LOOPBACK TRANSFORMER: TOWARDS A UNIFIED VIEW OF RESIDUAL LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers reduce parameter count by repeatedly applying shared blocks, but parameter sharing alone does not determine how the results of repeated computation are reused. Building on the depth-memory view of residual connections, we analyze how the written representation and weighting rule constrain the contribution of each update. Storing a full block output also writes back its input, changing how earlier information is carried forward. Reading individual updates with a separate input path exposes independent controls over input strength, total update weight, and allocation. Guided by this analysis, we develop the Loopback Transformer, which stores individual block updates and keeps the shared stack's input on a separate path. Geometric decay controls total update weight and supplies a recency prior, while attention learns how to allocate that weight. On ClimbMix, Loopback uses only about 40% and 58% of the dense controls' Transformer parameters at the 191M and 374M scales, respectively, while achieving lower validation loss and better downstream performance at the studied training budgets. Comparisons use the same training-token budget and computational depth within each scale.

1 INTRODUCTION

Looped Transformers reduce parameter count by repeatedly applying shared blocks to an evolving hidden representation (Dehghani et al., 2018; Lan et al., 2019). Sharing separates computational depth from the number of distinct blocks and supports latent reasoning and inference-time scaling (Saunshi et al., 2025; Geiping et al., 2026). Each block visit applies the same transformation to an evolving hidden state and can therefore produce a different update. This raises a fundamental question: *how should these repeated updates from different visits be stored, carried forward, and used?* Parameter sharing determines which transformation is reused, but does not determine how the resulting updates accumulate.

Residual connections provide a natural lens for this problem because they turn network depth into a form of memory: each layer writes new computation into a residual stream that is carried forward to later layers. Standard residual addition accumulates earlier branch outputs with fixed unit coefficients (He et al., 2016), whereas Attention Residuals learns how to weight those branch outputs and the input (Team et al., 2026). Hyper-Connections and Hyperloop further introduce learned pathways across residual streams or loop visits (Zhu et al., 2025; Zeitoun et al., 2026). From a depth-memory view (Team et al., 2026, Sec. 6), these methods differ in how they *write, transport, and read* intermediate computation. We examine how these choices constrain the contribution of earlier updates when the same blocks are repeatedly applied.

Two couplings matter. First, a block's full output contains both its input and the new update it computes. Weighting that output therefore also weights the history already carried by the input. Storing updates separately instead lets the reader assign a coefficient directly to each new update. Second, a joint softmax over the original input and stored updates makes them compete for a fixed total weight, whereas ordinary addition fixes the input coefficient but assigns every update weight one, causing total update weight to grow with the number of block applications. These observations motivate separate controls over *input strength*, *total update weight*, and *allocation across updates*.

Guided by these observations, we develop the **Loopback Transformer** (Section 5). It stores individual block updates rather than full block outputs and keeps the shared stack's input on a separate path with

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.

coefficient one. Geometric decay sets the total update weight and provides a prior favoring recent visits; attention learns how to distribute that weight across the stored updates.

On ClimbMix (Diao et al., 2026), Loopback achieves lower validation loss and higher seven-task mean zero-shot accuracy than its matched-depth dense baselines using about 40%, 58%, and 67% of their Transformer parameters at the 191M, 374M, and 778M scales respectively (Table 2 and Fig. 4). All comparisons use the same budget of training tokens and layer depth; parameter counts exclude embeddings and the vocabulary head. At 1.4B, quality metrics are mixed (Section 6.2). In our ablations, the decay-only rule accounts for most of the gain over the plain loop, with a further improvement from learned allocation. We also compare alternative memory interfaces in Section 6.

Our major contributions are threefold:

- Building on depth-memory views, we characterize how storing full outputs or updates changes the contribution of earlier computation, and identify constraints on input strength, total update weight, and weight allocation.
- Guided by this view, we develop the Loopback Transformer, which stores block updates, preserves a separate input path, and separates decay-controlled accumulation from learned allocation without changing shared blocks.
- We demonstrate improved parameter efficiency on ClimbMix across three model scales, achieving lower validation loss and higher downstream performance than matched-depth dense controls with substantially fewer Transformer parameters.

2 Related work

Parameter sharing and residual reuse. Models with shared depth use weight tying (Dehghani et al., 2018; Lan et al., 2019), relaxed tying with low-rank adaptations (Bae et al., 2025), latent reasoning (Saunshi et al., 2025; Geiping et al., 2026), and adaptive recursion (Bae et al., 2026). Hyperloop learns connections between loop visits (Zeitoun et al., 2026). Residual pathways use gating, addition, and concatenation (Srivastava et al., 2015; He et al., 2016; Huang et al., 2018), scaling (Bachlechner et al., 2021; Wang et al., 2024), depth mixtures (Bapna et al., 2018), multi-stream mixing (Zhu et al., 2025; Xie et al., 2025), and attention-based aggregation (Team et al., 2026; Luo et al., 2026a;b). We study constraints on update coefficients, separating input strength, total weight, and allocation rather than claiming delta storage as new.

