjNzCQm`.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhouhan Lin, et al. PonderLM: Pretraining language models to ponder in continuous space. In *International Conference on Learning Representations*, volume 2026, pp. 91874–91893, 2026b.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Training Implementation Details

A.1 Model Architecture

Our models build on a common Qwen-like pure-GDN backbone. Table 6 summarizes the reference configuration; model-specific differences are listed in appendix A.4.

Table 6: Reference GDN backbone configuration.

Configuration	Setting
Trainable parameters	944,052,224
GDN layers	32
Hidden size	1,024
FFN intermediate size	3,584
Key/value heads	16 / 16
Key/value head dimension	128 / 128
Local causal convolution width	4
Padded vocabulary size	248,320
Activation	SwiGLU
Normalization	RMSNorm, $\epsilon = 10^{-6}$
Linear-layer bias	None
Explicit positional embedding	None
Parameter and main-activation precision	BF16
GDN recurrent matrix-state precision	FP32
Input/output embeddings	Tied

The backbone follows the width, feed-forward, and vocabulary conventions of Qwen3.5-0.8B-Base (Qwen Team, 2026), but replaces every softmax-attention layer with GDN (Yang et al., 2025) and increases the depth to 32 layers. It uses no explicit positional embeddings; causal order is represented through the local causal convolutions and recurrent matrix states. LoopLA and the LT2-style baseline use a 12–8×2–12 prelude–recurrent–coda schedule, which shares the middle eight layers across two applications and produces 40 effective layer applications.

A.2 Data and Tokenizer

We train on Dolma 3 Stage A, distributed as `allenai/dolma3_mix-150B-1025` (Allen Institute for AI, 2025). This general-domain mixture includes natural text, scientific literature, code, mathematics, arXiv documents, and encyclopedic content; we do not add narrow training data targeted at the reported evaluation tasks. All text is tokenized with a fixed snapshot of the Qwen3.5-0.8B-Base tokenizer (Qwen Team, 2026). The tokenizer has 248,077 tokens; the model vocabulary is padded to 248,320, and token 248044 serves as both the end-of-document and padding token. We do not add a beginning-of-sequence token automatically. The train/validation/test split is 998:1:1, and both data construction and index generation use seed 1234.

We first convert source documents into the Megatron-LM indexed-dataset format (Shoeybi et al., 2019), then construct GPTDataset index caches bound to the data recipe, tokenizer, sequence length, and seed. Binary token data produced with different tokenizers are not shared. The 20B-token experiments use one data trajectory over Dolma shard 00. The 100B models share a full-corpus data trajectory with the same sample ordering, constructed independently of the 20B trajectory; consequently, a 20B checkpoint is not treated as a resumable prefix of a 100B run.

A.3 Optimization and Numerical Settings

We use AdamW (Loshchilov & Hutter, 2019) with the settings in table 7. All runs use a warmup–stable–decay schedule (Hu et al., 2024) with a 100B-token horizon: linear warmup for approximately 1B tokens, a stable learning-rate plateau, and cosine decay over approximately 90B–100B tokens. The 20B-token ablations retain this schedule and stop during the plateau, without rescaling the schedule or adding a decay phase before their endpoint. We evaluate the terminal checkpoints at optimizer steps 4,768 and 23,841 for the nominal 20B- and 100B-token budgets, respectively.

Table 7: Optimization and numerical configuration.

Configuration	Setting
Peak/minimum learning rate	3×10^{-4} / 3×10^{-5}
Adam β_1, β_2	0.9, 0.95
Adam ϵ	5×10^{-10}
Weight decay	0.1
Gradient clipping	1.0
Initialization standard deviation	0.02
Learning-rate schedule	Warmup–stable–decay (100B-token horizon)
Warmup	Approximately 1B tokens
Decay	Cosine over approximately 90B–100B tokens
Attention/hidden dropout	0 / 0
Global batch size	1,024
Training precision	BF16
Distributed optimizer	Enabled

Training is distributed across multiple nodes equipped with NVIDIA A100 40GB GPUs. We combine data parallelism with a distributed optimizer and do not use model-parallel techniques such as Fully Sharded Data Parallel (FSDP), tensor parallelism, or pipeline parallelism. We selectively enable activation recomputation when needed to reduce memory pressure.

A.4 Model-Specific Architecture Settings

Table 8 compares the 100B architectures, which share the hidden size, head dimensions, tokenizer, vocabulary, and tied embeddings in table 6. Parameter counts are 944,052,224 for the iso-parameter models and 1,116,495,104 for the 40-layer reference, whose independent layers match the 40 applications of the 12–8×2–12 schedule.

Table 8: Architecture configurations for the main comparison. Depth counts distinct parameterized layers; parameter counts are shown in millions.

Model	Depth	FFN width	Params. (M)	Repeated computation
No-loop (iso-param)	32	3,584	944.05	None
LT2-style loop	32	3,584	944.05	Two passes through layers 13–20
MixerLoop	32	3,584	944.05	Two attention applications per layer in layers 13–20
DeltaProduct	32	2,208	944.05	Two independent writer updates per token in each layer
LoopLA	32	3,584	944.05	Two passes through layers 13–20
No-loop (iso-FLOPs)	40	3,584	1,116.50	None

LT2-style and LoopLA repeat both attention and FFN; MixerLoop repeats only attention. DeltaProduct uses independent convolutional histories for its two writers and one FFN application per layer; narrowing the FFN to 2,208 offsets the added writer parameters. The LoopLA architecture is unchanged across Jacobi chunk sizes. The 20B ablations retain the reference width and depth, changing only the factors in section 4.2.

A.5 Training Dynamics

Fig. 6 shows training LM loss for the 100B comparison and four 20B ablations. Tokens equal optimizer updates × global batch size × sequence length, regardless of loop count. Curves use centered arithmetic smoothing over 0.4194304B tokens (± 50 updates), with endpoint windows restricted to available observations. All runs use seed 1234; the 20B and 100B data trajectories are independent. Final logged losses precede terminal checkpoints by eight updates at 20B and one at 100B.

