LT2: LINEAR-TIME LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers (LT) reuse the same layers several times before producing each token, gaining computational depth without adding parameters. However, this *parameter efficiency* does not simply translate to *computational efficiency*. When paired with full quadratic attention, the quadratic cost is incurred at every loop, making both training and inference slow. We thus study **LT2 (Linear-Time Looped Transformers)**, a family of looped architectures that swap quadratic softmax attention for linear-time attention, enabling fast looping over a fixed, non-growing recurrent memory. We first show, both in theory and in practice, that looping has a strong synergistic effect with linear attention: on recall—linear attention's known weakness—looping helps linear attention more than it helps full attention, improving linear attention's natural-text recall by +8.9 points compared with +4.8 for full attention. We then propose **historical KV replay**, a simple and effective parameter-free mechanism that co-designs *recurrence in depth* (looping) and *recurrence in time* (linear attention): at every loop, each token replays its historical key–value pairs from all earlier loops into the current loop's state in reverse order, so that each loop's update draws not only on the current loop but also on historical information. In vanilla looped gated deltanet (GDN), strengthening it with historical KV replay significantly improves natural-text recall (34.6 to 41.1) and needle-retrieval extrapolation capacity, and raises every downstream benchmark, by ∼2.7 points on average. Furthermore, we propose a cheaper variant based on insights from KV replay, which replays only the first loop's key–value pair instead of the full history. It runs at the cost of plain looping yet recovers about 70% of the full replay's gain, serving as the most efficient yet powerful variant in practice. Finally, we provide a simple recipe to convert a pre-trained looped Transformer into a hybrid LT2. With only ∼1.3B tokens of continued training, our *Ouro-LT2-1.4B* retains most of its teacher's quality, letting practitioners reuse existing checkpoints rather than training from scratch.[1]

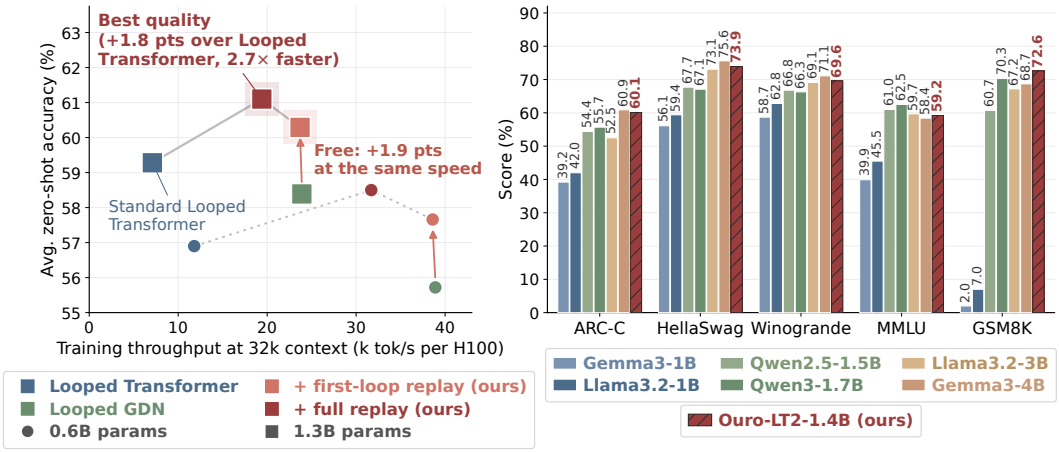


Figure 1: (**Left**) LT2 with historical KV replay sets a new quality–speed frontier for looped models: at 0.6B and 1.3B it is more accurate than the Looped Transformer, and at 0.6B it trains 2.7–3.3× faster at 32k context. (**Right**) Ouro-LT2-1.4B, a hybrid converted from Ouro-1.4B that keeps full attention in 6 of 24 layers, next to open models of similar and larger size.

[1]Code is available at https://anonymous.4open.science/r/LT2-326C/README.md.

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.

1 INTRODUCTION

Scaling neural language models along the parameter axis has driven much of modern NLP's progress (Brown et al., 2020; Kaplan et al., 2020; Hoffmann et al., 2022). A complementary axis—scaling depth through weight-shared recurrence—has recently emerged as a promising alternative. These architectures, called looped transformers (LT, originally *Universal Transformers* (Dehghani et al., 2019)), run the same layers for several steps before decoding each token (Giannou et al., 2023; Zhu et al., 2025), making looped transformers an appealing route to parameter-efficient reasoning.

However, the parameter efficiency of looped transformers does not simply imply computational efficiency. When paired with full attention, every loop re-applies the quadratic token-mixing operation over the entire sequence, so the attention FLOPs at training time and the KV cache at inference time both grow with sequence length and multiply with the number of loops, making additional loops at long context quickly become impractical (Tay et al., 2023; Zhu et al., 2025).

We study **LT2 (Linear-Time looped transformers)**, a family of looped transformers whose attention runs in linear time: each loop uses linear attention with a fixed recurrent memory instead of a growing KV cache. Here we focus on the diagonal-plus-low-rank (DPLR) family of models like GDN (Yang et al., 2025). We show, theoretically, that looping can *turn compute into context*: it gives linear attention finer control over its recurrent memory, upgrading the memory update from rank-1 in DPLR models to rank-T at most, where T is the loop count. Empirically, four loops raise GDN's natural-text recall by nearly twice the amount that full attention gains. However, in the vanilla LT2 family, the two forms of recurrence (*in depth* for looping and *in time* for linear attention) are designed separately, motivating us to co-design them to realize their full potential.

We then propose **historical KV replay**, a simple yet effective method that replays the historical KV pairs from each loop. At every loop, each token writes the key–value pairs from all its earlier loops into the current loop's state in reverse order. Early depth thus acts as a shared KV memory for all later loops, similar in spirit to YOCO (Sun et al., 2024). On top of looped GDN, full replay raises natural-text recall from 34.6 to 41.1, improves needle retrieval beyond the 4k training length, and lifts every downstream benchmark by $\sim$2.7 points on average (Figure 1, left). We ablated the design and found that the replay order matters a lot: *reverse replay* (writing the newest to oldest KV per loop) significantly outperforms *forward replay* (writing the oldest to newest KV per loop) in recall, needle retrieval, and language modeling tasks. This confirms that early-loop KV pairs carry the most informative representations, so writing their KV pairs last—giving them the highest effective priority—yields the strongest signal. Based on this observation, we further propose a cheaper form, **first-loop replay**, which writes only the first loop's KV pair: it runs at the cost of plain looping and recovers about 70% of the full replay's gain. With the depth-level chunkwise kernel, full replay incurs only a modest slowdown in training and prefill while maintaining nearly the same decoding speed as vanilla looped GDN, and first-loop replay achieves its gain for free, matching the speed of plain looping in both training/prefill and decoding.

Finally, we convert a pre-trained looped Transformer, Ouro (Zhu et al., 2025), into a hybrid LT2 that keeps full attention in a quarter of its layers. With only $\sim$1.3B tokens of continued training, our *Ouro-LT2-1.4B*, whose GDN layers use historical KV replay, retains most of the quality of its full-attention teacher (Figure 1, right). This gives practitioners a straightforward recipe to convert existing looped Transformer checkpoints into LT2 without retraining from scratch.

2 LT2: LINEAR-TIME LOOPED TRANSFORMER

2.1 ARCHITECTURE

Looped Transformer (LT). Let L be the sequence length and d the hidden dimension. We write the hidden-state sequence as $\mathbf{h} \in \mathbb{R}^{L\times d}$ and the state at position t as $\mathbf{h}_t \in \mathbb{R}^d$. A standard transformer of depth N stacks N blocks $\{\mathcal{F}_\ell\}_{\ell=1}^N$, each with its own parameters, consisting of an attention sub-layer and a feed-forward network (FFN) with residual connections:

$$\mathcal{F}_\ell(\mathbf{h}) = \mathbf{h}' + \mathrm{FFN}_\ell(\mathbf{h}'), \qquad \mathbf{h}' = \mathbf{h} + \mathrm{MHA}_\ell(\mathbf{h}), \tag{1}$$

where MHA_ℓ is multi-head self-attention (pre-norm omitted for brevity). A *Looped transformer* (LT) reuses the same N blocks for T iterations:

$$\mathbf{h}^{(0)} = \mathrm{Emb}(\mathbf{x}), \quad \mathbf{h}^{(\tau)} = \big(\mathcal{F}_N \circ \cdots \circ \mathcal{F}_1\big)\big(\mathbf{h}^{(\tau-1)}\big), \quad \tau = 1, \ldots, T, \quad \hat{\mathbf{y}} = \mathrm{Dec}\big(\mathbf{h}^{(T)}\big), \tag{2}$$

where $\mathbf{x}$ is the input token sequence, Emb and Dec are the embedding layer and output head, and $\hat{\mathbf{y}}$ denotes the next-token logits. This gives an effective depth of $T \cdot N$ with only N unique parameter sets—a $T\times$ parameter saving over a transformer of the same depth. Following Ouro, we fix T throughout pre-training. Adaptive computation time is discussed in Appendix A.

Vanilla LT2. Vanilla LT2 replaces the MHA sub-layer with a subquadratic attention variant, $\mathbf{h}' = \mathbf{h} + \mathrm{LinearAttn}_\ell(\mathbf{h})$. Every looped model (LT included) has a zero-initialized, per-channel learned gate $\boldsymbol{\rho}_\tau \in \mathbb{R}^d$ as a residual across loop iterations, $\mathbf{h}^{(\tau)} = \widetilde{\mathbf{h}}^{(\tau)} + \boldsymbol{\rho}_\tau \odot \mathbf{h}^{(\tau-1)}$, where $\widetilde{\mathbf{h}}^{(\tau)}$ is the output of the block stack at iteration τ. We refer to looped linear-attention models such as looped GDN as the vanilla LT2 family, and to their variants as LT2 with historical KV replay (Section 2.3).

2.2 Beyond Efficiency: Benefits of Looping

In this section, we first show the joint effect of depth recurrence and time recurrence. As shown in Figure 2, we find that looping benefits linear attention more than full attention: for 1.3B models, four loops raise GDN's natural-text recall by $+8.87$ points versus $+4.75$ for full attention, closing GDN's initial 3.9-point gap. Below we show what looping can add in theory: the T loops hand each token T distinct keys, which can turn the rank-1 state update of a diagonal-plus-low-rank (DPLR) linear-attention block into a rank-T one at most.

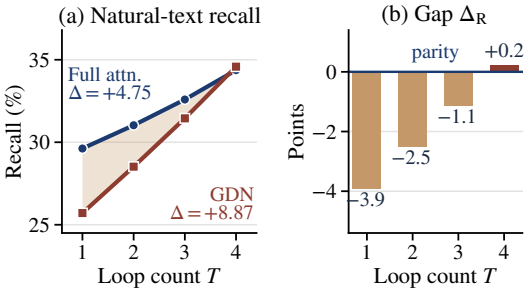


Figure 2: **Looping closes GDN's recall gap.** Natural-text recall at 2k (1.3B) vs. loop count. Panel (b) plots the gap shaded in (a).