Attention and recurrent memory. Sequence memory includes attention and additive fast weights (Vaswani et al., 2017; Katharopoulos et al., 2020; Schlag et al., 2021), retention and gating (Sun et al., 2023; Yang et al., 2023; Qin et al., 2024), structured state-space dynamics (Gu et al., 2021; Gu & Dao, 2023; Dao & Gu, 2024), delta-rule corrections (Yang et al., 2024; 2025; Team et al., 2025; Peng et al., 2025), and optimization-based memory (Von Oswald et al., 2023; Zhang et al., 2026). We adapt these operations along depth, retaining each block's token attention; results concern residual interfaces, not the original architectures' efficiency.

3 Preliminaries

At a fixed token position, let $\boldsymbol{h}_s \in \mathbb{R}^d$ denote the hidden state and g_s denote an attention or MLP branch, including normalization but excluding its skip; token attention remains causal. Ordinary residual addition (He et al., 2016) gives

$$\boldsymbol{h}_{s+1} = \boldsymbol{h}_s + g_s(\boldsymbol{h}_s) = \boldsymbol{h}_0 + \sum_{i=0}^{s} g_i(\boldsymbol{h}_i). \tag{1}$$

Full Attention Residuals (Team et al., 2026) instead learns a read of the input $\boldsymbol{v}_0 = \boldsymbol{h}_0$ and branch outputs $\boldsymbol{v}_{i+1} = g_i(\boldsymbol{h}_i)$:

$$\boldsymbol{h}_s = \sum_{i=0}^{s} \pi_{s,i} \boldsymbol{v}_i, \qquad \pi_{s,i} = \mathrm{softmax}_i\big(\boldsymbol{q}_s^\top \mathrm{RMSNorm}(\boldsymbol{v}_i)\big). \tag{2}$$

The query $\boldsymbol{q}_s \in \mathbb{R}^d$ scores normalized sources, while the read combines their original values. Hyper-Connections (HC) (Zhu et al., 2025) instead maintains n streams in $\boldsymbol{H}_s \in \mathbb{R}^{d\times n}$:

$$\boldsymbol{h}_s^{\text{in}} = \boldsymbol{H}_s \boldsymbol{p}_s, \qquad \boldsymbol{H}_{s+1} = \boldsymbol{H}_s \boldsymbol{W}_s + g_s(\boldsymbol{h}_s^{\text{in}})\boldsymbol{b}_s^\top. \tag{3}$$

Here $\boldsymbol{p}_s, \boldsymbol{b}_s \in \mathbb{R}^n$ read and write, and $\boldsymbol{W}_s \in \mathbb{R}^{n\times n}$ mixes streams; these controls may depend on the current streams.

4 Bridging the Gap – A Unified View

The residual rules in Section 3 differ in what they retain and how they reuse it. Building on depth-memory views (Team et al., 2026), we formalize these choices through a write–transport–read interface: what each computation writes, how existing memory is carried forward or mixed, and how that memory is read to form a hidden state.

4.1 A Common Memory Interface

Let $\boldsymbol{M}_s \in \mathbb{R}^{d\times m_s}$ store memory columns and $\boldsymbol{x}_s = \boldsymbol{M}_s \boldsymbol{p}_s$ be the computation's input. Indexing each new value $\boldsymbol{v}_s$ by its writing application, the rules in Section 3 fit

$$\begin{aligned} \boldsymbol{M}_{s+1} &= \boldsymbol{M}_s \boldsymbol{W}_s + \boldsymbol{v}_s \boldsymbol{b}_s^\top, \\ \boldsymbol{h}_{s+1} &= \boldsymbol{M}_{s+1} \boldsymbol{r}_s. \end{aligned} \tag{4}$$

Here $\boldsymbol{W}_s \in \mathbb{R}^{m_s\times m_{s+1}}$ carries old memory, $\boldsymbol{b}_s, \boldsymbol{r}_s \in \mathbb{R}^{m_{s+1}}$ write and read, and $\boldsymbol{p}_s \in \mathbb{R}^{m_s}$ forms the input; these controls may depend on content. Single-path models pass $\boldsymbol{h}_{s+1}$ onward; stream models retain $\boldsymbol{M}_{s+1}$ and form the next input with $\boldsymbol{p}_{s+1}$. Table 1 maps the families to this interface; we first examine their choice of written value.