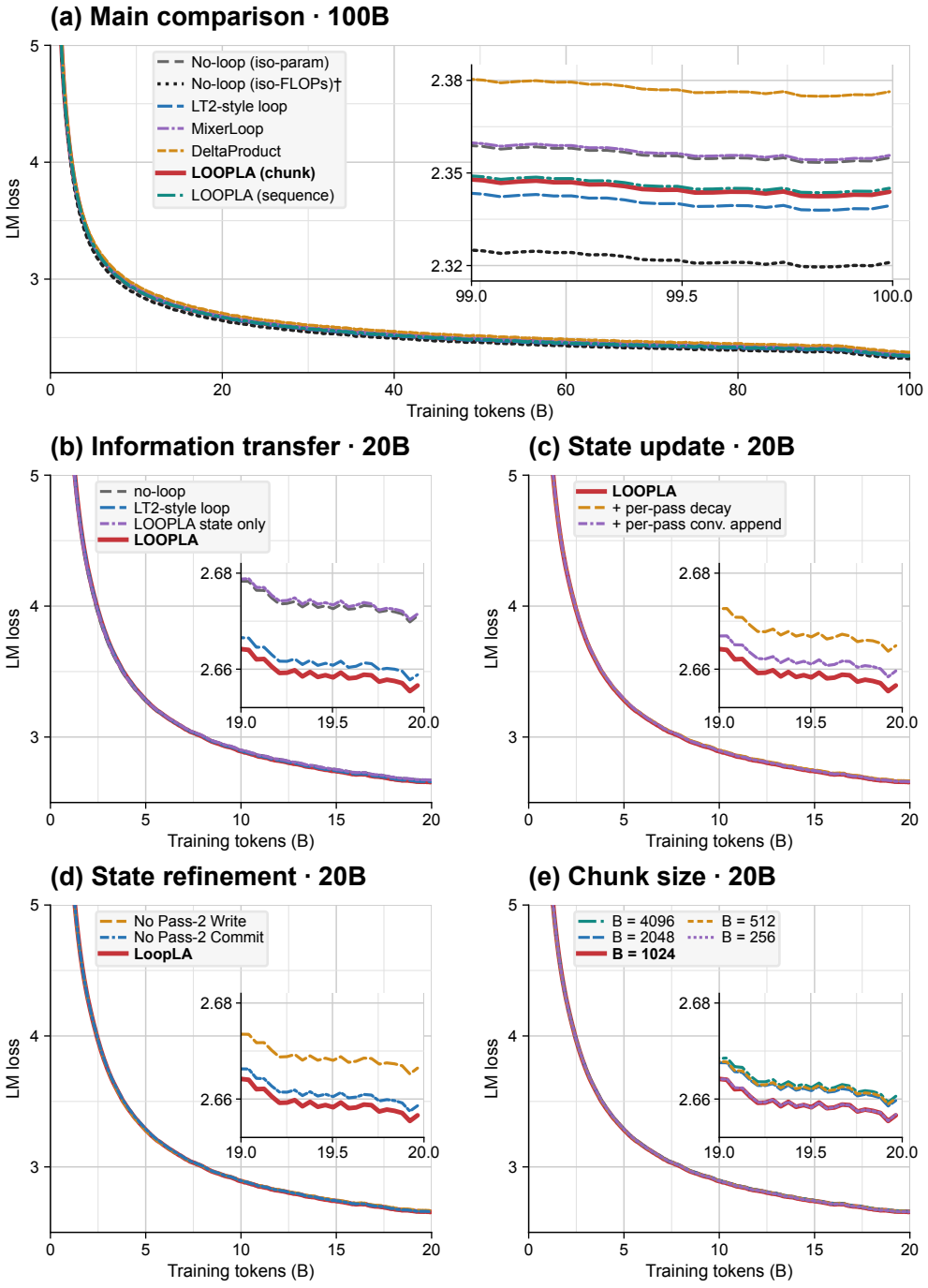

Figure 6: Training LM loss for the experiments in tables 1 to 5. The top panel shows the 100B comparison, followed by four 20B ablations. Insets magnify the final 1B tokens, with identical inset axes across the 20B panels. Main axes clip losses above 5. Native LOOPLA ($B = 1024$ for chunk execution) is highlighted in red. † marks the 40-layer no-loop iso-FLOPs reference, which has more parameters than the 32-layer models.

B Evaluation Implementation Details

B.1 Evaluation Protocol

We validate checkpoint conversion from Megatron Core to Hugging Face (Wolf et al., 2020) by checking complete weight mapping and comparing logits on identical inputs within numerical tolerances. For looped models, we additionally check consistency between full-sequence execution and cached prefill-plus-decode execution.

All scientific evaluations use BF16, seed 1234, and data parallelism over one node with eight H100 GPUs. The standard batch size is eight examples per rank, and generation uses greedy decoding. Table 9 lists the four evaluation suites. The LM Evaluation Harness v0.4.12 (Gao et al., 2026a) executes all tasks in these suites, including the adapted long-context tasks.

Table 9: Offline evaluation suites. All listed tasks use frozen task definitions and data.

Suite	Tasks	Setting
EvalSet1-LM	WikiText; LAMBADA OpenAI	0-shot
EvalSet2-BasicQA	ARC-Easy; OpenBookQA; SciQ; BoolQ; PIQA; HellaSwag; WinoGrande	0-shot
EvalSet3-Reasoning	MMLU; ARC-Challenge	5-shot
EvalSet4-LongContext	LongBench-E 4k; RULER 1k/2k/4k	0-shot

Metric reporting. For both the 100B-token main experiments and the 20B-token ablations, BasicQA is the unweighted arithmetic mean of all seven task accuracies: ARC-Easy, OpenBookQA, SciQ, BoolQ, PIQA, HellaSwag, and WinoGrande. Reasoning is the unweighted mean of MMLU accuracy and ARC-Challenge normalized accuracy.

We omit Reasoning results from the main 20B-token ablation tables because performance on MMLU and ARC-Challenge remains near chance at this budget. The detailed appendix tables report available Reasoning results in gray for reference.

We use the original benchmark definitions for WikiText (Merity et al., 2017), LAMBADA (Paperno et al., 2016), ARC (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), SciQ (Welbl et al., 2017), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), MMLU (Hendrycks et al., 2021), LongBench-E (Bai et al., 2024), and RULER (Hsieh et al., 2024). We use `acc_norm` when a task defines it and `acc` otherwise; superscript n denotes normalized accuracy. MMLU accuracy is weighted by the number of examples across its 57 subjects. Per-sample predictions and raw evaluation outputs are retained.

Both the 100B main comparison and the 20B ablations use the same LongBench-E subset of 592 examples across 13 tasks, with a 4,096-token context window. We use the task definitions in the official dataset release and report the unweighted mean of the 13 task scores. We first select examples whose official length field is below 4,000. Using the fixed tokenizer and prompt templates, we then retain only examples whose tokenized prompt length plus the task-specific generation budget fits within 4,096 tokens. The selected prompts are evaluated without truncation. RULER is evaluated separately at 1,024, 2,048, and 4,096 tokens. We retain every leaf-task result at each length rather than relying only on an aggregate that could hide floor, ceiling, or single-task-dominance effects.

B.2 Detailed Long-Context Results

Tables 10 to 13 report all 13 LongBench-E tasks and all 13 RULER tasks at each of the three context lengths for the 100B-token models in table 1. Each aggregate is an unweighted mean over its 13 tasks. Scores are multiplied by 100, with higher values better. Model order and column grouping follow the main table. Bold indicates the best result within the iso-parameter group. The rightmost column reports the iso-FLOPs reference.

Table 10: LongBench-E task results with a 4k context window. Tasks use their benchmark-defined F1, ROUGE, classification, counting, retrieval, or code-similarity scores. Passage Retrieval is the English task.