Loop $\times$ DPLR linear attention: rank-T update on recurrent memory. State-of-the-art linear attention uses DPLR mixers, e.g. GDN and KDA (Kimi Team et al., 2025). Each head maintains a recurrent state $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$ (key and value head dimensions d_k, d_v), updated at position t via

$$\mathbf{S}_t = \mathbf{A}_t\,\mathbf{S}_{t-1} + \beta_t\,\mathbf{k}_t\mathbf{v}_t^\top, \qquad \mathbf{A}_t = \big(\mathbf{I} - \beta_t\,\mathbf{k}_t\mathbf{k}_t^\top\big)\mathrm{Diag}(\boldsymbol{\alpha}_t), \qquad \mathbf{o}_t = \mathbf{S}_t^\top \mathbf{q}_t, \tag{3}$$

where $\mathbf{q}_t$, $\mathbf{k}_t$, $\mathbf{v}_t$, and $\mathbf{o}_t$ are token t's query, key, value, and output, $\beta_t \in (0, 2)$ is the write strength, and $\boldsymbol{\alpha}_t \in [0,1]^{d_k}$ is the decay, channel-wise in KDA and a scalar in GDN, so $\mathbf{A}_t$ is a rank-1 perturbation of $\mathrm{Diag}(\boldsymbol{\alpha}_t)$. When the *same* block is looped T times, loop $\tau \in \{1, \ldots, T\}$ computes its own key $\mathbf{k}_t^{(\tau)}$ and write factor $\mathbf{G}_t^{(\tau)} = \mathbf{I} - \beta_t^{(\tau)}\mathbf{k}_t^{(\tau)}\mathbf{k}_t^{(\tau)\top}$. On one shared state, they multiply into

$$\mathbf{P}_t = \mathbf{G}_t^{(1)}\mathbf{G}_t^{(2)}\cdots\mathbf{G}_t^{(T)} = \mathbf{I} + \mathbf{K}_t\mathbf{C}_t\mathbf{K}_t^\top, \qquad \mathbf{K}_t = \big[\mathbf{k}_t^{(1)}, \ldots, \mathbf{k}_t^{(T)}\big], \tag{4}$$

with $\mathbf{C}_t$ upper triangular: $\mathbf{P}_t - \mathbf{I}$ has rank at most T, exactly T for linearly independent keys, and $\mathbf{A}_t$ is a rank-T perturbation of a diagonal matrix at most. *DeltaProduct* (Siems et al., 2025) shows that this rank-T form is strictly more expressive: a single recurrent layer can represent any norm-bounded orthogonal transformation of the state and solve any group word problem. We extend the proof in Appendix B.1.

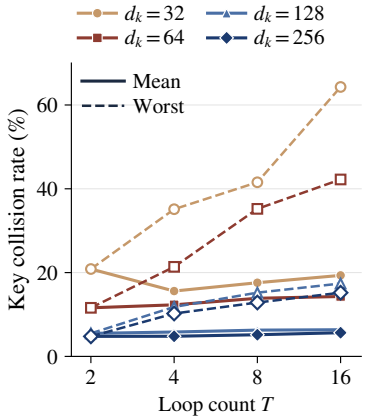


Figure 3: Key collision rate $|\cos(\mathbf{k}^{(\tau+1)}, \mathbf{k}^{(\tau)})|$ across loops (%).

From rank-1 to rank-T: the role of key geometry. When we say $\mathbf{P}_t - \mathbf{I}$ has rank *at most* T, we mean that the actual perturbation rank is decided by whether the keys at different loops modify the recurrent memory in different directions or in the same direction. We provide two extremes: If all keys are *identical*, $\mathbf{P}_t$ collapses to $\mathbf{I} - \beta^*\mathbf{k}\mathbf{k}^\top$ for some scalar β^*, and the rank stays at 1. If the keys are mutually *orthogonal*, the factors commute into a rank-T perturbation. Generally, in high dimensions, random vectors are nearly orthogonal (Blum et al., 2020). In practice, however, head dimensions are modest (d_k is typically 64–256), so this intuition needs to be checked empirically. Figure 3 shows that GDN keys from different loops are far from collinear: for

$d_k \geq 128$, the mean key collision rate between consecutive loops stays around 5%, and even the worst case stays below 20% at $T{=}16$. Modern linear-attention models such as GDN and KDA already use $d_k{=}128$ by default, so theory works with their standard configurations without specific modification.

2.3 Improving LT2 with Historical KV Replay

In vanilla LT2, each loop only writes its own key–value pair into its own recurrent memory, so in this view, *recurrence in depth* (loop) and *recurrence in time* (linear attention) are separate and independent. Historical KV replay changes this. We first give its general formulation and then a concrete example. At loop τ, token t first decays the loop's state, $\bar{\mathbf{S}}_{t-1}^{(\tau)} = \alpha_t^{(\tau)} \mathbf{S}_{t-1}^{(\tau)}$, then writes the pairs from all its loops so far, from the current loop back to the first, and reads the result with its own query:

$$\begin{aligned} \mathbf{S}_t^{(\tau)} &= \mathcal{W}_t^{(1)} \circ \mathcal{W}_t^{(2)} \circ \cdots \circ \mathcal{W}_t^{(\tau)} \big(\bar{\mathbf{S}}_{t-1}^{(\tau)}\big), \qquad \mathbf{o}_t^{(\tau)} = \mathbf{S}_t^{(\tau)\top} \mathbf{q}_t^{(\tau)}, \\ \mathcal{W}_t^{(j)}(\mathbf{S}) &= \big(\mathbf{I} - \beta_t^{(j)} \mathbf{k}_t^{(j)} \mathbf{k}_t^{(j)\top}\big)\mathbf{S} + \beta_t^{(j)}\, \mathbf{k}_t^{(j)} \mathbf{v}_t^{(j)\top}. \end{aligned} \tag{5}$$

Here $\mathbf{S}_t^{(\tau)}$ is the state of loop τ after token t, and we use $\mathcal{W}_t^{(j)}$ to denote one delta-rule key–value write, which stores token t's loop-j pair $(\mathbf{k}_t^{(j)}, \mathbf{v}_t^{(j)})$ in the state. The composition $\circ$ applies $\mathcal{W}_t^{(\tau)}$ first and $\mathcal{W}_t^{(1)}$ last. After the replayed writes, the current loop's query $\mathbf{q}_t^{(\tau)}$ reads out $\mathbf{o}_t^{(\tau)}$. Concretely, loop 1 writes its own pair. Loop 2 first writes its own pair and then loop 1's. Loop 3 writes its own pair, then loop 2's, then loop 1's. With $T{=}4$:

$$\begin{aligned} \text{loop 1:} \quad & \mathbf{S}_t^{(1)} = \mathcal{W}_t^{(1)} \big(\bar{\mathbf{S}}_{t-1}^{(1)}\big), \\ \text{loop 2:} \quad & \mathbf{S}_t^{(2)} = \mathcal{W}_t^{(1)} \circ \mathcal{W}_t^{(2)} \big(\bar{\mathbf{S}}_{t-1}^{(2)}\big), \\ \text{loop 3:} \quad & \mathbf{S}_t^{(3)} = \mathcal{W}_t^{(1)} \circ \mathcal{W}_t^{(2)} \circ \mathcal{W}_t^{(3)} \big(\bar{\mathbf{S}}_{t-1}^{(3)}\big), \\ \text{loop 4:} \quad & \mathbf{S}_t^{(4)} = \mathcal{W}_t^{(1)} \circ \mathcal{W}_t^{(2)} \circ \mathcal{W}_t^{(3)} \circ \mathcal{W}_t^{(4)} \big(\bar{\mathbf{S}}_{t-1}^{(4)}\big). \end{aligned} \tag{6}$$

So here with historical replay, loop 4 replays its history: the key–value pairs of loops 1, 2, and 3 are also written into loop 4's recurrent memory, which forms the depth and time recurrence. Finally, loop 4 reads out this memory, which now holds the pairs of all four loops, with its own query:

$$\mathbf{o}_t^{(4)} = \mathbf{S}_t^{(4)\top} \mathbf{q}_t^{(4)} = \Big(\mathcal{W}_t^{(1)} \circ \mathcal{W}_t^{(2)} \circ \mathcal{W}_t^{(3)} \circ \mathcal{W}_t^{(4)} \big(\bar{\mathbf{S}}_{t-1}^{(4)}\big) \Big)^{\top} \mathbf{q}_t^{(4)}. \tag{7}$$

The order of these writes matters, because it builds an inductive bias into the design (Appendix E). We view the loops as an unrolling of effective depth. Forward replay (loop 1 to the current loop) lets the past fade: each earlier pair is partly overwritten by later ones, so the state leans toward the newest loops. Our reverse order instead treats early-loop information as more important, with later loops mainly refining it. This is close to YOCO, which reuses KV cache from early layers in later layers.

In practice, we find that the values can also be shared without loss, as the rank-τ update comes only from the keys. Every write stores the loop-1 value $\mathbf{v}_t^{(1)}$, which saves compute.

First-loop replay. Our second method keeps only the last write of reverse replay, the loop-1 pair:

$$\mathbf{S}_t^{(\tau)} = \alpha_t^{(\tau)} \big(\mathbf{I} - \beta_t^{(\tau)} \mathbf{k}_t^{(1)} \mathbf{k}_t^{(1)\top}\big)\, \mathbf{S}_{t-1}^{(\tau)} + \beta_t^{(\tau)}\, \mathbf{k}_t^{(1)} \mathbf{v}_t^{(1)\top}, \qquad \mathbf{o}_t^{(\tau)} = \mathbf{S}_t^{(\tau)\top} \mathbf{q}_t^{(\tau)}, \tag{8}$$

where only the key and value are reused: every loop writes the first loop's pair $(\mathbf{k}_t^{(1)}, \mathbf{v}_t^{(1)})$ in place of its own, while the decay $\alpha_t^{(\tau)}$, write strength $\beta_t^{(\tau)}$, and query $\mathbf{q}_t^{(\tau)}$ remain per loop. It keeps the first loop's view of each token and recovers about 70% of the gain of full replay (Section 3.2 and Table 4), which serves as a more practical and efficient default.

Hardware-efficient replay via depth-level chunks. Full replay makes τ delta-rule writes at loop τ (10 instead of 4 per token at $T{=}4$, Table 1), but they need not update the state one by one. Their write

Table 1: **What each loop writes at position** t (T=4). $(\mathbf{k}, \mathbf{v})^{(j)}$: the token's pair from loop j, written left to right. Training throughput at 4k and decoding speed at 32k: change vs. Looped GDN at 0.6B (Table 5); theoretical costs in Appendix D.3.