4.2 Delta: Full Outputs or Residual Updates

In Huginn's prefix–middle–coda layout (Geiping et al., 2026), topology $[P\text{L} \to K\text{L}\,(\times R) \to C\text{L}]$ (Zeitoun et al., 2026) denotes P prefix blocks, K shared blocks visited R times, and C coda blocks. Thus $P + K + C$ distinct blocks execute $P + RK + C$ applications, with prefix output $\boldsymbol{h}_0$. Extra visits extend history without adding blocks. Middle application $s = rK + k$ uses $f_s = f_k$, including token attention, an MLP, and internal residuals. Its net update is

$$\boldsymbol{\delta}_s = f_s(\boldsymbol{x}_s) - \boldsymbol{x}_s. \tag{5}$$

At sublayer granularity, $f_s(\boldsymbol{x}_s) = \boldsymbol{x}_s + g_s(\boldsymbol{x}_s)$ gives $\boldsymbol{\delta}_s = g_s(\boldsymbol{x}_s)$. A binary switch selects the stored value:

$$\boldsymbol{v}_s^{(\chi)} = f_s(\boldsymbol{x}_s) - \chi \boldsymbol{x}_s = \boldsymbol{\delta}_s + (1-\chi)\boldsymbol{x}_s, \qquad \chi \in \{0,1\}. \tag{6}$$

Substituting $\boldsymbol{v}_s = \boldsymbol{v}_s^{(\chi)}$ into Eq. (4) exposes the effective transport of earlier information:

$$\boldsymbol{M}_{s+1} = \boldsymbol{M}_s \underbrace{[\boldsymbol{W}_s + (1-\chi)\boldsymbol{p}_s \boldsymbol{b}_s^\top]}_{\boldsymbol{A}_s} + \boldsymbol{\delta}_s \boldsymbol{b}_s^\top. \tag{7}$$

Full-output writes ($\chi = 0$) add $\boldsymbol{p}_s \boldsymbol{b}_s^\top$, so constraints on $\boldsymbol{W}_s$ need not hold for $\boldsymbol{A}_s$. Delta writes ($\chi = 1$) remove this term, not the input dependence of $\boldsymbol{\delta}_s$. An unrestricted carry can absorb it, making this a parameterization choice rather than an expressivity advantage (Fig. 1).

4.3 Decay and History Weighting

Decay multiplicatively retains past writes, whereas attention can increase an older value's weight at a new destination. With $\boldsymbol{M}_0 = \boldsymbol{h}_0 \boldsymbol{a}_0^\top$, $\boldsymbol{a}_0 \in \mathbb{R}^{m_0}$, unrolling Eq. (7) gives

$$\boldsymbol{h}_{s+1} = a_{s+1}\boldsymbol{h}_0 + \sum_{i=0}^{s} c_{s,i}\boldsymbol{\delta}_i, \tag{8}$$

$$c_{s,i} = \boldsymbol{b}_i^\top \boldsymbol{A}_{i+1} \cdots \boldsymbol{A}_s \boldsymbol{r}_s, \qquad a_{s+1} = \boldsymbol{a}_0^\top \boldsymbol{A}_0 \cdots \boldsymbol{A}_s \boldsymbol{r}_s. \tag{9}$$

Write, transport, and read determine each coefficient; empty products are identity, and a_{s+1} includes input carried inside values. This trajectory-conditioned identity is not a linear model or Jacobian. It applies headwise; feature projections require operator-valued coefficients (Section A).

Nonnegative normalized reads of ordinary additive states impose $c_{s,0} \geq \cdots \geq c_{s,s}$; delta reads remove this ordering (Section A.2). For $c_i \geq 0$ and $B = \sum_i c_i > 0$, write

$$\boldsymbol{h} = a\boldsymbol{h}_0 + B \sum_i \pi_i \boldsymbol{\delta}_i, \qquad \pi_i = c_i/B, \quad \sum_i \pi_i = 1. \tag{10}$$

These are input strength a, total weight B (not a representation norm), and allocation π_i. Joint softmax over the input and raw updates ties $a + B = 1$; a separate input path decouples them. The identities also hold without tying: shared blocks need not share reuse controls.