	Iso-parameter comparison						Reference
Task	No-loop (iso-param)	LT2-style loop	MixerLoop	DeltaProduct	LoopLA (chunk)	LoopLA (sequence)	No-loop (iso-FLOPs)
2WikiMultihopQA	13.82	12.43	12.00	12.88	**14.62**	7.50	11.50
GovReport	11.86	11.37	12.46	**13.02**	10.17	12.21	12.95
HotpotQA	**9.91**	8.61	6.69	9.67	6.70	8.89	8.75
LCC	37.75	30.61	36.61	**39.08**	34.59	34.96	41.34
MultiNews	**14.56**	11.16	9.92	11.52	12.97	10.85	14.13
MultiFieldQA-en	11.42	12.55	12.02	12.71	11.46	**13.27**	11.74
Passage Count	8.80	1.61	7.99	1.91	**9.34**	7.39	7.92
Passage Retrieval	10.00	10.00	12.00	13.52	**15.33**	10.00	4.00
Qasper	5.25	**5.75**	5.55	4.41	4.87	5.53	5.45
RepoBench-P	32.00	32.60	22.00	**33.00**	32.10	26.30	35.00
SAMSum	**31.50**	31.19	28.45	27.89	26.03	27.57	29.36
TREC	16.92	23.08	**29.23**	27.69	20.00	18.46	29.23
TriviaQA	45.97	43.54	47.21	36.25	**50.53**	39.07	40.70
Mean	**19.21**	18.04	18.62	18.74	19.13	17.08	19.39

Table 11: RULER task results at 1,024 tokens. NIAH denotes needle-in-a-haystack retrieval; CWE and FWE denote common- and frequent-word extraction.

	Iso-parameter comparison						Reference
Task	No-loop (iso-param)	LT2-style loop	MixerLoop	DeltaProduct	LoopLA (chunk)	LoopLA (sequence)	No-loop (iso-FLOPs)
NIAH single-key 1	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00
NIAH single-key 2	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00
NIAH single-key 3	92.20	76.20	97.00	84.80	**99.20**	92.00	99.60
NIAH multi-key 1	33.60	36.00	34.20	26.60	36.60	**38.00**	38.00
NIAH multi-key 2	1.00	2.00	2.40	1.20	**2.60**	2.40	2.60
NIAH multi-key 3	0.00	0.20	0.40	0.00	0.40	**0.60**	0.40
NIAH multi-query	66.25	**68.10**	62.90	47.40	66.05	64.85	65.05
NIAH multi-value	63.25	**64.85**	57.20	53.95	61.90	49.05	75.35
CWE	50.62	**51.68**	46.04	37.68	39.22	50.22	40.70
FWE	30.27	19.20	24.80	3.13	**37.47**	17.20	39.73
QA (HotpotQA)	33.14	**34.29**	33.14	31.41	32.28	33.72	32.85
QA (SQuAD)	36.02	38.95	34.08	35.62	**41.02**	39.45	36.05
Variable tracking	3.60	21.76	11.28	**23.20**	18.96	14.56	12.96
Mean	46.92	47.17	46.42	41.92	**48.90**	46.31	49.48

Table 12: RULER task results at 2,048 tokens. NIAH denotes needle-in-a-haystack retrieval; CWE and FWE denote common- and frequent-word extraction.

	Iso-parameter comparison						Reference
Task	No-loop (iso-param)	LT2-style loop	MixerLoop	DeltaProduct	LoopLA (chunk)	LoopLA (sequence)	No-loop (iso-FLOPs)
NIAH single-key 1	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	100.00
NIAH single-key 2	98.40	**100.00**	**100.00**	98.60	**100.00**	**100.00**	100.00
NIAH single-key 3	65.40	66.00	86.40	74.60	**92.00**	66.40	66.20
NIAH multi-key 1	29.60	**37.20**	34.20	29.60	35.00	33.20	29.80
NIAH multi-key 2	0.00	**0.40**	**0.40**	0.00	0.00	0.00	0.20
NIAH multi-key 3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00
NIAH multi-query	37.15	43.30	42.60	25.80	**45.10**	44.55	44.30
NIAH multi-value	37.20	33.05	**49.60**	32.70	35.90	38.35	46.35
CWE	23.44	**27.88**	19.06	13.90	17.46	21.42	20.44
FWE	23.33	11.47	23.73	3.13	**30.13**	6.33	25.73
QA (HotpotQA)	20.40	22.00	22.80	21.60	**23.60**	20.80	22.20
QA (SQuAD)	15.62	18.60	16.85	17.12	18.13	**19.05**	17.22
Variable tracking	2.80	4.88	7.40	12.92	**17.52**	2.84	8.40
Mean	34.87	35.75	38.70	33.07	**39.60**	34.84	36.99

Table 13: RULER task results at 4,096 tokens. NIAH denotes needle-in-a-haystack retrieval; CWE and FWE denote common- and frequent-word extraction.

	Iso-parameter comparison						Reference
Task	No-loop (iso-param)	LT2-style loop	MixerLoop	DeltaProduct	LoopLA (chunk)	LoopLA (sequence)	No-loop (iso-FLOPs)
NIAH single-key 1	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	99.80	100.00
NIAH single-key 2	51.80	79.00	75.00	54.00	**88.20**	80.00	66.40
NIAH single-key 3	50.00	43.60	39.40	48.60	50.60	**58.40**	41.80
NIAH multi-key 1	23.20	**26.80**	22.60	18.40	25.00	20.60	24.40
NIAH multi-key 2	0.00	**0.40**	0.20	0.00	0.00	0.00	0.00
NIAH multi-key 3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00
NIAH multi-query	**24.05**	22.40	19.45	11.20	15.15	20.20	25.40
NIAH multi-value	24.25	12.65	**26.45**	9.85	13.90	17.35	25.70
CWE	39.80	**43.14**	30.48	25.80	22.54	25.92	28.74
FWE	26.40	12.13	**30.13**	7.53	29.87	8.47	32.00
QA (HotpotQA)	**18.60**	18.40	18.00	17.40	16.20	16.40	18.60
QA (SQuAD)	14.95	**19.33**	16.55	16.45	18.78	18.42	17.18
Variable tracking	4.24	4.28	7.32	15.44	**22.52**	11.56	10.44
Mean	29.02	29.40	29.66	24.97	**30.98**	29.01	30.05

B.3 Detailed Carrier-Ablation Results

Table 14 provides the detailed task results, including available Reasoning scores, for the four 20B-token variants in table 2. LongBench-E uses a 4k context window.

Table 14: Complete task results for the 20B-token information-carrier ablation. RULER entries list 1k/2k/4k scores.