Loop	**Looped GDN**	**First-loop replay**	**Full replay** (current → loop 1)
τ=1	$(\mathbf{k}, \mathbf{v})^{(1)}$	$(\mathbf{k}, \mathbf{v})^{(1)}$	$(\mathbf{k}, \mathbf{v})^{(1)}$
τ=2	$(\mathbf{k}, \mathbf{v})^{(2)}$	$(\mathbf{k}, \mathbf{v})^{(1)}$	$(\mathbf{k}, \mathbf{v})^{(2)} \rightarrow (\mathbf{k}, \mathbf{v})^{(1)}$
τ=3	$(\mathbf{k}, \mathbf{v})^{(3)}$	$(\mathbf{k}, \mathbf{v})^{(1)}$	$(\mathbf{k}, \mathbf{v})^{(3)} \rightarrow (\mathbf{k}, \mathbf{v})^{(2)} \rightarrow (\mathbf{k}, \mathbf{v})^{(1)}$
τ=4	$(\mathbf{k}, \mathbf{v})^{(4)}$	$(\mathbf{k}, \mathbf{v})^{(1)}$	$(\mathbf{k}, \mathbf{v})^{(4)} \rightarrow (\mathbf{k}, \mathbf{v})^{(3)} \rightarrow (\mathbf{k}, \mathbf{v})^{(2)} \rightarrow (\mathbf{k}, \mathbf{v})^{(1)}$
Query	$\mathbf{q}^{(\tau)}$	$\mathbf{q}^{(\tau)}$	$\mathbf{q}^{(\tau)}$
Decay, strength	$\alpha^{(\tau)}$, $\beta^{(\tau)}$	$\alpha^{(\tau)}$, $\beta^{(\tau)}$	$\alpha^{(\tau)}$ once, then each pair's own $\beta^{(j)}$
Writes per token	4	4	10
Training throughput	41.2k tok/s	same ✓ (−1%)	lower (−16%)
Decoding speed	19.4 tok/s	faster ✓ (+4%)	nearly the same ✓ (−5%)

factors multiply into $\mathbf{I} + \mathbf{KCK}^\top$ as in Eq. (4), so the state receives one update of rank at most τ (Eq. (21)). This is the WY form that chunkwise GDN kernels use across tokens (Yang et al., 2025), here across loops. We call it a *depth-level chunk* (Appendix D.2). Memory-bound decoding thus reads and writes each loop's state once, as in Looped GDN, and only the flops grow. In training and prefill, the replayed pairs add rows to the time-level chunks (Appendix D.1), where the extra cost lies (Table 7). At 0.6B (Table 5), full replay trains 16% slower at 4k but decodes only 5% slower at 32k (18.4 vs. 19.4 tokens/s), while first-loop replay adds no writes and is essentially free (Section 3.3).

3 Experiments

We pre-train all models on FineWeb-Edu (Penedo et al., 2024) at 0.6B and 1.3B parameters for 100B tokens, with T=4 for every looped variant as in Ouro (Appendix C). Sections 3.1–3.2 compare the models, and Section 3.3 analyzes replay.

3.1 Recall within and beyond the training length

We evaluate the 1.3B models, trained at 4k, on natural-text recall at 2k (Table 2) and on RULER (Hsieh et al., 2024) needle retrieval from 4k to 32k (Figure 4). Looped GDN matches the Looped Transformer in average recall (34.58 vs. 34.37) but trails it by 11.4 points on FDA. Full replay raises the average to 41.10 and improves all six tasks, most on FDA (+8.4), DROP (+7.3), and SWDE (+7.2); first-loop replay reaches 39.14, while forward replay stays at 34.71. On RULER, the Looped Transformer is best at 4k (97.3 vs. 77.5 for full replay) but falls to 0.5 at 8k, whereas the linear-time models degrade gradually. At 32k, full replay scores 25.5, twice Looped GDN's 12.8.

Table 2: **Natural-text recall at 2k** (1.3B, T=4, trained at 4k). Cream = ours. Best in **bold**, second underlined. Forward/reverse: full replay in either order.

Model	SWDE	SQuAD	FDA	TQA	NQ	DROP	Avg.↑
Looped Transformer	36.28	34.15	**43.87**	52.94	21.86	17.09	34.37
▶ *LT2-linear attention*							
Looped Mamba2	31.86	32.41	24.73	56.92	22.58	23.14	31.94
Looped GDN	35.42	35.17	32.48	57.83	24.19	22.36	34.58
▶ *LT2-GDN w/ historical KV replay (ours)*							
forward replay	35.63	34.92	32.97	58.04	24.33	22.37	34.71
first-loop replay	40.71	39.52	38.36	60.93	28.14	27.18	39.14
reverse replay	**42.63**	**41.28**	40.92	**62.41**	**29.66**	**29.70**	**41.10**

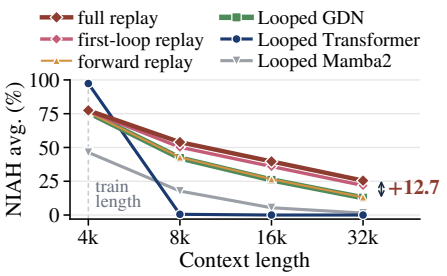


Figure 4: **Needle retrieval vs. context length** (RULER NIAH avg., 1.3B, trained at 4k, Appendix G).

Table 3: **Zero-shot downstream performance on FineWeb-Edu at two scales (T=4).** Cream = ours: full replay in reverse order and first-loop replay. Forward replay is the control. Best looped model per scale in **bold**, second best underlined. Avg. subscripts: std. over three seeds. Speed: decoding tokens/s at 32k, batch size 1, same inference code for all models (×: vs. Looped Transformer).

Model	PPL(↓)	ARC-E	ARC-C	HellaS.	PIQA	WG	OBQA	SciQ	BoolQ	Avg.(↑)	Speed↑ tok/s
0.6B parameters / 100B tokens (8× Chinchilla ratio)											
Transformer	13.14	63.09	30.72	47.43	69.53	56.24	35.6	68.2	50.07	52.61$_{\pm 0.26}$	31.4
Looped Transformer	11.92	67.13	34.67	53.29	70.58	62.83	38.2	73.6	54.87	56.90$_{\pm 0.22}$	6.3 ×1.0
▶ *Window attention*											
Looped Window	12.87	64.23	31.53	48.83	69.82	57.34	35.8	68.5	51.23	53.41$_{\pm 0.24}$	16.4 ×2.6
▶ *LT2-linear attention*											
Looped RetNet	—	*training diverged*									**24.8** ×3.9
Looped Mamba2	12.78	64.53	31.82	49.87	69.74	58.63	35.6	68.8	51.83	53.85$_{\pm 0.27}$	23.9 ×3.8
Looped HGRN2	14.59	59.82	27.93	43.17	67.34	52.13	33.4	65.2	48.53	49.69$_{\pm 0.31}$	22.6 ×3.6
Looped DeltaNet	14.16	60.47	28.53	44.22	67.87	53.24	33.8	65.5	49.13	50.35$_{\pm 0.29}$	21.3 ×3.4
Looped GDN	12.06	66.43	33.89	52.62	70.27	61.48	36.4	70.5	54.13	55.72$_{\pm 0.21}$	19.4 ×3.1
Looped KDA	12.13	66.12	33.63	52.37	70.13	61.22	36.2	70.2	53.92	55.47$_{\pm 0.23}$	18.7 ×3.0
▶ *LT2-GDN w/ historical KV replay (ours)*											
forward replay	12.03	66.71	33.62	52.84	70.41	61.12	36.8	70.3	54.46	55.78$_{\pm 0.24}$	18.4 ×2.9
first-loop replay	11.64	68.61	35.95	54.58	71.70	63.71	37.8	72.8	56.14	57.66$_{\pm 0.19}$	20.1 ×3.2
reverse replay	**11.46**	**69.47**	**36.83**	**55.52**	**72.31**	**64.58**	**38.4**	**73.8**	**57.09**	**58.50**$_{\pm 0.18}$	18.4 ×2.9
1.3B parameters / 100B tokens (4× Chinchilla ratio)											
Transformer	10.65	67.52	33.84	52.47	71.03	61.48	36.6	71.3	54.02	56.03$_{\pm 0.19}$	18.6
Looped Transformer	9.87	70.83	37.54	57.06	72.43	65.83	38.6	74.1	57.83	59.28$_{\pm 0.17}$	3.8 ×1.0
▶ *Window attention*											
Looped Window	10.63	67.73	34.87	54.12	70.94	62.53	36.5	71.2	55.23	56.64$_{\pm 0.21}$	9.9 ×2.6
▶ *LT2-linear attention*											
Looped Mamba2	10.55	67.97	35.13	54.44	71.18	62.87	36.7	71.5	55.53	56.92$_{\pm 0.22}$	**14.6** ×3.8
Looped HGRN2	12.21	63.48	30.62	47.35	69.12	55.84	34.6	67.9	51.27	52.52$_{\pm 0.26}$	13.7 ×3.6
Looped DeltaNet	11.83	64.21	31.37	48.52	69.58	57.03	35.1	68.4	52.06	53.28$_{\pm 0.24}$	12.9 ×3.4
Looped GDN	10.02	69.78	36.83	56.23	71.87	64.76	37.6	72.8	57.28	58.39$_{\pm 0.16}$	11.9 ×3.1
Looped KDA	9.93	70.07	37.12	56.49	72.03	64.92	37.8	73.1	57.48	58.63$_{\pm 0.18}$	11.3 ×3.0
▶ *LT2-GDN w/ historical KV replay (ours)*											
forward replay	10.00	69.94	36.61	56.47	72.02	64.51	37.9	72.6	57.55	58.45$_{\pm 0.20}$	11.3 ×3.0
first-loop replay	9.61	72.07	38.99	58.36	73.52	66.93	39.2	73.8	59.46	60.29$_{\pm 0.15}$	12.3 ×3.2
reverse replay	**9.44**	**73.12**	**39.91**	**59.18**	**74.23**	**67.94**	**39.9**	**74.2**	**60.32**	**61.10**$_{\pm 0.14}$	11.3 ×3.0

3.2 Language Modeling

In Table 3, the DPLR mixers GDN and KDA are the strongest looped linear-attention models, yet Looped GDN still trails the Looped Transformer by 1.2 and 0.9 Avg. points at 0.6B and 1.3B. Full replay improves Looped GDN on all eight tasks, by 2.8 and 2.7 points on average, and overtakes the Looped Transformer by 1.6 and 1.8 points with lower perplexity, while decoding 2.9–3.0× faster at 32k. First-loop replay keeps 70% of this gain (57.66 and 60.29), still above the Looped Transformer, and decodes slightly faster than Looped GDN. Forward replay stays within 0.06 of Looped GDN.

3.3 Ablations

Which pairs to replay, and in what order. Table 4 varies the replayed pairs, their order, and the stored value at 0.6B. The number of writes does not explain the gain: forward replay makes the same 10 writes as full replay and gains 0.06 Avg., and replaying the previous loop gains 0.45. The gain comes from writing the loop-1 pair last: current + loop 1 reaches 84% of the full-replay gain with 7 writes, and first-loop replay 70% with no extra writes. Since the last write is the least overwritten, every loop's state then keeps the same early-depth view of each token, much as YOCO reuses an early-layer KV cache. Per-loop values instead of the shared $\mathbf{v}_t^{(1)}$ give 58.43, so sharing costs nothing.

Test-time interventions. Figure 5 changes only the replay of the trained 1.3B full-replay model at inference. Reversing the order, with the same pairs and writes, costs 7.7 recall points, 71% of the 10.9 lost without replay. Dropping the loop-1 pair costs 8.5, and dropping the previous loop's pair only 0.8. The trained model relies on the loop-1 pair being written last.

Table 4: **Which pairs, in which order, with which value** (0.6B, T=4). Pattern: writes at loop 4, by source loop (1→4). Writes: per token and layer. Rec.: recall at 2k.

Variant	Pattern	Writes	PPL↓	Avg.↑	Rec.↑
No replay (Looped GDN)	4	4	12.06	55.72	29.83
Forward replay	1 2 3 4	10	12.03	55.78	30.12
Current + previous loop	4 3	7	11.94	56.17	31.05
Current + loop 1	4 1	7	11.58	58.06	34.27
Full replay, own values	4 3 2 1	10	11.49	58.43	35.34
First-loop replay	1	4	11.64	57.66	33.89
Full replay	4 3 2 1	10	**11.46**	**58.50**	**35.61**

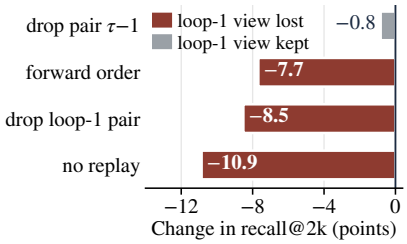


Figure 5: **Replay changed at test time** (1.3B, trained with full replay): change in recall at 2k. Values in Appendix F.