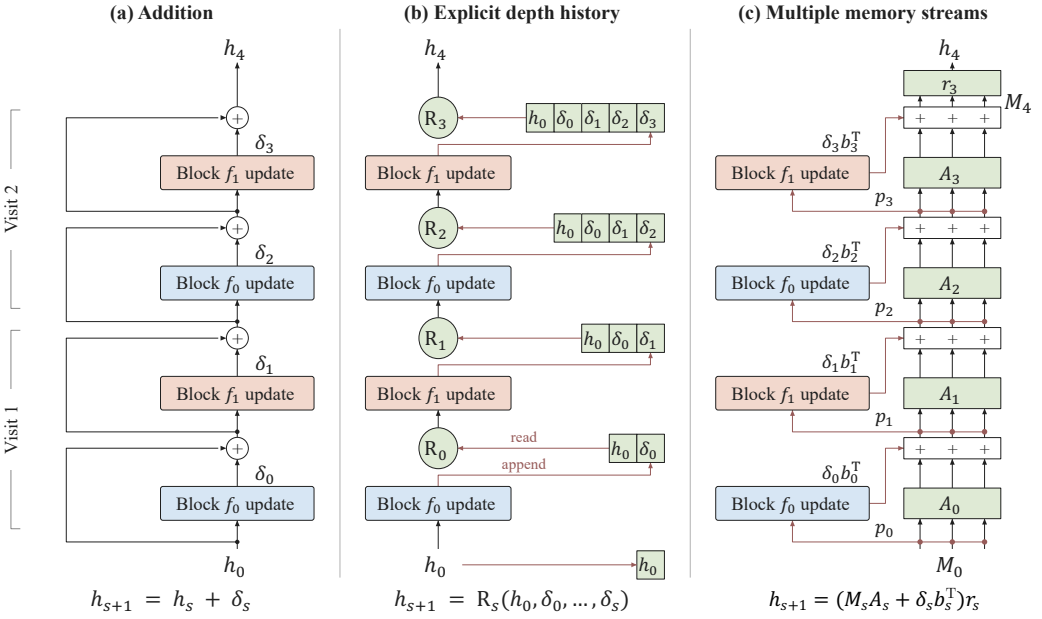


Figure 1: **Same shared blocks, three residual pathways.** Repeated colors denote shared blocks emitting net updates. (a) Addition. (b) Causal snapshots of one history bank, read after each write. (c) Multiple streams with input read, transport, write, and final readout (Sections 4.1 and 4.2).

4.4 Residual families and the choice of readout

Table 1 contrasts explicit history, recurrent compression, and stream transport. Residual rows write branch outputs $\boldsymbol{v}_s = g_s(\boldsymbol{x}_s)$; Addition and Full AttnRes start from $\boldsymbol{M}_0 = \boldsymbol{h}_0$. In Full AttnRes, $\boldsymbol{e}_{\text{new}}$ selects the appended column and $\boldsymbol{\pi}_{s+1}$ reads it with prior sources (Eq. (2)); "full" means all history, not full-block outputs. iHC[1] uses identity carry. Native linear attention uses the positive normalizer $\boldsymbol{z}_{s+1}^{\top}\boldsymbol{q}_s$, where $\boldsymbol{z}_{s+1} = \sum_{i=0}^{s} \boldsymbol{k}_i$; the other recurrent cores omit this denominator. The recurrent depth adapters evaluated in Section 6.4 also omit it (Section B.2); derivations and implementation conventions are in Section B.

Loopback instantiates these choices with a separate input path, decay-controlled total weight, and learned delta allocation (Section 5).

5 Loopback Transformer

Loopback keeps the input coefficient at one and stores raw block updates (Fig. 2). A fixed decay rule sets their prior and total weight; attention learns how to redistribute it.

[1] Identity Hyper-Connections; see the official Hy4-preview model card.

Table 1: Residual and attention cores in Eq. (4), omitting projections, feedthrough terms, and output gates. Native rules and depth adapters are detailed in Section B.

Common interface: $\boldsymbol{M}_{s+1} = \boldsymbol{M}_s \boldsymbol{W}_s + \boldsymbol{v}_s \boldsymbol{b}_s^\top, \qquad \boldsymbol{h}_{s+1} = \boldsymbol{M}_{s+1} \boldsymbol{r}_s.$