Task	No-loop	LT2-style loop	LoopLA (state only)	LoopLA
BasicQA (0-shot)				
ARC-Easy^n	46.55	46.46	**48.15**	46.55
SciQ^n	68.00	69.30	69.30	**69.50**
PIQA^n	63.22	**64.91**	64.36	64.31
HellaSwag^n	35.20	35.30	35.33	**35.70**
OpenBookQA^n	30.20	**31.20**	30.20	30.60
BoolQ	**57.52**	55.14	52.75	56.88
WinoGrande	49.49	**50.67**	49.41	**50.67**
Mean	50.03	50.43	49.93	**50.60**
Reasoning (5-shot)				
MMLU	25.20	26.28	25.23	25.72
ARC-C^n	27.05	26.11	25.26	25.94
Mean	26.13	26.19	25.24	25.83
LongBench-E (4k)				
2WikiMultihopQA	6.98	6.51	**7.51**	7.23
GovReport	7.63	5.88	**8.23**	5.15
HotpotQA	**7.97**	5.96	5.12	6.72
LCC	32.69	34.81	**40.62**	35.24
MultiNews	8.28	**9.76**	8.27	6.99
MultiFieldQA-en	9.34	**10.36**	9.19	6.65
Passage Count	**8.50**	7.33	5.87	3.39
Passage Retrieval	**16.82**	3.33	3.33	10.00
Qasper	3.76	**4.00**	3.94	3.84
RepoBench-P	31.10	34.60	35.10	**39.00**
SAMSum	9.68	**15.24**	13.19	12.75
TREC	10.77	23.08	27.69	**38.46**
TriviaQA	18.20	18.74	**19.06**	17.14
Mean	13.21	13.82	14.39	**14.81**
RULER (1k/2k/4k)				
NIAH S1	**100.0**/99.8/**100.0**	99.8/99.8/99.6	95.0/94.2/84.8	**100.0/100.0/100.0**
NIAH S2	**100.0/99.0**/22.6	99.8/**99.0**/48.2	95.8/34.6/8.8	99.8/97.0/**48.8**
NIAH S3	**82.4**/57.0/**19.6**	65.6/53.4/16.6	71.4/28.0/13.2	62.8/**64.8**/17.4
NIAH MK1	**24.4/23.6**/17.6	24.2/20.4/18.6	17.6/16.8/9.8	22.6/22.2/**21.2**
NIAH MK2	0.4/**0.6**/0.0	0.2/0.4/0.0	**0.6**/0.0/**0.2**	0.0/0.0/0.0
NIAH MK3	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**
NIAH MQ	**22.2/24.3**/16.3	19.8/20.0/10.8	14.7/9.2/2.4	21.8/23.0/**16.7**
NIAH MV	19.5/**23.1**/16.3	17.4/16.9/15.6	15.7/11.5/6.4	**21.6**/22.1/**18.4**
CWE	**9.64**/0.54/**3.54**	9.42/0.20/2.52	3.86/0.04/0.24	7.64/**0.56**/3.00
FWE	2.00/0.40/0.20	**3.47**/1.20/1.93	2.00/0.33/0.13	3.40/**1.93/2.67**
Variable tracking	3.32/2.80/3.24	4.40/**7.32/7.64**	**5.24**/7.00/5.92	2.36/2.92/2.40
QA (SQuAD)	**22.22**/8.07/9.87	19.62/7.55/**10.22**	19.80/**8.13**/7.75	17.97/7.77/8.93
QA (HotpotQA)	22.77/**14.20/12.80**	24.50/13.20/10.60	25.07/12.80/9.20	**25.36**/12.60/**12.80**
Mean	**31.44**/27.18/17.08	29.86/26.11/18.64	28.21/17.12/11.45	29.63/**27.29/19.40**

RULER NIAH S, MK, MQ, and MV denote single-key, multi-key, multi-query, and multi-value retrieval; CWE/FWE denote common-/frequent-word extraction. All four variants use the explicit-colon prompt for NIAH S3. Reasoning scores are shown in gray for reference.

B.4 Detailed State-Advancement Ablation Results

Table 15 reports the detailed task results, including available Reasoning scores, for the three 20B-token variants in table 3. LongBench-E uses a 4k context window.

Table 15: Complete task results for the 20B-token state-advancement ablation. RULER entries list 1k/2k/4k scores.

Task	LoopLA	+ per-pass decay	+ per-pass conv. append
BasicQA (0-shot)			
ARC-Easy^n	46.55	**48.95**	46.59
SciQ^n	**69.50**	68.70	66.90
PIQA^n	64.31	**65.07**	64.20
HellaSwag^n	**35.70**	**35.70**	35.48
OpenBookQA^n	**30.60**	30.20	30.20
BoolQ	56.88	**57.16**	55.41
WinoGrande	50.67	49.41	**53.04**
Mean	50.60	**50.74**	50.26
Reasoning (5-shot)			
MMLU	25.72	24.92	25.78
ARC-C^n	25.94	26.79	26.54
Mean	25.83	25.85	26.16
LongBench-E (4k)			
2WikiMultihopQA	**7.23**	5.93	5.72
GovReport	5.15	7.71	**8.15**
HotpotQA	6.72	4.80	**7.44**
LCC	**35.24**	31.81	33.72
MultiNews	6.99	**10.26**	8.36
MultiFieldQA-en	6.65	9.27	**10.63**
Passage Count	3.39	**6.45**	3.54
Passage Retrieval	**10.00**	**10.00**	**10.00**
Qasper	3.84	**4.32**	3.26
RepoBench-P	**39.00**	28.20	37.40
SAMSum	12.75	12.37	**13.86**
TREC	**38.46**	27.69	20.00
TriviaQA	**17.14**	10.94	12.19
Mean	**14.81**	13.06	13.41
RULER (1k/2k/4k)			
NIAH S1	**100.0/100.0/100.0**	99.8/**100.0**/99.8	**100.0/100.0**/99.8
NIAH S2	99.8/97.0/**48.8**	**100.0**/94.4/32.8	**100.0/100.0**/24.8
NIAH S3	**62.8/64.8/17.4**	60.8/44.0/**17.4**	55.6/39.8/11.4
NIAH MK1	22.6/**22.2/21.2**	22.2/21.8/17.4	**25.0**/22.0/17.2
NIAH MK2	0.0/0.0/**0.0**	**0.2/0.2/0.0**	0.0/**0.2/0.0**
NIAH MK3	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**
NIAH MQ	21.8/**23.0/16.7**	**22.2**/21.7/6.5	5.7/10.1/2.2
NIAH MV	**21.6/22.1/18.4**	15.7/13.4/7.0	5.6/10.5/4.2
CWE	**7.64/0.56**/3.00	7.50/0.12/**3.04**	6.12/0.18/0.58
FWE	**3.40/1.93/2.67**	0.80/0.20/0.33	0.20/0.20/0.07
Variable tracking	2.36/2.92/2.40	5.04/6.72/10.80	**7.60/10.60/12.64**
QA (SQuAD)	**17.97**/7.77/8.93	16.10/7.65/7.28	17.20/**10.28/10.52**
QA (HotpotQA)	25.36/**12.60/12.80**	24.50/10.00/10.20	**26.51**/11.80/11.20
Mean	**29.63/27.29/19.40**	28.83/24.63/16.35	26.88/24.28/14.97

NIAH abbreviations follow table 14. All three variants use the explicit-colon prompt for NIAH S3. Reasoning scores are shown in gray for reference.