Hybrids and robustness. Replay acts only on GDN layers, so it applies unchanged to hybrid LT2 (Figure 6). Full replay adds +1.4 Avg. to the Full+GDN hybrid and +2.0 to the GDN+DSA hybrid, whose attention layers use the DeepSeek Sparse Attention (DSA) of DeepSeek-AI et al. (2025). First-loop replay adds +0.9 and +1.4. The gain is smaller with full-attention layers, consistent with full attention already covering part of the recall that replay adds. Even so, the replayed Full+GDN hybrid reaches 60.14 Avg. at 0.6B, 3.2 points above the Looped Transformer. Over three seeds, the standard deviation of Avg. is at most 0.31 for every pre-trained model, well below these gains.

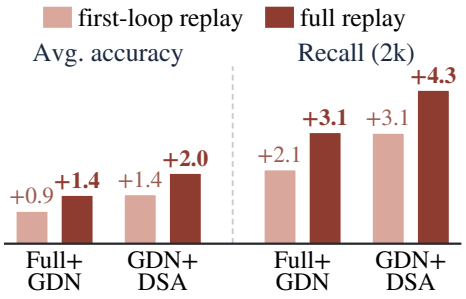


Figure 6: **Replay on hybrid LT2** (0.6B, T=4, GDN layers only): gain over no replay. Values in Appendix F.

Comparison at matched or higher compute. Table 5 reports measured costs at 0.6B (theory in Appendix D.3). Replay adds no parameters and keeps the 0.05 GB state of Looped GDN, against a 12.1 GB KV cache for the Looped Transformer at 32k. First-loop replay stays within 2% of Looped GDN in training and prefill and decodes 4% faster. Full replay trains 16% slower at 4k and prefills 28% slower, but decodes only 5% slower because its writes merge into one state update per loop (Section 2.3); at 32k it still trains 2.7× and decodes 2.9× faster than the Looped Transformer. The lower block of Table 5 and Figure 7 spend similar or more compute without replay: two more loops, a 0.78B model, or Gated DeltaProduct, whose extra keys come from additional projections rather than from earlier loops. The best of them, DeltaProduct with 12 writes per token, gains 1.01 Avg.; full replay gains 2.78 with 10 writes and trains faster (34.6 vs. 32.2k tokens/s), and two more loops cost 33% of training throughput for +0.56. Extra writes or a higher-rank update alone therefore do not account for the gain.

Table 5: **Cost of replay vs. spending compute in other ways** (0.6B, T=4). Training: k tokens/s per H100. At 32k: prefill (s), decoding tokens/s at batch 1 (■ ours), state/KV cache (GB). Subscripts: gain over Looped GDN. Bottom: similar compute, no replay. Best **bold**, second underlined. Theoretical costs are in Appendix D.3. Forward replay has exactly the cost of full replay and is omitted.

		Training (k tok/s)		Inference at 32k			
Model	Params	4k↑	32k↑	Prefill↓	Decode↑	Cache↓	Avg.↑
Looped Transformer	0.60B	39.6	11.8	2.47	6.3	12.1	56.90
Looped GDN	0.60B	**41.2**	**38.9**	**0.93**	19.4	0.05	55.72 ref.
w/ **first-loop replay**	0.60B	40.9	38.6	0.94	**20.1**	0.05	57.66 +1.94
w/ **full replay**	0.60B	34.6	31.7	1.19	18.4	0.05	**58.50** +2.78
GDN, T=6	0.60B	27.8	26.1	1.38	12.9	0.08	56.28 +0.56
GDN, wider	0.78B	32.1	30.4	1.16	14.6	0.06	56.52 +0.80
Gated DeltaProduct							
n_h=2 writes per loop	0.63B	36.4	34.1	1.08	15.3	0.05	56.41 +0.69
n_h=3 writes per loop	0.66B	32.2	29.8	1.26	13.2	0.05	56.73 +1.01

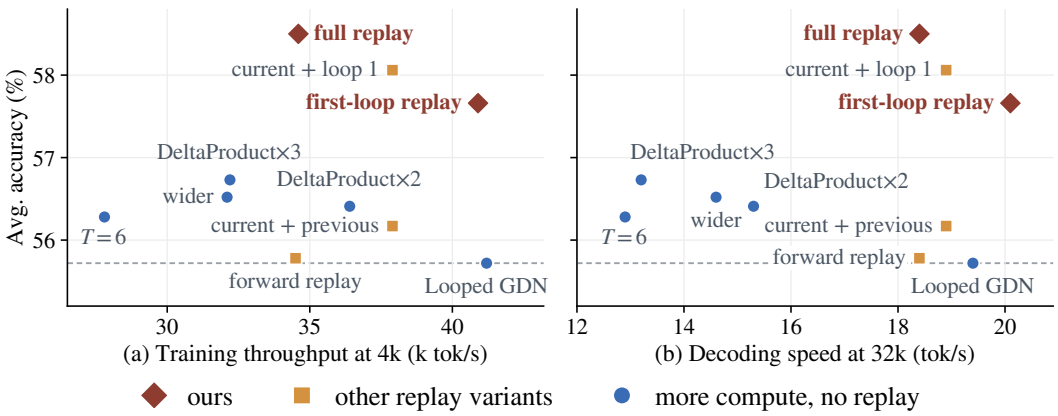


Figure 7: **Replay beats spending similar compute in other ways** (0.6B, $T{=}4$, DeltaProduct$\times n$: $n_h{=}n$), measured by **(a)** training speed at 4k and **(b)** decoding speed at 32k. Values in Appendix F.

4 Distilling Looped Transformers into Hybrid LT2

LT2 can also be obtained *post hoc* from a pre-trained Looped Transformer. Our teacher $f_{\theta_\mathcal{T}}$ is Ouro-1.4B. The conversion has two independent parts. The *architecture* of the student $f_{\theta_\mathcal{S}}$ is LT2-GDN: it shares the teacher's embedding, FFN, and norm parameters, keeps full attention in a set of layers $\mathcal{F} \subseteq \{1, \ldots, N\}$ (25%), and uses GDN elsewhere, optionally with first-loop or full replay, which adds no parameters. The *method* extends the RADLADS-style distillation of Li et al. (2025b) to loops in three stages. At each loop τ, both networks emit logits $z^{(\tau)}_{\mathcal{T}/\mathcal{S}}$.

Stage 1 (linear pre-alignment). With $\mathcal{F} = \varnothing$, we align each GDN block to the corresponding teacher attention output via an MSE loss on the residual stream $\mathbf{h}^{(\tau,\ell)}$ after the mixing block of layer ℓ at loop τ (100M tokens, length 512):

$$\mathcal{L}_{\text{align}} = \sum_{\tau=1}^{T} \sum_{\ell=1}^{N} \left\| \mathbf{h}^{(\tau,\ell)}_{\mathcal{S}} - \mathbf{h}^{(\tau,\ell)}_{\mathcal{T}} \right\|_2^2, \tag{9}$$

Stage 2 (hybrid logit distillation). We restore $|\mathcal{F}| = 6$ softmax layers chosen by a KL-guided selector and distill the teacher's logits. Looping adds a new knob, the *per-loop* KL schedule. We minimize

$$\mathcal{L}_{\text{KD}} = \sum_{\tau=1}^{T} w_t^{(\tau)} \, \text{KL}\Big(\sigma_{\text{top-}k}\big(z^{(\tau)}_{\mathcal{T}} / T_{\text{kd}}\big) \,\Big\|\, \sigma_{\text{top-}k}\big(z^{(\tau)}_{\mathcal{S}} / T_{\text{kd}}\big) \Big), \tag{10}$$

where $\sigma_{\text{top-}k}$ renormalizes the softmax over the teacher's top-k tokens and $\mathbf{w}_t \in \Delta^{T-1}$ controls how much supervision each loop receives at training step t. We use uniform weights $\mathbf{w}_t = \frac{1}{T}\mathbf{1}$ for the first half of training and supervise only the final output ($\mathbf{w}_t = \mathbf{e}_T$) afterwards (600M tokens, length 4096).

Stage 3 (long-context continuation). We continue Stage 2 on 35K OpenThoughts-v3 samples (Guha et al., 2025) with the same KD loss and a constant LR (600M tokens, length 32768).

Table 6 separates the two parts. Our method gains most where the others fail: with the same LT2-GDN student, Uniform reaches only 14.9 on GSM8K and 40.8 on RULER, and the previous state of the art 49.6 and 75.3, compared with 71.0 and 80.6 for ours. These rows compare full recipes. We did not ablate the data (DCLM (Li et al., 2025a), then reasoning data), length schedule, or per-loop KL on their own. RULER is run at 4k only, below the 32k length of Stage 3.

The architecture gain is independent of the method. Full replay adds +1.9 Avg. with our method, most of it on RULER (+4.3), and +1.6 with the previous one. First-loop replay adds +1.3 at no extra cost, again about 70% of the gain. Our best student (our method with full replay) is Ouro-LT2-1.4B. It keeps full attention in 6 of 24 layers, so it is a hybrid, not a purely linear-time model.

Table 6: **Converting Ouro-1.4B into LT2: method vs. architecture.** Every LT2-GDN student keeps 6 of 24 layers as full attention (Uniform: fixed interleaving, prev. SoTA: selector of Li et al. (2025b)). RULER: mean of 13 subtasks at 4k. Best student in **bold**, second underlined.

Method	Architecture	ARC-C	HellaS.	WG	MMLU	GSM8K	RULER	Avg.
Ouro-1.4B (teacher, Looped Transformer)		61.3	74.9	71.5	67.6	78.6	86.7	73.4
Uniform	LT2-GDN	56.9	70.2	66.8	37.7	14.9	40.8	47.9
Prev. SoTA	LT2-GDN	60.0	73.4	67.9	56.8	49.6	75.3	63.8
	w/ full replay	**60.9**	73.8	69.0	57.7	51.4	79.8	65.4
Ours	LT2-GDN	58.6	72.7	68.3	58.0	71.0	80.6	68.2
	w/ first-loop replay	59.6	73.5	69.1	58.8	72.0	83.8	69.5
	w/ full replay	60.1	**73.9**	**69.6**	**59.2**	**72.6**	**84.9**	**70.1**

5 Related Work

Looped Transformers and scalable recursion. Universal Transformers (Dehghani et al., 2019) tie layer weights and apply one block for a fixed or adaptive number of steps, a form of depth recurrence related to deep equilibrium models (Bai et al., 2019). Weight sharing can improve systematic generalization (Csordás et al., 2021) and is a viable design choice (Takase & Kiyono, 2023). Looped architectures are Turing-complete under mild assumptions (Pérez et al., 2019), can emulate iterative algorithms such as multi-step gradient descent (Giannou et al., 2023; Yang et al., 2024a; Gatmiry et al., 2024a;b; Fan et al., 2025), and motivate a "latent thought" view in which each pass refines an internal computation (Saunshi et al., 2025; Gong et al., 2026; Blayney et al., 2026; Chen et al., 2026a). This view has inspired recursive reasoning models (Wang et al., 2025; Jolicoeur-Martineau, 2025; Gao et al., 2025) and looped language models trained at scale, including Huginn (Geiping et al., 2025a) and Ouro (Zhu et al., 2025). The challenge is compute: looping multiplies FLOPs by T, and Universal Transformers can underperform standard Transformers at matched FLOPs (Tay et al., 2023; Prairie et al., 2026). Prior work closes this gap with sparse or expert capacity (Tan et al., 2023; Csordás et al., 2024; Chen et al., 2026b; Nie et al., 2026), low-rank relaxations of weight tying (Bae et al., 2025a), token-wise adaptive recursion depth (Bae et al., 2025b; Goyal et al., 2026; Knupp et al., 2026; Yu et al., 2026; Shu et al., 2026), or parallel sampling at inference (Geiping et al., 2025b).