Method	**Carry $\boldsymbol{W}_s$**	**Write $\boldsymbol{b}_s$**	**Read $\boldsymbol{r}_s$**
Addition (He et al., 2016)	1	1	1
Attention family: explicit history and compressed recurrent memory			
Full AttnRes (Team et al., 2026)	$[\boldsymbol{I}, \boldsymbol{0}]$	$\boldsymbol{e}_{\text{new}}$	$\boldsymbol{\pi}_{s+1}$ (Eq. (2))
Recurrent cores start from $\boldsymbol{M}_0 = 0$; $\boldsymbol{k}_s, \boldsymbol{q}_s$ are keys and queries after any feature mapping.			
Linear (Katharopoulos et al., 2020)	$\boldsymbol{I}$	$\boldsymbol{k}_s$	$\frac{\boldsymbol{q}_s}{\boldsymbol{z}_{s+1}^\top \boldsymbol{q}_s}$
RetNet (Sun et al., 2023)	$\gamma \boldsymbol{I}$	$\boldsymbol{k}_s$	$\boldsymbol{q}_s$
GLA (Yang et al., 2023)	$\mathrm{Diag}(\boldsymbol{g}_s)$	$\boldsymbol{k}_s$	$\boldsymbol{q}_s$
Mamba-2 (Dao & Gu, 2024)	$\gamma_s \boldsymbol{I}$	$\tau_s \boldsymbol{k}_s$	$\boldsymbol{q}_s$
HGRN-2 (Qin et al., 2024)	$\mathrm{Diag}(\boldsymbol{g}_s)$	$\boldsymbol{1} - \boldsymbol{g}_s$	$\boldsymbol{q}_s$
DeltaNet (Yang et al., 2024)	$\boldsymbol{I} - \eta_s \boldsymbol{k}_s \boldsymbol{k}_s^\top$	$\eta_s \boldsymbol{k}_s$	$\boldsymbol{q}_s$
Gated DeltaNet (Yang et al., 2025)	$\gamma_s(\boldsymbol{I} - \eta_s \boldsymbol{k}_s \boldsymbol{k}_s^\top)$	$\eta_s \boldsymbol{k}_s$	$\boldsymbol{q}_s$
KDA (Team et al., 2025)	$\mathrm{Diag}(\boldsymbol{g}_s)(\boldsymbol{I} - \eta_s \boldsymbol{k}_s \boldsymbol{k}_s^\top)$	$\eta_s \boldsymbol{k}_s$	$\boldsymbol{q}_s$
RWKV-7 (Peng et al., 2025)	$\mathrm{Diag}(\boldsymbol{g}_s) - \boldsymbol{\kappa}_s(\boldsymbol{a}_s \odot \boldsymbol{\kappa}_s)^\top$	$\boldsymbol{k}_s$	$\boldsymbol{q}_s$
Hyper-connection family: $\boldsymbol{M}_0 = \boldsymbol{h}_0 \boldsymbol{1}^\top$; read gives the next input $\boldsymbol{x}_{s+1}$.			
HC (Zhu et al., 2025)	Learned $\boldsymbol{W}_s$	Learned $\boldsymbol{b}_s$	$\boldsymbol{p}_{s+1}$
mHC (Xie et al., 2025)	$\mathrm{Sinkhorn}(\boldsymbol{U}_s)$	$\boldsymbol{b}_s \geq 0$	$\boldsymbol{p}_{s+1} \geq 0$
iHC[1]	$\boldsymbol{I}$	$\boldsymbol{b}_s \geq 0$	$\boldsymbol{p}_{s+1} \geq 0$

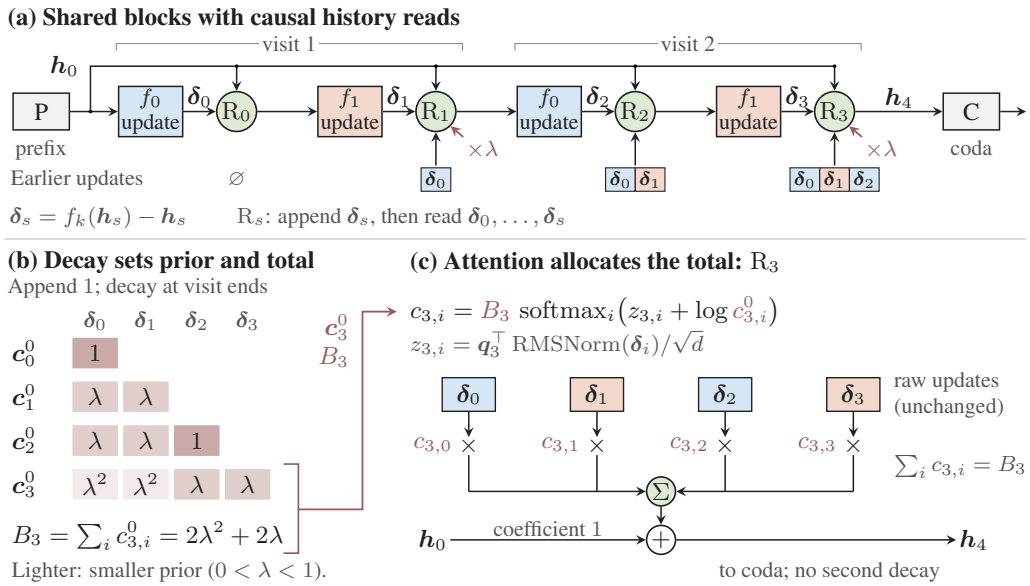


Figure 2: **Loopback separates input, decay, and allocation.** (a) Shared blocks with causal history reads. (b) Visit-end decay sets the prior and total weight. (c) Attention redistributes that total over raw updates, keeping the input coefficient at one.