B.5 Detailed State-Refinement Ablation Results

Table 16 reports the detailed task results, including available Reasoning scores, for the three 20B-token variants in table 4. LongBench-E uses a 4k context window.

Table 16: Complete task results for the 20B-token state-refinement ablation. RULER entries list 1k/2k/4k scores.

Task	No Pass-2 Write	No Pass-2 Commit	LoopLA
BasicQA (0-shot)			
ARC-Easy^n	46.38	**47.14**	46.55
SciQ^n	67.60	68.20	**69.50**
PIQA^n	64.04	**64.53**	64.31
HellaSwag^n	35.42	**36.06**	35.70
OpenBookQA^n	29.80	30.40	**30.60**
BoolQ	51.35	52.02	**56.88**
WinoGrande	48.62	49.88	**50.67**
Mean	49.03	49.75	**50.60**
Reasoning (5-shot)			
MMLU	25.51	25.71	25.72
ARC-C^n	27.13	28.16	25.94
Mean	26.32	26.93	25.83
LongBench-E (4k)			
2WikiMultihopQA	7.51	**9.66**	7.23
GovReport	6.30	**6.91**	5.15
HotpotQA	6.04	6.41	**6.72**
LCC	30.34	33.08	**35.24**
MultiNews	**9.53**	8.74	6.99
MultiFieldQA-en	**11.30**	10.79	6.65
Passage Count	7.84	**9.29**	3.39
Passage Retrieval	0.00	2.00	**10.00**
Qasper	**4.17**	3.68	3.84
RepoBench-P	27.50	25.00	**39.00**
SAMSum	10.82	11.56	**12.75**
TREC	13.85	16.92	**38.46**
TriviaQA	19.72	**23.01**	17.14
Mean	11.92	12.85	**14.81**
RULER (1k/2k/4k)			
NIAH S1	99.2/99.8/99.4	99.8/99.6/**100.0**	**100.0/100.0/100.0**
NIAH S2	99.2/88.2/22.4	**100.0**/96.4/12.8	99.8/**97.0/48.8**
NIAH S3	60.6/48.2/14.4	**73.8**/52.2/15.2	62.8/**64.8/17.4**
NIAH MK1	**24.2**/20.2/16.8	21.8/**22.4**/19.6	22.6/22.2/**21.2**
NIAH MK2	**0.4**/0.0/0.0	0.2/**0.2/0.2**	0.0/0.0/0.0
NIAH MK3	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**
NIAH MQ	22.1/17.0/12.8	**24.5/23.6/17.9**	21.8/23.0/16.7
NIAH MV	21.6/19.8/14.6	**24.1/23.4/18.8**	21.6/22.1/18.4
CWE	4.04/0.00/0.26	**14.38/1.20/3.82**	7.64/0.56/3.00
FWE	0.40/0.13/0.27	2.13/1.60/1.20	**3.40/1.93/2.67**
Variable tracking	**8.36/4.24**/1.48	3.20/3.40/0.72	2.36/2.92/**2.40**
QA (SQuAD)	19.25/10.50/9.00	**21.73/10.72/10.48**	17.97/7.77/8.93
QA (HotpotQA)	23.34/11.80/10.00	**25.94**/11.40/12.20	25.36/**12.60/12.80**
Mean	29.43/24.60/15.49	**31.66**/26.62/16.37	29.63/**27.29/19.40**

NIAH abbreviations follow table 14. All three variants use the explicit-colon prompt for NIAH S3. Reasoning scores are shown in gray for reference.

B.6 Detailed Chunk-Size Ablation Results

Table 17 reports the complete task results for the five 20B-token models in table 5. Each model is evaluated with its native training chunk size. LongBench-E uses a 4k context window.

Table 17: Complete task results for the 20B-token chunk-size ablation. RULER entries list 1k/2k/4k scores.

Task	$B = 4096$	$B = 2048$	$B = 1024$	$B = 512$	$B = 256$
BasicQA (0-shot)					
ARC-Easy^n	47.10	**47.85**	46.55	47.26	47.18
SciQ^n	68.60	69.80	69.50	68.60	**69.90**
PIQA^n	**65.34**	64.69	64.31	63.98	64.80
HellaSwag^n	**36.02**	35.84	35.70	35.92	35.50
OpenBookQA^n	29.40	30.40	**30.60**	29.40	29.60
BoolQ	51.53	49.82	**56.88**	53.39	51.25
WinoGrande	48.93	48.22	**50.67**	48.70	47.91
Mean	49.56	49.52	**50.60**	49.61	49.45
Reasoning (5-shot)					
MMLU	25.30	24.88	25.72	25.13	24.78
ARC-C^n	26.19	26.79	25.94	27.05	28.33
Mean	25.75	25.84	25.83	26.09	26.55
LongBench-E (4k)					
2Wiki MultihopQA	6.21	6.53	7.23	5.12	**8.46**
GovReport	8.13	**9.07**	5.15	8.53	6.64
HotpotQA	5.33	4.71	**6.72**	6.59	6.23
LCC	27.60	31.36	**35.24**	34.91	27.60
MultiNews	9.76	8.70	6.99	9.29	**9.89**
MultiField QA-en	7.99	8.32	6.65	9.04	**10.84**
Passage Count	8.15	8.77	3.39	**9.64**	8.65
Passage Retrieval	5.45	4.50	10.00	**11.67**	3.67
Qasper	**3.88**	3.79	3.84	3.57	3.80
RepoBench-P	25.60	17.60	**39.00**	26.70	35.90
SAMSum	**18.00**	10.16	12.75	17.44	14.15
TREC	15.38	32.31	**38.46**	13.85	26.15
TriviaQA	20.52	15.59	17.14	**21.05**	18.64
Mean	12.46	12.42	**14.81**	13.64	13.89
RULER (1k/2k/4k)					
NIAH S1	**100.0**/99.8/99.6	**100.0**/99.6/99.4	**100.0/100.0/100.0**	99.8/99.0/95.8	99.8/97.6/94.0
NIAH S2	**100.0/99.2**/34.6	99.6/98.6/18.4	99.8/97.0/**48.8**	89.6/53.0/10.0	53.0/20.8/2.2
NIAH S3	60.6/44.4/13.8	**66.8**/41.0/9.2	62.8/**64.8/17.4**	55.6/40.0/12.4	64.0/31.6/9.0
NIAH MK1	19.8/18.4/14.0	**23.4/25.4**/17.0	22.6/22.2/**21.2**	22.8/19.2/9.4	21.8/17.2/3.2
NIAH MK2	0.2/0.0/**0.2**	**0.4**/0.0/**0.2**	0.0/0.0/0.0	0.0/0.0/0.0	0.2/**0.2**/0.0
NIAH MK3	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**	**0.0/0.0/0.0**
NIAH MQ	19.7/8.7/5.4	**23.7/23.3**/12.0	21.8/23.0/**16.7**	13.7/6.9/1.0	21.6/8.6/1.2
NIAH MV	18.5/15.2/10.1	**23.5/24.1**/16.2	21.6/22.1/**18.4**	18.5/10.7/3.1	21.9/7.9/0.8
CWE	7.86/0.32/2.04	2.66/0.00/0.46	7.64/0.56/3.00	3.26/0.02/0.40	**16.38/3.54/7.48**
FWE	**7.47/4.40/4.33**	2.07/1.13/1.33	3.40/1.93/2.67	0.07/0.00/0.13	0.73/0.13/0.20
Variable tracking	5.64/**6.88/5.72**	**6.36**/5.96/3.84	2.36/2.92/2.40	4.20/5.20/5.08	5.64/2.40/1.08
QA (SQuAD)	**20.25/9.67**/9.22	16.62/8.00/8.08	17.97/7.77/8.93	18.65/8.47/9.07	19.25/9.55/**9.87**
QA (HotpotQA)	**25.94**/12.40/11.40	19.60/12.60/11.40	25.36/12.60/12.80	21.61/**13.20**/10.60	21.33/**13.20/13.00**
Mean	**29.69**/24.57/16.18	29.59/26.13/15.19	29.63/**27.29/19.40**	26.75/19.66/12.08	26.58/16.36/10.92