Subquadratic attention. A complementary line of work makes each pass cheaper by replacing softmax attention with linear or near-linear sequence mixers. Linear attention (Katharopoulos et al., 2020) rewrites attention through kernel feature maps as a recurrent state update, linking it to fast-weight programming and associative memory (Ba et al., 2016; Schlag et al., 2021; Irie et al., 2021). This has led to a broad family of recurrent mixers: RetNet (Sun et al., 2023), Gated Linear Attention (Yang et al., 2024b), HGRN2 (Qin et al., 2024), DeltaNet and its gated variant (Yang et al., 2024c; 2025), Mamba-2 (Dao & Gu, 2024), Mamba-3 (Lahoti et al., 2026), and RWKV-7 (Peng et al., 2025). Negative eigenvalues (Grazzi et al., 2025) and Householder products (Siems et al., 2025) further improve their state tracking. Since purely linear models can struggle with exact recall (Arora et al., 2025), hybrid architectures interleave linear and softmax layers, as in Jamba (Lieber et al., 2024), Kimi Linear (Kimi Team et al., 2025), and Olmo Hybrid (Merrill et al., 2026), and pretrained softmax Transformers can be distilled into hybrid or linear models (Goldstein et al., 2026; Li et al., 2025b). Another approach keeps softmax attention but sparsifies it, from streaming attention sinks (Xiao et al., 2024) to hardware-aligned and receptive-field-scaling sparse designs (Yuan et al., 2025; Chen et al., 2025; DeepSeek-AI et al., 2025). These methods make each pass cheaper, while looping reuses parameters across passes; LT2 combines the two. Historical KV replay also reuses computation across depth, similar to CoTFormer (Mohtashami et al., 2025), whose later passes attend to earlier ones, and YOCO (Sun et al., 2024), which builds one global KV cache from early layers.

6 Conclusion

We studied LT2, looped Transformers with linear-time attention, and showed that looping benefits linear attention more than full attention. Historical KV replay writes all earlier loops' key–value pairs into the current loop's state in reverse order, so that early depth acts as a global memory. Without adding parameters, it improves language modeling and recall, including needle retrieval beyond the training length. First-loop replay keeps about 70% of the gain at the cost of plain looping.

AI USE STATEMENT

In this work, we used generative AI tools (large language models) to proofread the paper, edit it for readability, and review the manuscript before submission. The review gave feedback on the experiments and on the precision of the mathematical statements. Following it, we corrected the range of the write strength in Theorem B.1, whose endpoint values are now reached as limits, and removed an inaccurate example and an unused lemma from Appendix B.1. We checked every such change ourselves. We have not used generative AI tools to generate synthetic data sets, develop theoretical models or conceptual frameworks, formulate the paper's mathematical claims, provide critical ingredients for their proofs, propose or refine hypotheses, design research methodology or experiments, implement methods, assist with translation, clean or reformat datasets, support qualitative or thematic data analysis, or interpret results. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix C gives the full pre-training setup: data and tokenization, model scales, looped variants and layer mixes, optimization, seeds, and the evaluation suite. Section 4 gives the losses, data, token counts, and sequence lengths of every conversion stage. Appendix D gives the chunkwise forms of historical KV replay for training, prefill, and decoding, with their costs, and Appendix G gives the recall, needle-retrieval, and speed protocols. Appendix F lists the numbers behind Figures 5–7. Appendix B.1 states the assumptions of each theoretical result and gives complete proofs. Code is available at the anonymous repository linked in the abstract.

REFERENCES

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2025. URL https://arxiv.org/abs/2402.18668.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora, 2025a. URL https://arxiv.org/abs/2410.20672.

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. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*. NeurIPS, 2025b.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models, 2019. URL https://arxiv.org/abs/1909.01377.

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

Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of data science*. Cambridge University Press, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Guanxu Chen, Dongrui Liu, and Jing Shao. Loop as a bridge: Can looped transformers truly link representation space and natural language outputs?, 2026a. URL https://arxiv.org/abs/2601.10242.

Lida Chen, Dong Xu, Chenxin An, Xintao Wang, Yikai Zhang, Jiangjie Chen, Zujie Liang, Feng Wei, Jiaqing Liang, Yanghua Xiao, and Wei Wang. Powerattention: Exponentially scaling of receptive fields for effective sparse attention, 2025. URL https://arxiv.org/abs/2503.03588.

Yilong Chen, Naibin Gu, Junyuan Shang, Zhenyu Zhang, Yuchen Feng, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Mixture of universal experts: Scaling virtual width via depth-width transformation, 2026b. URL https://arxiv.org/abs/2603.04971.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 619–634, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.49. URL https://aclanthology.org/2021.emnlp-main.49/.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D Manning. Moeut: Mixture-of-experts universal transformers. *Advances in Neural Information Processing Systems*, 37:28589–28614, 2024.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, and et al. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019. URL https://arxiv.org/abs/1807.03819.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019. URL https://arxiv.org/abs/1903.00161.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization, 2025. URL https://arxiv.org/abs/2409.15647.

Zitian Gao, Lynx Chen, Yihao Xiao, He Xing, Ran Tao, Haoming Luo, Joey Zhou, and Bryan Dai. Universal reasoning model, 2025. URL https://arxiv.org/abs/2512.14693.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *International Conference on Machine Learning*, pp. 15130–15152. PMLR, 2024a.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. On the role of depth and looping for in-context learning with task diversity, 2024b. URL https://arxiv.org/abs/2410.21698.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025a. URL https://arxiv.org/abs/2502.05171.

Jonas Geiping, Xinyu Yang, and Guinan Su. Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models, 2025b. URL https://arxiv.org/abs/2510.14961.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Daniel Goldstein, Eric Alcaide, Janna Lu, and Eugene Cheah. Radlads: Rapid attention distillation to linear attention decoders at scale, 2026. URL https://arxiv.org/abs/2505.03005.

Zixuan Gong, Yong Liu, and Jiaye Teng. What makes looped transformers perform better than non-recursive ones, 2026. URL https://arxiv.org/abs/2510.10089.

Sahil Goyal, Swayam Agrawal, Gautham Govind Anil, Prateek Jain, Sujoy Paul, and Aditya Kusupati. Elt: Elastic looped transformers for visual generation, 2026. URL https://arxiv.org/abs/2604.09168.

Alex Graves. Adaptive computation time for recurrent neural networks, 2016. URL https://arxiv.org/abs/1603.08983.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues, 2025. URL https://arxiv.org/abs/2411.12537.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, Ashima Suvarna, Benjamin Feuer, Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik Sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah Pratt, Vivek Ramanujan, Jon Saad-Falcon, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A. Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alexandros G. Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers, 2021. URL https://arxiv.org/abs/2106.06295.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL https://arxiv.org/abs/1705.03551.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *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, 2025. URL https://arxiv.org/abs/2510.26692.

Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. Depth-recurrent attention mixtures: Giving latent reasoning the attention it deserves, 2026. URL https://arxiv.org/abs/2601.21582.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles, 2026. URL https://arxiv.org/abs/2603.15569.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2025a. URL https://arxiv.org/abs/2406.11794.

Yanhong Li, Songlin Yang, Shawn Tan, Mayank Mishra, Rameswar Panda, Jiawei Zhou, and Yoon Kim. Distilling to hybrid attention models via kl-guided layer selection, 2025b. URL https://arxiv.org/abs/2512.20569.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model, 2024. URL https://arxiv.org/abs/2403.19887.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Chuan Li, Kyle Lo, Saumya Malik, DJ Matusz, Benjamin Minixhofer, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A. Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. Olmo hybrid: From theory to practice and back, 2026. URL https://arxiv.org/abs/2604.03444.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. CoTFormer: A chain of thought driven architecture with budget-adaptive computation cost at inference. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2310.10845.

Ying Nie, Kai Han, Hongguang Li, Hang Zhou, Tianyu Guo, Enhua Wu, Xinghao Chen, and Yunhe Wang. Versatileffn: Achieving parameter efficiency in llms via adaptive wide-and-deep reuse, 2026. URL https://arxiv.org/abs/2512.14531.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. Rwkv-7 "goose" with expressive dynamic state evolution, 2025. URL https://arxiv.org/abs/2503.14456.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models, 2026. URL https://arxiv.org/abs/2604.12946.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures, 2019. URL https://arxiv.org/abs/1901.03429.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016. URL https://arxiv.org/abs/1606.05250.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers, 2025. URL https://arxiv.org/abs/2502.17416.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Wen-Jie Shu, Xuerui Qiu, Rui-Jie Zhu, Harold Haodong Chen, Yexin Liu, and Harry Yang. Loopvit: Scaling visual arc with looped transformers, 2026. URL https://arxiv.org/abs/2602.02156.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. DeltaProduct: Improving state-tracking in linear RNNs via householder products. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.10297.

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.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.05254.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In Nafise Sadat Moosavi, Iryna Gurevych, Yufang Hou, Gyuwan Kim, Young Jin Kim, Tal Schuster, and Ameeta Agrawal (eds.), *Proceedings of the Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pp. 78–90, Toronto, Canada (Hybrid), July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.sustainlp-1.5. URL https://aclanthology.org/2023.sustainlp-1.5/.

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 169–179, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.12. URL https://aclanthology.org/2023.emnlp-main.12/.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling? In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 12342–12364, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.825. URL https://aclanthology.org/2023.findings-emnlp.825/.

Mathurin Videau, Badr Youbi Idrissi, Daniel Haziza, Luca Wehrstedt, Jade Copet, Olivier Teytaud, and David Lopez-Paz. Meta Lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms, 2024a. URL https://arxiv.org/abs/2311.12424.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 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, 2024b. URL https://arxiv.org/abs/2312.06635.

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, 2024c.

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

Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, You Wu, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. Spiralformer: Looped transformers can learn hierarchical dependencies via multi-resolution recursion, 2026. URL https://arxiv.org/abs/2602.11698.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

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, 2025. URL https://arxiv.org/abs/2510.25741.

A Adaptive Computation Time for Looped Transformers

This appendix describes Adaptive Computation Time (ACT), introduced by Graves (2016), for Looped Transformers and explains why we use a fixed number of loop iterations during pre-training.

A.1 Per-Token, Input-Dependent Compute Allocation

In the basic Looped Transformer, every token at every input passes through exactly T loop iterations. ACT relaxes this constraint by letting the number of iterations depend on (i) the input sequence and (ii) the token position $t \in \{1, \ldots, L\}$ within it. We call this property *adaptivity*: the model can spend more compute on positions whose representations are still changing between iterations and stop early on positions that have already stabilized. Two tokens in the same sequence may therefore halt at different iterations τ_t, and the same token may halt at different iterations across different inputs.