5.1 GEOMETRIC DECAY SETS THE PRIOR AND TOTAL WEIGHT

A decay-only reference loop applies the shared stack, then shrinks its accumulated change relative to the fixed input $\boldsymbol{h}_0$. One visit is

$$\boldsymbol{h} \leftarrow \boldsymbol{h}_0 + \lambda\big(F(\boldsymbol{h}) - \boldsymbol{h}_0\big), \qquad F = f_{K-1} \circ \cdots \circ f_0, \quad 0 < \lambda \leq 1. \tag{11}$$

Decay follows every visit, including the last before the coda. This reference defines Loopback's prior and initialization, not an extra forward pass.

Each new update enters with weight one, and every visit end multiplies all weights, including the newest, by λ. Counting visit ends from write i through read s gives the prior and its total:

$$c^0_{s,i} = \lambda^{\lfloor (s+1)/K \rfloor - \lfloor i/K \rfloor}, \qquad B_s = \sum_{i=0}^{s} c^0_{s,i}. \tag{12}$$

Superscript 0 marks the fixed prior. For $\lambda < 1$, it favors later visits while controlling the total: $B_{KR-1} = K \sum_{j=1}^{R} \lambda^j$, which equals KR without decay and is bounded by $K\lambda/(1-\lambda)$ for fixed $0 < \lambda < 1$. The bound concerns coefficient mass, not representation norms or loss.

5.2 SOFTMAX ATTENTION REWEIGHTS THE DELTAS

After appending $\boldsymbol{\delta}_s = f_k(\boldsymbol{h}_s) - \boldsymbol{h}_s$ on Loopback's own trajectory, a destination-specific query scores normalized messages and reads their unprojected values:

$$\boldsymbol{k}_i = \frac{\boldsymbol{\delta}_i}{\sqrt{d^{-1}\|\boldsymbol{\delta}_i\|^2 + \varepsilon}}, \qquad z_{s,i} = \boldsymbol{q}_s^\top \boldsymbol{k}_i / \sqrt{d},$$

$$c_{s,i} = B_s \,\mathrm{softmax}_i(z_{s,i} + \log c^0_{s,i}), \qquad \boldsymbol{h}_{s+1} = \boldsymbol{h}_0 + \sum_{i=0}^{s} c_{s,i} \boldsymbol{\delta}_i. \tag{13}$$

Here $\varepsilon > 0$; token-shared queries read content-dependent keys. Attention can favor older messages but preserves $\sum_i c_{s,i} = B_s$ and the unit input coefficient. Decay changes the prior and total, not stored messages; no second decay follows the read.

Zero queries recover the decay-only weights and trajectory; with $\lambda = 1$ they recover ordinary addition. The implementation replaces, rather than adds to, the current mixture (Section D). The one-source first read needs no query, leaving $(KR-1)d$ query parameters. Explicit history costs $O(KRd)$ storage and $O((KR)^2 d)$ total read work per token beyond block computation; parameter savings do not imply a wall-clock speedup (Section 6.5).

6 EXPERIMENTS

We evaluate parameter efficiency, residual design choices, and alternative readers on ClimbMix (Diao et al., 2026). Main comparisons match tokens and executed depth within each scale. Scale names and 20-TPP budgets use the dense control's total parameters; reported Transformer counts exclude embeddings and the vocabulary head. Training and evaluation details are in Section E; Fig. 3 compares learning curves across four scales.

Baselines. Dense uses independent blocks; Plain reuses shared blocks with ordinary addition. Among stream methods (Table 1), HC learns full mixing, mHC constrains carry by Sinkhorn normalization, and iHC fixes identity carry while learning read and write gates. HC and mHC operate at visit boundaries, as does Hyperloop's diagonal carry (Zeitoun et al., 2026); iHC acts after each middle block. AttnRes reads the embedding and earlier sublayer outputs, whereas our multi-head MHAR adaptation reads loop states (Luo et al., 2026b). Looped arms share each setting's topology; these are complete-design comparisons with different placements and sources (Table 7).