NIAH abbreviations follow table 14. All five models use the explicit-colon prompt for NIAH S3. The $B = 4096$ BasicQA and Reasoning results are recovered from the historical evaluation log. Reasoning scores are shown in gray for reference.

C Infrastructure Implementation Details

C.1 Reproducibility and Supplementary Materials

All experiments use publicly available data sources. The supplementary ZIP archive attached to the OpenReview submission serves as a detailed reference for our implementation and experimental setup. It includes source code, training and evaluation configurations, data-construction manifests, checkpoint-conversion utilities, detailed evaluation outputs, and figure-generation scripts.

Training builds on NVIDIA Megatron Core v0.17.1 (Shoeybi et al., 2019) with the additional LoopLA implementation, and linear-attention kernels build on Flash Linear Attention (Yang & Zhang, 2024). The data and tokenizer come from the public Dolma 3 and Qwen3.5 releases (Allen Institute for AI, 2025; Qwen Team, 2026). Our software environment uses the NVIDIA PyTorch 25.11 container, `nvcr.io/nvidia/pytorch:25.11-py3` (NVIDIA Corporation, 2025). Table 18 records the pinned software versions and data revisions.

Table 18: Pinned training software versions and data revisions used in our experiments.

Aspect	Component	Pinned version or revision
Data	Qwen tokenizer	`dc7cdfe2ee4154fa7e30f5b51ca41bfa40174e68`
Data	Dolma 3	`afa92bfb22366821c5e6cd427cdd036b34b713ef`
Training	Container image	`nvcr.io/nvidia/pytorch:25.11-py3`
Training	Transformer Engine	`2.9.0`
Training	fla-core	`0.5.2`
Training	Megatron-LM baseline	`core_v0.17.1`; `266f1c97dca477ca8b92c16087506da2000b0b84`

C.2 Jacobi-Chunk Algorithm

Algorithm 1 describes the stack schedule used by the current implementation. Each pass traverses the entire shared recurrent core in layer order, including attention and FFN. Let S_ℓ and C_ℓ denote the committed matrix state and convolutional tail of recurrent layer ℓ. At each chunk boundary, we retain fixed copies S_ℓ^{in} and C_ℓ^{in} for all passes. The main experiments use $K = 2$; the same procedure supports the $K = 1, \ldots, 4$ implementation used in the efficiency sweep.

For each layer, $\mathrm{Project}_\ell$ applies input normalization, the shared projections, causal Q/K/V convolution, and the native activations and Q/K normalization. It returns queries Q, keys k, values v, write strengths β, decay factors α, output gates G, and a candidate convolutional tail C'. All passes use the same incoming tail, while the projected sequence inside the chunk depends on the current pass. Finish_ℓ applies the gated output normalization and projection, attention residual, and the normalized FFN with its residual.

$\mathrm{ScanRead}$ applies the affine updates in eq. (4) by a parallel prefix scan over E. At stage r, the event order is $(a, p_1), \ldots, (a, p_r), (a+1, p_1), \ldots, (b, p_r)$. Each query reads the state after its token's last writer p_r, and S' is the state after (b, p_r). The first stage uses the gated-delta kernel; later stages use the delta-product kernel with the retained writers interleaved in this order. Each scan restarts from S_ℓ^{in}, so earlier writers are replayed without applying their updates twice to the incoming state. The implementation applies decay only once per token in this consolidated recurrence and commits only the final pass's convolutional tail.

Intermediate scans produce draft hidden trajectories, but their terminal states are not carried into the next chunk. The approximation is in the hidden-conditioned writer proposals; the scan is exact for the fixed proposals. The pseudocode shows training and full-chunk prefill. Incremental decoding preserves the corresponding draft caches within a chunk and resets them at chunk rollover, as described below. Prelude and coda use ordinary causal layer execution, and their internal states are omitted here.

Algorithm 1 LOOPLA with Jacobi chunks and a shared recurrent core

Input: hidden sequence X, chunk size B, passes K, ordered recurrent layers $\mathcal{L}$
$H \leftarrow \mathrm{Prelude}(X)$; initialize $S_\ell \leftarrow 0$, $C_\ell \leftarrow \varnothing$ for $\ell \in \mathcal{L}$
for each contiguous chunk $J = [a, b]$ of H, in sequence order **do**
  $h \leftarrow H_J$
  for $\ell \in \mathcal{L}$ **do**
    $(S_\ell^{\mathrm{in}}, C_\ell^{\mathrm{in}}) \leftarrow (S_\ell, C_\ell)$; $\mathcal{W}_\ell \leftarrow [\,]$
  end for
  for $r = 1, \ldots, K$ **do**
    for $\ell \in \mathcal{L}$, in layer order **do**
      $(Q, k, v, \beta, \alpha, G, C') \leftarrow \mathrm{Project}_\ell(h, C_\ell^{\mathrm{in}})$
      if $r > 1$ **then** $\alpha \leftarrow \mathbf{1}$
      $W_{\ell,r} \leftarrow (k, v, \beta, \alpha)$
      Append $W_{\ell,r}$ to $\mathcal{W}_\ell$; retain all earlier writers
      $E \leftarrow [(t, p_j) : t = a, \ldots, b;\ j = 1, \ldots, r]$ (token-major order)
      $(R, S') \leftarrow \mathrm{ScanRead}(S_\ell^{\mathrm{in}}, \mathcal{W}_\ell, E, Q)$
      $h \leftarrow \mathrm{Finish}_\ell(h, R, G)$
      if $r = K$ **then** $(S_\ell, C_\ell) \leftarrow (S', C')$
    end for
  end for
  $Y_J \leftarrow h$
end for
return $\mathrm{Coda}(\mathrm{Concat}_J\, Y_J)$

C.3 IMPLEMENTATION OPTIMIZATIONS AND EFFICIENCY MEASUREMENT

The efficiency results in section 4.3 and fig. 5 use the implementation described below. Our optimizations reduce kernel-launch overhead, intermediate tensor traffic, and repeated recurrent-state transfers while preserving the causal draft-and-consolidate procedure.