A.2 Halting Probabilities and Halting Rule

Following Gao et al. (2025), ACT adds a small auxiliary network (a linear layer followed by a sigmoid) that maps the hidden state at position t and iteration τ to a scalar in $(0, 1)$:

$$p_t^{(\tau)} = \sigma\Big(\mathbf{w}^\top \mathbf{h}_t^{(\tau)} + b\Big) \in (0, 1), \tag{11}$$

where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are learned parameters shared across positions and iterations. We call $p_t^{(\tau)}$ the *halting probability*: it represents the model's estimate, conditioned on the current hidden state, of how likely it is that no further loop iterations are needed for position t.

Position t keeps iterating until the running sum of halting probabilities crosses $1 - \epsilon$:

$$\tau_t \;=\; \min\Big\{ \tau \;\Big|\; \sum_{\tau'=1}^{\tau} p_t^{(\tau')} \;\geq\; 1 - \epsilon \Big\}, \tag{12}$$

where ϵ is a small positive constant (we use $\epsilon = 0.01$, matching the value used by (Dehghani et al., 2019)). Once position t halts, its hidden state is frozen at $\mathbf{h}_t^{(\tau_t)}$ and does not participate in further updates. The threshold form $1 - \epsilon$, rather than exactly 1, ensures that the rule can be triggered after a single iteration if $p_t^{(1)}$ is sufficiently large. Without the slack ϵ, at least two iterations would always be required because $p_t^{(\tau)} < 1$ by construction.

A.3 Pondering Cost and Turing Completeness

To prevent the model from trivially driving every $p_t^{(\tau)}$ to a small value and exhausting the maximum iteration budget on every token, ACT adds a *pondering cost* to the training loss that penalizes the expected number of iterations per position. A scalar hyperparameter weights this cost, controlling the trade-off between accuracy and compute.

The combination of input-dependent halting and unbounded effective depth is what allows looped Transformers with ACT to simulate arbitrary Turing machines (Pérez et al., 2019). Without ACT, a Looped Transformer with a fixed iteration count T has bounded computational depth and is therefore not Turing complete in the same formal sense.

A.4 Why We Use Fixed Iterations During Pre-training

Despite its theoretical appeal, ACT poses several practical difficulties at pre-training scale:

- **Optimization instability.** The halting rule is non-differentiable, and the standard relaxation couples the pondering-cost gradient with the main-loss gradient in ways that can cause sudden jumps in the average iteration count during training.
- **Sensitivity to the pondering weight.** Small changes in the pondering hyperparameter can push the model between two extremes: halting after a single iteration on every token, or never halting until the iteration cap is reached.

- **Throughput loss from ragged halting.** When different positions in the same batch halt at different iterations, the implementation must either pad to the longest unhalted position—giving up the savings ACT was meant to provide—or use specialized ragged kernels.

The Ouro authors report these instabilities for the Ouro model family at pre-training scale, and our experiments reproduce the same behavior. We therefore use a fixed iteration count T throughout pre-training in the main paper and leave a stable ACT variant for future work.

B Expressivity of Looped DPLR Linear Attention

B.1 Loop × DPLR linear attention: expressivity analysis

We analyze how unrolling a DPLR linear-attention block for T iterations enriches its state-transition operator. We show two things in sequence. (i) A single block applies a rank-1 update to the recurrent state, while T loops, when their factors act on one state (as in historical KV replay), compose into an update of rank up to T. (ii) By the Cartan–Dieudonné theorem, this composition is expressive enough to realize any orthogonal transformation in $\mathrm{O}(d_k)$ once $T \geq d_k$. Throughout, the spectral norm of the operator stays bounded by 1, so stability is preserved.

Setup. A DPLR mixer has a recurrent state $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$ that evolves as

$$\mathbf{S}_t \;=\; \mathbf{A}_t\,\mathbf{S}_{t-1} \;+\; \beta_t\,\mathbf{k}_t\mathbf{v}_t^\top, \qquad \mathbf{A}_t \;=\; \big(\mathbf{I} - \beta_t \mathbf{k}_t\mathbf{k}_t^\top\big)\,\mathrm{Diag}(\boldsymbol{\alpha}_t), \tag{13}$$

where the symbols denote the following quantities:

- $\mathbf{S}_t \in \mathbb{R}^{d_k \times d_v}$ is the recurrent state at step t, with key dimension d_k and value dimension d_v.
- $\mathbf{k}_t \in \mathbb{R}^{d_k}$ is a unit-norm key vector ($\|\mathbf{k}_t\|_2 = 1$).
- $\mathbf{v}_t \in \mathbb{R}^{d_v}$ is the value vector.
- $\beta_t \in (0, 2)$ is a scalar gain controlling the strength of the rank-1 update. Values above 1 give the factor below the negative eigenvalue $1 - \beta_t$ (Grazzi et al., 2025).
- $\boldsymbol{\alpha}_t \in [0, 1]^{d_k}$ is the decay, applied as $\mathrm{Diag}(\boldsymbol{\alpha}_t) \in \mathbb{R}^{d_k \times d_k}$. It is channel-wise in KDA; GDN uses a scalar decay α_t, i.e., $\boldsymbol{\alpha}_t = \alpha_t \mathbf{1}$ and $\mathrm{Diag}(\boldsymbol{\alpha}_t) = \alpha_t \mathbf{I}$.
- $\mathbf{I} \in \mathbb{R}^{d_k \times d_k}$ is the identity matrix.

Geometrically, the factor $(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)$ is a generalized Householder transformation: it shrinks the component of any vector along $\mathbf{k}_t$ (by a factor $1 - \beta_t$) and leaves the orthogonal complement untouched. The decay $\mathrm{Diag}(\boldsymbol{\alpha}_t)$ is applied first. The whole map $\mathbf{A}_t \in \mathbb{R}^{d_k \times d_k}$ is therefore a *rank*-1 *perturbation of the diagonal matrix* $\mathrm{Diag}(\boldsymbol{\alpha}_t)$, not of the identity: with $\boldsymbol{\alpha}_t = \frac{1}{2}\mathbf{1}$, $\mathbf{k}_t = \mathbf{e}_1$, $\beta_t = 1$ and $d_k = 3$, $\mathbf{A}_t - \mathbf{I} = \mathrm{diag}(-1, -\frac{1}{2}, -\frac{1}{2})$ has rank 3.

When the same block is unrolled for T loops at step t with per-loop parameters $\{\boldsymbol{\alpha}_t^{(\tau)}, \beta_t^{(\tau)}, \mathbf{k}_t^{(\tau)}\}_{\tau=1}^{T}$ and its writes act on one state, as in historical KV replay, the state is decayed once, with the last loop's $\boldsymbol{\alpha}_t^{(T)}$, and the T writes are applied undecayed. The state-transition operator is then

$$\mathbf{A}_t^{\text{eff}} \;=\; \mathbf{P}_t\,\mathrm{Diag}\big(\boldsymbol{\alpha}_t^{(T)}\big), \qquad \mathbf{P}_t \;=\; \prod_{\tau=1}^{T}\Big(\mathbf{I} - \beta_t^{(\tau)}\mathbf{k}_t^{(\tau)}\mathbf{k}_t^{(\tau)\top}\Big). \tag{14}$$

Lemma 1 (Rank of the write product)**.** *Let $\mathbf{G}_j = \mathbf{I} - \beta_j \mathbf{k}_j \mathbf{k}_j^\top$ for $j = 1, \ldots, \tau$, with unit keys and $\beta_j \in (0, 2)$, and let $\mathbf{K} = [\mathbf{k}_1, \ldots, \mathbf{k}_\tau]$. Then*

$$\mathbf{G}_1\mathbf{G}_2 \cdots \mathbf{G}_\tau = \mathbf{I} + \mathbf{K}\mathbf{C}\mathbf{K}^\top$$

for an upper-triangular $\mathbf{C} \in \mathbb{R}^{\tau \times \tau}$ with $\mathbf{C}_{jj} = -\beta_j$. Hence $\mathbf{G}_1 \cdots \mathbf{G}_\tau - \mathbf{I}$ has rank at most $\mathrm{rank}\,\mathbf{K} \leq \tau$, and exactly τ when the keys are linearly independent.

Proof. Expand the product. Each term takes $-\beta_j \mathbf{k}_j \mathbf{k}_j^\top$ from some factors $j_1 < \cdots < j_m$ and $\mathbf{I}$ from the others, so it equals $(-1)^m \beta_{j_1} \cdots \beta_{j_m} (\mathbf{k}_{j_1}^\top \mathbf{k}_{j_2}) \cdots (\mathbf{k}_{j_{m-1}}^\top \mathbf{k}_{j_m})\, \mathbf{k}_{j_1} \mathbf{k}_{j_m}^\top$, a scalar times $\mathbf{k}_{j_1} \mathbf{k}_{j_m}^\top$ with $j_1 \leq j_m$. Summing these scalars by first and last index gives $\mathbf{C}$, which is therefore upper

triangular. Its diagonal comes only from the single-factor terms, so $\mathbf{C}_{jj} = -\beta_j$. If the keys are linearly independent, $\mathbf{K}$ has full column rank, so $\operatorname{rank}(\mathbf{K}\mathbf{C}\mathbf{K}^\top) = \operatorname{rank}\mathbf{C} = \tau$, because $\mathbf{C}$ is triangular with a nonzero diagonal. □

So $\mathbf{A}_t^{\text{eff}} = \operatorname{Diag}(\boldsymbol{\alpha}_t^{(T)}) + \mathbf{K}\mathbf{C}\mathbf{K}^\top \operatorname{Diag}(\boldsymbol{\alpha}_t^{(T)})$ is a rank-$\leq T$ perturbation of a diagonal matrix. The two extremes in the main text are the two corners of the lemma: identical keys give $\operatorname{rank}\mathbf{K} = 1$, and orthogonal keys make every overlap $\mathbf{k}_i^\top \mathbf{k}_j$ vanish, so $\mathbf{C} = -\operatorname{Diag}(\beta_1, \ldots, \beta_T)$. We now proceed to the geometric consequences.

Reflections, rotations, and arbitrary orthogonal maps. Stepping up from rank to geometry, we now compare the transformations reachable by a single block versus T stacked blocks. With $\boldsymbol{\alpha}_t^{(\tau)} = \mathbf{1}$, each factor $\mathbf{I} - \beta_t^{(\tau)} \mathbf{k}_t^{(\tau)} \mathbf{k}_t^{(\tau)\top}$ tends to the Householder reflection $\mathbf{I} - 2\mathbf{k}_t^{(\tau)} \mathbf{k}_t^{(\tau)\top}$ as $\beta_t^{(\tau)} \to 2$ and to $\mathbf{I}$ as $\beta_t^{(\tau)} \to 0$. Since $\beta_t^{(\tau)} \in (0, 2)$ is open, both endpoints are reached only as limits, so below we say that a matrix is *realized* if the operator can approximate it arbitrarily well. A single block can therefore realize any reflection, but no rotation other than $\mathbf{I}$: an orthogonal limit needs $|\det \mathbf{A}_t| = |1 - \beta_t| \prod_i \alpha_{t,i} = 1$, hence $\boldsymbol{\alpha}_t = \mathbf{1}$ and $\beta_t \in \{0, 2\}$, which gives $\mathbf{I}$ or a reflection (determinant -1). Loop unrolling lifts this restriction:

Theorem B.1 (Universal orthogonal representation)**.** *Let $T \geq d_k$. Every orthogonal matrix $\mathbf{Q} \in \mathrm{O}(d_k)$ is realized by $\mathbf{A}_t^{\text{eff}}$: for every $\varepsilon > 0$ there exist per-loop parameters with $\beta_t^{(\tau)} \in (0, 2)$ such that $\|\mathbf{A}_t^{\text{eff}} - \mathbf{Q}\|_2 < \varepsilon$.*

Proof. By the Cartan–Dieudonné theorem, every orthogonal matrix $\mathbf{Q} \in \mathrm{O}(d_k)$ can be written as a product of at most d_k Householder reflections:

$$\mathbf{Q} \;=\; \prod_{\tau=1}^{m} \big(\mathbf{I} - 2\,\mathbf{k}^{(\tau)} \mathbf{k}^{(\tau)\top}\big), \qquad m \leq d_k,$$

for some unit vectors $\mathbf{k}^{(\tau)} \in \mathbb{R}^{d_k}$. Set $\boldsymbol{\alpha}_t^{(\tau)} = \mathbf{1}$, $\beta_t^{(\tau)} = 2$, and $\mathbf{k}_t^{(\tau)} = \mathbf{k}^{(\tau)}$ for $\tau \leq m$, and set $\beta_t^{(\tau)} = 0$ (identity factor) for $\tau > m$. At these endpoint values, (14) gives $\mathbf{A}_t^{\text{eff}} = \mathbf{Q}$. Since $\mathbf{A}_t^{\text{eff}}$ is continuous in the per-loop parameters, moving every $\beta_t^{(\tau)}$ into $(0, 2)$, and every decay slightly below 1 if the decay range excludes 1, changes $\mathbf{A}_t^{\text{eff}}$ by less than ε in spectral norm. □

This is the geometric content of the upgrade from rank-1 to rank-T: looping the same DPLR block converts a reflection-only operator into one that covers the full orthogonal group.