6.1 PARAMETER-EFFICIENT LANGUAGE MODELING

We first ask whether improved residual reuse closes Plain's validation-loss gap to matched-depth Dense. At 191M and 374M, Loopback achieves lower loss than both controls and mHC, the strongest residual baseline, using only 40% and 58.3% of Dense's Transformer parameters (Table 2).

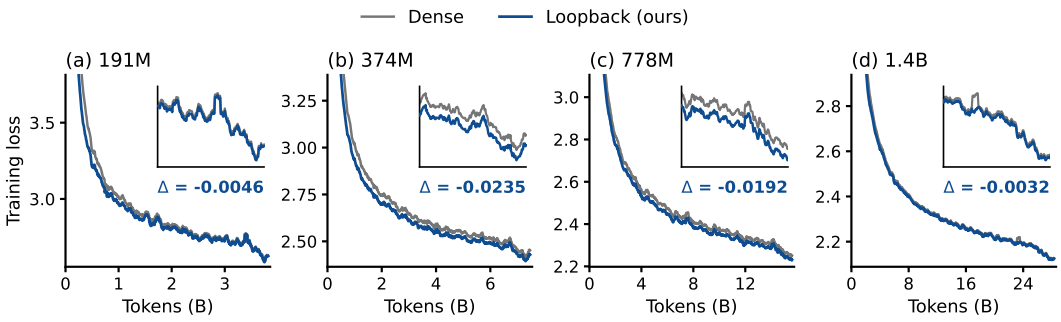


Figure 3: **Training loss of Dense and Loopback.** 100-step moving averages against tokens seen; insets show the last quarter of training; Δ is the final training-loss difference, Loopback minus Dense, over the last 100 steps. Loopback stores 40–67% of Dense's Transformer parameters.

Table 2: **Residual adapters at matched tokens and depth.** Validation loss, zero-shot accuracy (%), and WikiText perplexity. Arrows mark lower-is-better metrics; bold marks the best result per row. Transformer counts exclude embeddings and the vocabulary head; shared weights are counted once.

			Hyper-connection family				Attention family		
Benchmark	Dense	Plain	mHC	HC	iHC	Hyperloop	MHAR	AttnRes	**Loopback**
191M, $[2\text{L} \to 4\text{L}\,(\times 4) \to 2\text{L}]$**, 40% of the control, 3.82B tokens (20 TPP)**									
Transformer params (M)	141.59	56.64	56.93	56.65	57.04	56.79	56.64	56.67	56.65
ClimbMix loss ↓	2.6234	2.6802	2.6250	2.6449	2.6564	2.6489	2.6564	2.6647	**2.6187**
PIQA	65.45	64.91	65.29	64.58	65.34	64.47	64.91	65.29	**65.56**
HellaSwag	33.11	31.93	33.59	32.90	32.70	32.94	32.83	32.48	**33.62**
WinoGrande	51.46	51.30	51.62	51.85	52.64	48.70	49.72	**53.04**	51.62
OpenBookQA	31.60	29.80	31.20	29.00	**32.40**	32.20	31.00	30.60	31.00
ARC-Easy	**47.18**	46.89	46.68	46.13	46.42	45.41	45.62	46.17	46.93
ARC-Challenge	**26.54**	26.45	25.26	25.94	24.66	26.11	26.11	26.19	**26.54**
LAMBADA	23.31	21.97	25.19	23.99	23.50	23.99	23.05	24.35	**25.89**
WikiText word ppl ↓	46.30	50.30	45.86	47.52	48.72	47.87	48.30	48.31	**45.74**
Mean (7 tasks)	39.81	39.03	39.83	39.20	39.67	39.12	39.04	39.73	**40.16**
374M, $[2\text{L} \to 10\text{L}\,(\times 2) \to 2\text{L}]$**, 58.3% of the control, 7.47B tokens (20 TPP)**									
Transformer params (M)	308.33	179.86	180.06	179.87	180.53	179.96	179.86	179.91	179.88
ClimbMix loss ↓	2.4198	2.4415	2.4062	2.4249	2.4217	2.4179	2.4223	2.4601	**2.3971**
PIQA	68.44	68.93	67.08	68.93	**69.31**	68.82	68.66	68.50	69.04
HellaSwag	39.17	38.88	40.32	39.52	39.59	39.33	39.46	37.85	**40.62**
WinoGrande	51.07	51.22	52.80	50.59	49.88	52.25	52.25	51.93	**53.28**
OpenBookQA	32.00	**33.60**	33.00	32.00	33.20	33.00	32.20	**33.60**	33.00
ARC-Easy	52.74	51.43	**54.92**	54.50	52.48	53.91	53.37	51.64	52.78
ARC-Challenge	28.84	29.52	**30.12**	29.78	28.16	29.52	27.73	28.41	28.16
LAMBADA	30.68	29.98	30.95	30.70	30.84	31.61	31.81	28.27	**33.01**
WikiText word ppl ↓	33.71	34.88	32.77	33.94	33.78	33.54	33.83	35.46	**32.41**
Mean (7 tasks)	43.28	43.37	44.17	43.72	43.35	44.07	43.64	42.89	**44.27**