Parallel scans and CUDA Graphs. Training uses FLA's chunkwise gated-delta kernel for the first pass and its delta-product kernel to consolidate the writers generated through each subsequent pass. Token updates within a Jacobi chunk use parallel scans; chunks and passes retain their required sequential dependencies. We capture the complete training iteration, including the forward pass, vocabulary loss, backward pass, and optimizer update, with CUDA Graphs. Replay reduces Python dispatch and host-scheduling overhead across the repeated operations. Compilation and kernel autotuning finish during eager warmup, using the same FLA path as graph capture. Generated writer tensors are retained for later consolidation, although each consolidation still performs its required writer transitions and materializes their interleaved inputs.

Convolution and activation fusion. For a chunk with no incoming convolutional history, the supported causal-convolution path fuses SiLU into the convolution kernel. This removes a separate activation intermediate and its forward/backward kernel launches. Calls with an incoming convolutional tail use the stateful path described below; unsupported configurations retain the original implementation.

Regional compilation for small chunks. Smaller chunks repeatedly incur convolution-tail handling, normalization, gating, and tensor-conversion overhead. We apply `torch.compile` to two fixed-shape regions: the operations following the input projection and the output gate. The first region joins the incoming tail with the projected Q/K/V, performs depthwise convolution and SiLU, normalizes Q/K, computes the write strength and decay, and retains the outgoing tail. The second fuses casts and products around the output gate. Both regions preserve the original precision boundaries, including FP32 output-gate SiLU and the decay-gradient casting and reduction order. The implementation retains `F.conv1d` and explicit autograd boundaries for numerically sensitive operations. Regional compilation is initialized before the outer CUDA Graph capture, with compiler-managed CUDA Graphs disabled. In the reported sweep, this path is enabled for $B \in \{256, 512, 1024, 2048\}$; $B = 4096$ uses the native projection path.

Fused recurrent decoding. When the query length is one and a recurrent state is available, we use FLA's fused recurrent kernel, avoiding the chunkwise scan setup needed for parallel prefill. At each consolidation pass, all writers generated so far are packed into consecutive events in one recurrent-kernel call. The matrix state stays within that call across writer transitions, reducing separate launches and repeated state loads and stores. Decay is applied only at the first event, and the pass reads the final event's output. Passes still execute in order because each consumes the preceding hidden representation. Only the final consolidation pass commits its state; temporary draft caches support incremental execution and reset at chunk boundaries. Prefill retains the chunkwise kernel. The measured decoding path uses neither CUDA Graphs nor the training-only regional compiler.

Measurement protocol. The chunk-size and loop-count sweeps use one NVIDIA A100 80GB PCIe GPU, the software environment in appendix C.1, random model weights, and synthetic tokens. Training measures complete iterations at sequence length 4,096 and batch size one, without activation or vocabulary-loss recomputation. Two fresh processes run each configuration in reversed experiment orders. Each process runs ten iterations; only graph replays 4–10 are timed, and the reported latency averages the two process medians. Decoding measures the embedding and all model layers at batch size one, excluding the LM head and argmax. Each process uses ten warmups and 100 samples, with cache preparation outside timing. Ordinary-token and chunk-rollover latency are measured separately and amortized as $t_{\text{ordinary}} + (t_{\text{rollover}} - t_{\text{ordinary}})/B$. Decode bars average the two process means. Training error bars span all replay samples; decode error bars span the two process means.

D Causal Leakage Diagnostics

This appendix extends the discussion in section 3.1 and further explains why state carry in looped linear attention must remain causally safe. The experiment in fig. 3 uses a non-causal loop that first processes a sequence of tokens with a parallel linear-attention scan, then directly uses the resulting final state S_{T,p_1} to initialize the next loop pass:

$$S_{T,p_1} = F(x_{1:T}), \qquad h_t^{(2)} = G(x_{1:t}, S_{T,p_1})\,. \tag{8}$$

Although each scan is causal in isolation and can be evaluated efficiently in parallel, directly reusing its full-sequence final state breaks causality across loop passes: S_{T,p_1} contains information written by $x_{t+1:T}$. An early token in the next pass can therefore access later tokens, creating direct causal leakage. Detaching the carried state can block gradients through this shortcut, but it does not change the non-causal forward computation.

The LAMBADA audit and Basic QA results in this appendix follow a separate diagnostic protocol from the main experiments in appendix B.1. In particular, LAMBADA uses paired ordinary and strict-prefix scoring on 500 examples, while Basic QA replaces SciQ with zero-shot ARC-Challenge; these results should be compared within this diagnostic study.

Strict-prefix audit. We reevaluate the first 500 LAMBADA examples under ordinary and strict-prefix scoring. Ordinary scoring places the context and complete continuation in one forward pass and reads all continuation-token logits at once. For a causal model, this is equivalent to the autoregressive factorization $p(y_{1:m} \mid x) = \prod_{j=1}^{m} p(y_j \mid x, y_{<j})$. For the non-causal loop, however, the state carried from the full sequence allows the logit for y_j to depend on y_j and later continuation tokens, so the same batched procedure no longer measures genuine next-token prediction. Strict-prefix scoring instead runs the model separately for every continuation position j, using only $[x, y_{<j}]$ as input and reading the next-token probability assigned to y_j. The target and remaining suffix are absent from the forward pass. Both evaluations are zero-shot, use BF16 and batch size 1, and score the same examples. Under ordinary scoring, the non-causal loop appears highly capable on language modeling, but its performance collapses under strict-prefix evaluation. In contrast, the no-loop model gives the same result under both protocols. The apparent gain of the non-causal loop therefore comes from future-token access rather than improved causal prediction.