C Pre-training Setup

This appendix describes the experimental setup behind the language modeling results. We pre-train every model from scratch on the FineWeb-Edu corpus at two parameter scales, 0.6B and 1.3B, under a fixed 100B-token budget. Our implementation builds on the Lingua framework (Videau et al., 2024), and all linear-attention variants use the kernels of the FLA library (Yang & Zhang, 2024). Code is available at `https://anonymous.4open.science/r/LT2-326C/README.md`.

Data and tokenization. All runs draw from the FineWeb-Edu 100BT shard, packed at sequence length 4096. A held-out split of this shard is excluded from training and used only to measure perplexity. We tokenize with the Llama tiktoken tokenizer (vocabulary size 128,256) and prepend a BOS and append an EOS token to every document. The data loader runs asynchronously with a prefetch buffer of 1024 shards.

Token budget. Every model is trained for 255,000 optimizer steps at sequence length 4096. With the per-scale batch sizes given below, this corresponds to roughly 100 billion training tokens, i.e. about one epoch over the training portion of the FineWeb-Edu 100BT shard. The same token budget is used for every variant, so all comparisons use the same data budget.

Model scales. We use two parameter scales:

- **0.6B**: hidden dimension 1024, 25 physical layers, 16 attention heads, FFN multiple-of 256, RoPE base $\theta = 10{,}000$.
- **1.3B**: hidden dimension 2048, 16–25 physical layers (depending on the layer mix described below), 16 attention heads, same FFN and RoPE settings.

We call the larger scale "1.3B" because the exact parameter count varies slightly across variants (DSA layers are heavier than dense blocks). Width and head count are fixed.

Looped variants. Every looped variant uses $T = 4$ loops over the physical layer stack: the same parameters are applied four times in sequence, so the effective depth is $4 \times n_{\text{layers}}$ while the parameter count stays at the single-pass value. Every looped model, including the Looped Transformer, uses the learned per-channel residual gate $\boldsymbol{\rho}_\tau$ between iterations (Section 2.1). We do not use cross-block residuals. Non-looped baselines run a standard Transformer stack of the same physical depth.

Layer mixes. The "hybrid" variants interleave a sub-quadratic mixer with periodic full softmax-attention layers in a 4:1 pattern: four mixer layers followed by one full-attention layer, repeated. For the 1.3B hybrid runs the physical stack has 16 mixer layers and 4 (or 3) full-attention layers. For 0.6B the stack has 17 mixer layers and 4 full-attention layers. The mixers include Gated DeltaNet (GDN), DeltaNet, KDA, RetNet, HGRN2, Mamba2, and DSA. Pure baselines replace the mixer slots with the same operator and remove the full-attention layers. Sub-quadratic mixers use the `flash-linear-attention` kernels. GDN layers, with and without replay, use a write strength $\beta_t \in (0, 2)$, which allows negative eigenvalues in the state transition (Grazzi et al., 2025). Full-attention layers use FlashAttention-3, except for sliding-window variants which use the FMHA path with `window_left = w-1`, `window_right = 0` and a default window size of $w = 256$ at 0.6B and $w = 2048$ at 1.3B.

Optimization. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient clipping at norm 1.0, and bf16 mixed precision. Peak learning rate is 3×10^{-4} for 1.3B runs and 1.5×10^{-4} for 0.6B runs (lowered for stability with the looped sliding-window 0.6B model). The schedule is linear warmup, then cosine decay to a floor of $10^{-6}\times$ the peak rate. Warmup is 5,000 steps for every run. Weight decay is 0.1 for almost all runs (0.033 for the 1.3B 4:1 window baseline, needed for stability).

Seeds. Every model in Table 3 is trained with three seeds, and the Avg. subscripts there give the standard deviation over the three runs. The first run uses seed 777 for the trainer (data order) and seed 42 for initialization. The other two runs use different seeds. Unless stated otherwise, all other experiments use a single run with the first pair of seeds.

Distributed setup. Training is distributed with FSDP (`full_shard`) and `torch.compile` enabled wherever the kernel allows it (we disable compilation only for DSA, whose TileLang kernel is incompatible with `torch.compile`). We do not use tensor parallelism. Per-GPU micro-batches of 2–12 sequences are combined with gradient accumulation between 1 and 6 steps to reach roughly 4×10^5 tokens per step. Over 255,000 steps this gives the $\sim$100B-token budget.

Evaluation. We report validation loss and perplexity on the held-out split of the FineWeb-Edu 100BT shard, which no model sees during training, and zero-shot accuracy with the LM-Evaluation-Harness on eight tasks: ARC-Easy, ARC-Challenge, HellaSwag, PIQA, WinoGrande, OpenBookQA, SciQ, and BoolQ. Avg. is the unweighted mean of these eight accuracies. It is the Avg. reported in every pre-training table, while the converted Ouro models use their own suite. All numbers use the final checkpoint of each run.

D Chunkwise Form of Historical KV Replay

At loop τ, each token decays the state once, then makes τ delta-rule writes, $\mathcal{W}_t^{(\tau)}$ first and $\mathcal{W}_t^{(1)}$ last; first-loop replay is the single-write case. The same WY form handles these writes at two levels: across tokens for training and prefill (Appendix D.1), and within one token for decoding (Appendix D.2).

D.1 TIME-LEVEL CHUNKS (TRAINING AND PREFILL)

The kernel splits the sequence into chunks of C tokens processed in parallel via the WY form. Take chunk c with n writes and let $\mathbf{S}_{c-1}$ be the state before it. $\mathbf{K}_c \in \mathbb{R}^{n\times d_k}$ and $\mathbf{V}_c \in \mathbb{R}^{n\times d_v}$ stack the keys and values of the writes, $\mathbf{B}_c = \mathrm{Diag}(\beta_1, \ldots, \beta_n)$ their write strengths, and γ_i is the product of all decays from the start of the chunk up to write i, with $\mathbf{\Gamma}_c = \mathrm{Diag}(\gamma_1, \ldots, \gamma_n)$. Then

$$\mathbf{L}_c = \mathrm{tril}\big(\mathbf{B}_c\mathbf{K}_c\mathbf{K}_c^\top \odot \mathbf{D}_c,\, -1\big), \qquad (\mathbf{D}_c)_{ij} = \gamma_i/\gamma_j, \tag{15}$$

$$\mathbf{U}_c = (\mathbf{I} + \mathbf{L}_c)^{-1}\,\mathbf{B}_c\mathbf{V}_c, \qquad \mathbf{W}_c = (\mathbf{I} + \mathbf{L}_c)^{-1}\,\mathbf{B}_c\mathbf{\Gamma}_c\mathbf{K}_c, \tag{16}$$

$$\mathbf{R}_c = \mathbf{U}_c - \mathbf{W}_c\,\mathbf{S}_{c-1}, \tag{17}$$

$$\mathbf{O}_c = \mathbf{\Gamma}_c^q\,\mathbf{Q}_c\,\mathbf{S}_{c-1} + (\mathbf{Q}_c\mathbf{K}_c^\top \odot \mathbf{M}_c)\,\mathbf{R}_c, \tag{18}$$

$$\mathbf{S}_c = \gamma_n\,\mathbf{S}_{c-1} + \mathbf{K}_c^\top\,\mathrm{Diag}\big(\gamma_n/\gamma_1, \ldots, \gamma_n/\gamma_n\big)\,\mathbf{R}_c, \tag{19}$$

where $\mathbf{L}_c$ is the strictly lower-triangular, decay-weighted Gram matrix of the keys and $\mathbf{U}_c$ holds the values after the WY correction. $\mathbf{Q}_c$ stacks the queries. If $\ell(i)$ is the last write of query i's token, then $\mathbf{\Gamma}_c^q = \mathrm{Diag}(\gamma_{\ell(1)}, \ldots, \gamma_{\ell(C)})$, and $\mathbf{M}_c$ is the decay-weighted causal mask, $(\mathbf{M}_c)_{ij} = \gamma_{\ell(i)}/\gamma_j$ for $j \le \ell(i)$ and 0 otherwise. $\mathbf{U}_c$ does not depend on $\mathbf{S}_{c-1}$. The term $-\mathbf{W}_c\mathbf{S}_{c-1}$ adds what the delta rule subtracts: the part of each value that the incoming state already stores under that key. Row i of $\mathbf{R}_c$ is $\beta_i\big(\mathbf{v}_i - \gamma_i\mathbf{S}_{i-1}^\top\mathbf{k}_i\big)$, with $\mathbf{S}_{i-1}$ the state after write $i-1$, which is exactly the residual of the sequential update. Without this term, the chunkwise form would match the sequential recurrence only when $\mathbf{S}_{c-1} = \mathbf{0}$. Without replay, each token makes one write, so $n = C$ and $\ell(i) = i$.

Replay only changes what is stacked: at loop τ, each token adds τ rows to $\mathbf{K}_c$ and $\mathbf{V}_c$ (its pairs in replay order, with value rows that are copies of $\mathbf{v}_t^{(1)}$ under the shared value), so $n = \tau C$, while $\mathbf{Q}_c$ keeps C rows and each query sees every write of its own and earlier tokens. The decay is applied only at each token's first write, so the other writes of a token have decay 1 in γ_i. The WY solve and the state update keep the same form. Only the number of rows grows from C to τC. We checked these equations numerically against the sequential recurrence, with a nonzero incoming state and zero values, for both plain GDN and replay. Replay is also causal: later tokens change the logits at position t by at most 3×10^{-6} (fp32).

D.2 DEPTH-LEVEL CHUNKS (DECODING)

Fix token t and loop τ, and drop the index t. Let $\bar{\mathbf{S}}$ be the decayed state, $\mathbf{G}^{(j)} = \mathbf{I} - \beta^{(j)}\mathbf{k}^{(j)}\mathbf{k}^{(j)\top}$, $\mathbf{K} = [\mathbf{k}^{(1)}, \ldots, \mathbf{k}^{(\tau)}]$, and $\mathbf{V} = [\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(\tau)}]^\top$.