As a broader transfer diagnostic, we also report the unweighted macro average over HellaSwag, WinoGrande, BoolQ, OpenBookQA, ARC-Challenge, ARC-Easy, and PIQA, covering 20,465 zero-shot examples. We use normalized accuracy where defined by the benchmark and native accuracy for BoolQ and WinoGrande. This Basic QA aggregate measures whether an apparent language-modeling

Variant	LAMBADA	Strict prefix	Gap (pp)	Basic QA
Dense/no-loop	27.8%	27.8%	0.0	44.8%
Non-causal loop	39.6%	2.0%	37.6	37.4%
Gradient blocking				
Full-state carry + detach	55.8%	35.2%	20.6	38.9%
Capacity restriction				
Dual-Memory K8	29.0%	26.0%	3.0	45.2%
Dual-Memory K16	41.4%	32.0%	9.4	43.7%
Stochastic regularization				
Full-state carry + dropout 0.5	64.8%	33.2%	31.6	38.1%
K16 layers 16–24 + dropout 0.3	28.4%	27.0%	1.4	44.9%
Information bottleneck				
K16 VIB $\beta = 0.03$	26.2%	26.0%	0.2	45.2%
K16 VIB $\beta = 0.1$	26.8%	26.8%	0.0	45.7%
K16 VIB $\beta = 0.5$	27.8%	27.8%	0.0	45.5%

Table 19: Causal audit and broad QA results for the no-loop baseline, non-causal state reuse, and representative mitigations. All rows use exact iteration-6,000 checkpoints and the same evaluation contract. Gap is ordinary-minus-strict LAMBADA accuracy in percentage points. A smaller gap does not establish causal safety.

gain transfers to ordinary downstream tasks; it is not itself a strict-prefix causal audit. Nevertheless, we observe a clear performance degradation of the non-causal loop on this suite.

Mitigating leakage does not ensure causality. We also audit several historical attempts to make the leaked channel harder to exploit. These methods fall into four categories:

- **Gradient blocking.** The full-state detach variant carries the same final recurrent state into the next loop pass, but applies a stop-gradient operation at the pass boundary. Consequently, a later-pass loss cannot directly train the preceding pass to encode future tokens into the carried state. This removes the most direct optimization shortcut while leaving the forward state and its visibility unchanged.
- **Capacity restriction.** The Dual-Memory variants separate the ordinary fine-grained recurrent state from a smaller coarse memory used for cross-pass communication. Instead of exposing the full state to the next pass, they compress the carried information into $M = 8$ or $M = 16$ coarse slots. The historical experiment labels K8 and K16 denote these memory sizes; M counts slots, while K elsewhere counts loop passes. Comparing the two sizes tests whether reducing the bandwidth of this auxiliary channel makes future information more difficult to transmit.
- **Stochastic regularization.** The full-state dropout variant randomly masks the carried state with dropout probability 0.5 during training. A second, more restrictive design uses the K16 coarse memory only in layers 16–24 and applies dropout 0.3 to that channel. The first perturbs a high-capacity full-state path, whereas the second combines a narrower layer range with a noisy low-capacity path.
- **Information bottleneck.** The VIB variants place a learned variational bottleneck on the K16 coarse-memory channel. During training, the projected memory is sampled with noise and receives a β-weighted KL penalty that discourages it from carrying excessive information. We test $\beta \in \{0.03, 0.1, 0.5\}$ to determine whether progressively stronger information control can remove the leakage while retaining useful cross-pass memory.

All rows in table 19 use exact iteration-6,000 checkpoints from seed-42 runs, with sequence length 4,096 and global batch size 1,024. Each model has therefore processed the same 25.17B tokens. Within every row, ordinary and strict-prefix LAMBADA use the same checkpoint and the same 500 examples under the protocol above.

As visualized in fig. 7, gradient detachment blocks the strongest optimization shortcut, but LAMBADA still drops by 20.6 points under strict scoring because the forward state continues to contain

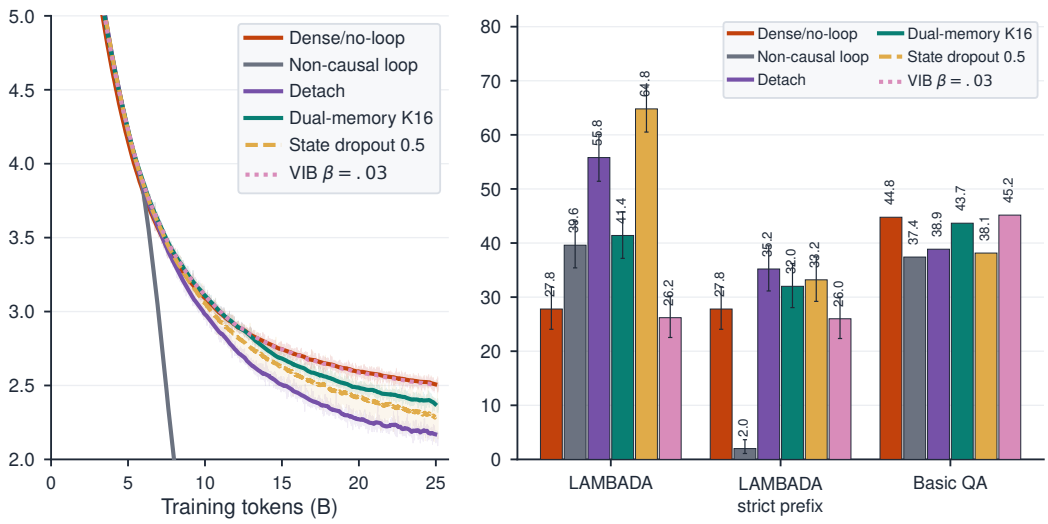

Figure 7: Complementary view of representative leakage-mitigation experiments. **Left:** training loss as a function of processed tokens; the non-causal loop rapidly leaves the displayed range as its loss collapses toward zero. **Right:** ordinary LAMBADA, strict-prefix LAMBADA, and Basic QA results at iteration 6,000. Mitigations can reduce the ordinary–strict gap, but either retain leakage or weaken useful cross-pass communication.

future information. Bottleneck size and dropout change the observed gap without changing the underlying visibility rule. Their effects are also non-monotonic: K8 drops by 3.0 points while K16 drops by 9.4, and full-state dropout 0.5 still drops by 31.6 points. A small gap can therefore mean only that a weak or narrow channel was not strongly exploited, not that the channel is causal.

VIB provides a particularly clear negative result. All three tested coefficients nearly or exactly close the ordinary–strict gap, but ordinary LAMBADA simultaneously falls to the Dense/no-loop range. Independent training diagnostics show that these variants subsequently converge to a degenerate zero-KL solution; for $\beta = 0.03$, the down-projection norm reaches 0.014 at iteration 19,000, compared with 9.71 for its matched no-VIB control. VIB thus suppresses leakage by effectively deleting cross-pass communication rather than preserving a useful memory and making it causal.

Structural causal safety. These results motivate treating causality as an architectural invariant rather than an empirical effect of regularization. As positive controls, the no-loop model, a two-pass model without state carry, and causally safe boundary-memory variants produce identical ordinary and strict-prefix results within each checkpoint. The state used at token t must be computed only from $x_{\leq t}$, as enforced by the token–pass ordering in eq. (5). Regularization may reduce observed leakage at one model scale or training stage, but only this prefix-safety constraint removes the invalid path by construction. This is why LOOPLA uses causally aligned state refinement instead of reusing a full-sequence state across loop passes.