Step 1: unroll. Expanding Eq. (5),

$$\mathbf{S}^{(\tau)} = \mathbf{G}^{(1)} \cdots \mathbf{G}^{(\tau)}\,\bar{\mathbf{S}} + \sum_{j=1}^{\tau} \mathbf{G}^{(1)} \cdots \mathbf{G}^{(j-1)}\,\beta^{(j)}\mathbf{k}^{(j)}\mathbf{v}^{(j)\top}. \tag{20}$$

Step 2: merge. By Lemma 1, the product is $\mathbf{I} + \mathbf{K}\mathbf{C}\mathbf{K}^\top$. Grouping its terms by their last factor shows that column j of $\mathbf{K}\mathbf{C}$ is $-\beta^{(j)}\mathbf{G}^{(1)} \cdots \mathbf{G}^{(j-1)}\mathbf{k}^{(j)}$, so the sum equals $-\mathbf{K}\mathbf{C}\mathbf{V}$ and

$$\mathbf{S}^{(\tau)} = \bar{\mathbf{S}} + \mathbf{K}\mathbf{C}\big(\mathbf{K}^\top\bar{\mathbf{S}} - \mathbf{V}\big), \tag{21}$$

one update of rank at most τ (with the shared value, $\mathbf{V} = \mathbf{1}\mathbf{v}^{(1)\top}$).

Step 3: grow $\mathbf{C}$ across loops. Since $\mathbf{G}^{(1)} \cdots \mathbf{G}^{(\tau)} = (\mathbf{G}^{(1)} \cdots \mathbf{G}^{(\tau-1)})\,\mathbf{G}^{(\tau)}$, loop τ adds a column:

$$\mathbf{C}^{(\tau)} = \begin{bmatrix} \mathbf{C}^{(\tau-1)} & -\beta^{(\tau)}\mathbf{C}^{(\tau-1)}\mathbf{K}^{(\tau-1)\top}\mathbf{k}^{(\tau)} \\ \mathbf{0}^\top & -\beta^{(\tau)} \end{bmatrix}, \qquad \mathbf{C}^{(1)} = -\beta^{(1)}. \tag{22}$$

All inputs come from the current token, so no cache across tokens is needed. Each loop still reads and writes its state once, as in Looped GDN; only the flops grow, from $O(d_k d_v)$ to $O(\tau d_k d_v)$.

D.3 COST SUMMARY

Table 7 collects the costs per token and layer. n_τ is the number of rows per time-level chunk at loop τ.

Table 7: **Cost of replay per token and layer**, general T (value at $T{=}4$).

	Looped GDN	First-loop replay	Full replay
Training and prefill (time-level chunks of C tokens)			
Writes	T (4)	T (4)	$T(T{+}1)/2$ (10)
Rows per chunk n_τ	C	C	τC
WY solve, $\propto \sum_\tau n_\tau^2$	T (4)	T (4)	$\sum_\tau \tau^2$ (30)
State update, $\propto \sum_\tau n_\tau$	T (4)	T (4)	$T(T{+}1)/2$ (10)
Loops with K / V projections	T / T	1 / 1	T / 1
Decoding (depth-level chunks)			
State reads and writes	T (4)	T (4)	T (4)
Rank of the update at loop τ	1	1	τ

E Write Order of Historical KV Replay

Table 8 spells out, for $T{=}4$, the order in which forward and reverse replay write a token's key–value pairs at each loop. Eq. (6) gives LT2's reverse order, and Section 3.3 and Table 4 compare the two.

Table 8: **Forward vs. reverse historical KV replay at position t.** $(\mathbf{k},\mathbf{v})^{(j)}$: the token's pair from loop j. Writes go left to right. Only the last is not altered by a later write. LT2 uses reverse replay.

Loop	**Forward replay** (loop 1 → current)	**Reverse replay** (current → loop 1)
$\tau{=}1$	$(\mathbf{k},\mathbf{v})^{(1)}$	$(\mathbf{k},\mathbf{v})^{(1)}$
$\tau{=}2$	$(\mathbf{k},\mathbf{v})^{(1)} \to (\mathbf{k},\mathbf{v})^{(2)}$	$(\mathbf{k},\mathbf{v})^{(2)} \to (\mathbf{k},\mathbf{v})^{(1)}$
$\tau{=}3$	$(\mathbf{k},\mathbf{v})^{(1)} \to (\mathbf{k},\mathbf{v})^{(2)} \to (\mathbf{k},\mathbf{v})^{(3)}$	$(\mathbf{k},\mathbf{v})^{(3)} \to (\mathbf{k},\mathbf{v})^{(2)} \to (\mathbf{k},\mathbf{v})^{(1)}$
$\tau{=}4$	$(\mathbf{k},\mathbf{v})^{(1)} \to (\mathbf{k},\mathbf{v})^{(2)} \to (\mathbf{k},\mathbf{v})^{(3)} \to (\mathbf{k},\mathbf{v})^{(4)}$	$(\mathbf{k},\mathbf{v})^{(4)} \to (\mathbf{k},\mathbf{v})^{(3)} \to (\mathbf{k},\mathbf{v})^{(2)} \to (\mathbf{k},\mathbf{v})^{(1)}$
First write	loop-1 pair	current loop's pair
Last write	current loop's pair ✗	loop-1 pair, refreshed last ✓
Intuition	the newest loop's view is written last and partly overwrites the loop-1 view	early depth (loop 1) serves as memory for later depth (loop 4) like YOCO (Sun et al., 2024)

F Numbers for the Replay Ablations

This appendix lists the numbers behind Figures 5, 6, and 7 in three groups. All models use $T{=}4$, and the columns are the same throughout:

- **Writes**: delta-rule writes per token and layer (per GDN layer for the hybrids).
- **Train**: training throughput at 4k context, k tokens/s per H100.
- **Speed**: decoding tokens/s at 32k context, batch size 1.
- **PPL**: perplexity on the held-out FineWeb-Edu split.
- **Avg.**: mean zero-shot accuracy over the eight downstream tasks.
- **Rec.**: average natural-text recall, inputs truncated to 2k tokens.

(a) Replay vs. other ways to spend compute. At 0.6B, Looped GDN is the reference. The next four models spend similar or more compute without replay. The last five are replay variants that differ in which pairs are written and in which order.

Table 9: **Replay vs. spending similar compute without replay** (0.6B, T=4).

Model	Writes	Train	Speed	PPL↓	Avg.↑	Rec.↑
Looped GDN (reference)	4	41.2	19.4	12.06	55.72	29.83
More compute, no replay						
more loops (T=6)	6	27.8	12.9	11.91	56.28	31.17
wider (0.78B)	4	32.1	14.6	11.83	56.52	30.64
Gated DeltaProduct, n_h=2	8	36.4	15.3	11.87	56.41	31.76
Gated DeltaProduct, n_h=3	12	32.2	13.2	11.79	56.73	32.35
Replay variants						
Forward replay	10	34.5	18.4	12.03	55.78	30.12
Current + previous loop	7	37.9	18.9	11.94	56.17	31.05
Current + loop 1	7	37.9	18.9	11.58	58.06	34.27
First-loop replay	4	40.9	20.1	11.64	57.66	33.89
Full replay	10	34.6	18.4	11.46	58.50	35.61

(b) Replay changed at test time. We take the 1.3B model trained with full (reverse) replay and fix its weights. Only the replay used at inference changes: dropping the previous loop's pair, writing the pairs in forward order, dropping the loop-1 pair, or removing replay.

Table 10: **Replay changed at test time** (1.3B, trained with full replay, T=4).

Replay at test time	Writes	PPL↓	Avg.↑	Rec.↑
As trained	10	9.44	61.10	41.10
Drop the pair of loop $\tau-1$	7	9.61	60.64	40.27
Forward order	10	10.57	57.96	33.42
Drop the loop-1 pair	7	10.80	57.51	32.58
No replay	4	11.31	56.62	30.24

(c) Replay on hybrid LT2. The 0.6B hybrids combine GDN layers with full-attention (Full+GDN) or DSA (GDN+DSA) layers. Replay is applied to the GDN layers only.

Table 11: **Replay on hybrid LT2** (0.6B, T=4, replay on the GDN layers only).

Model	Writes	PPL↓	Avg.↑	Rec.↑
Hybrid (Full+GDN)	4	11.43	58.77	36.82
w/ first-loop replay	4	11.19	59.69	38.91
w/ full replay	10	11.09	60.14	39.96
Hybrid (GDN+DSA)	4	11.85	56.53	32.41
w/ first-loop replay	4	11.50	57.92	35.53
w/ full replay	10	11.37	58.52	36.74

G Recall and Speed Evaluation

We follow Mamba-3 (Lahoti et al., 2026) for natural-text recall and decoding-speed measurements.

Needle retrieval. NIAH Single-1/2/3 are the three single-needle tasks of RULER, run at context lengths of 4k, 8k, 16k, and 32k. All models are trained at 4k, so the longer lengths test retrieval beyond the training length. Table 12 lists the per-task scores and their average.

Natural-text recall. The six natural-text recall tasks (SWDE, SQuAD (Rajpurkar et al., 2016), FDA, TriviaQA (Joshi et al., 2017), NQ (Kwiatkowski et al., 2019), and DROP (Dua et al., 2019)) are recall-intensive tasks used to evaluate linear attention (Arora et al., 2025) and GDN. Following GDN and Mamba-3, each input is truncated to 2k tokens, half the 4k training length. Documents in several of these tasks (e.g., SQuAD, TriviaQA, and DROP) are only a few hundred tokens long, so

Table 12: **Needle retrieval by context length** (1.3B, T=4, trained at 4k). S1–S3: RULER NIAH Single-1/2/3. Avg.: their mean.

	4k				8k				16k				32k			
Model	S1	S2	S3	Avg.	S1	S2	S3	Avg.	S1	S2	S3	Avg.	S1	S2	S3	Avg.
Looped Transformer	100.0	98.4	93.6	97.3	1.6	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Looped Mamba2	72.0	61.0	6.0	46.3	34.0	18.4	0.8	17.7	12.0	4.2	0.0	5.4	3.6	0.8	0.0	1.5
Looped GDN	95.6	93.8	38.4	75.9	90.8	31.6	4.2	42.2	68.0	9.4	0.6	26.0	36.4	2.0	0.0	12.8
forward replay	95.4	94.0	38.8	76.1	91.2	32.4	4.6	42.7	68.6	10.2	0.8	26.5	37.0	2.4	0.0	13.1
first-loop replay	96.0	94.6	40.2	76.9	95.2	45.8	9.6	50.2	84.4	20.6	2.8	35.9	58.8	6.4	0.4	21.9
reverse (full) replay	96.4	95.0	41.0	77.5	96.6	52.4	12.8	53.9	88.2	26.8	4.0	39.7	66.4	9.2	0.8	25.5

this suite measures recall within the training length. The NIAH sweep above measures retrieval at and beyond it. Every natural-text recall number in the paper is the average over these six tasks at 2k.

Speed measurement. All speeds are measured on one H100 at batch size 1, with the same inference code for every model. The speedups compare all models under the same setup. Absolute numbers depend on the inference stack and will differ with an optimized serving engine. The scaling with context, however, is implementation-independent: at each decoding step, a full-attention layer reads a KV cache that grows with context, while a GDN layer reads a fixed-size state (12.1 GB vs. 0.05 GB at 32k for the 0.6B models).