The improvement extends beyond held-out loss to zero-shot task means and WikiText perplexity (Gao et al., 2021), though individual tasks are mixed and small mean margins need not be significant (Table 16). Shallower, parameter-matched Dense controls also trail Plain: simply reducing dense depth does not recover the benefit of repeated computation (Sections E.1 and E.3).

6.2 Parameter efficiency at larger scale

We scale evaluation to 778M and 1.4B to test whether parameter efficiency persists. Loopback uses fewer Transformer parameters and improves all aggregate metrics at 778M; at 1.4B, thirteen-task accuracy improves, but validation loss and seven-task accuracy worsen (Fig. 4 and Table 15). Varying topologies and training recipes prevent isolating a scaling trend (Section E.5).

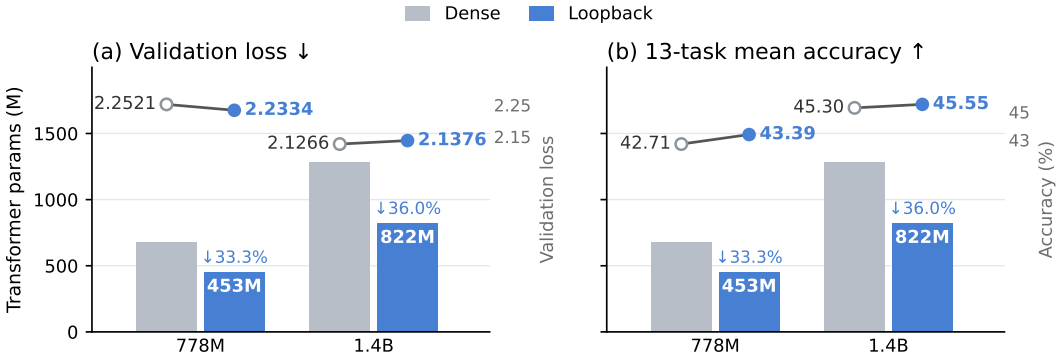


Figure 4: **Fewer Transformer parameters at 778M and 1.4B.** Bars give stored Transformer parameters; the segments above them join Dense to Loopback on (a) ClimbMix validation loss and (b) the thirteen-task mean accuracy. Comparisons match tokens and depth.

6.3 Ablation Study

Table 3: **Decay, attention, and initialization.** Validation loss (↓); shaded cells are Loopback, with its difference from Dense in green. (a) Fixed allocation uses zero queries. (b) Learning rates are not retuned; mHC is included as the strongest baseline of Table 2.

(a) Decay and attention at two scales

	191M (Dense: 2.6234)		374M (Dense: 2.4198)	
Decay	Fixed	Learned	Fixed	Learned
Off ($\lambda = 1$)	2.6802	2.6621	2.4415	2.4308
On ($\lambda = 1/4$)	2.6220	**2.6187** (−0.0047)	2.3999	**2.3971** (−0.0227)

(b) Weight initialization at 191M

Model	Params (M)	$\mathcal{N}(0, 0.02^2)$	GPT-2	PyTorch
Dense	141.59	2.6234	2.6419	2.7145
Plain loop	56.64	2.6802	2.6977	2.6642
mHC	56.93	2.6250	2.6408	2.6278
Loopback	56.65	**2.6187** (−0.0047)	**2.6178** (−0.0241)	**2.6138** (−0.1007)

To separate decay from learned allocation, we toggle both components (Table 3). Decay supplies most of the gain; allocation further improves both scales. To distinguish decay's effects at 191M, uniform allocation at its total weight recovers 78% of its gain, versus 20% for recency alone at Plain's total (Table 9), favoring total-weight control. Testing initialization sensitivity without learning-rate retuning, Loopback retains the lowest loss across three schemes (Table 3; protocols in Section F.4). These tests strongly substantiate the robustness of our method.

6.4 Linear attention and recurrent memory along depth

We adapt nine recurrent cores (Table 1), Mesa, and a LaCT-inspired reader to whole-visit updates, preserving token attention (Section C). Delta Attention Residuals (Luo et al., 2026a) also routes updates, adding their weighted read to the current residual stream.

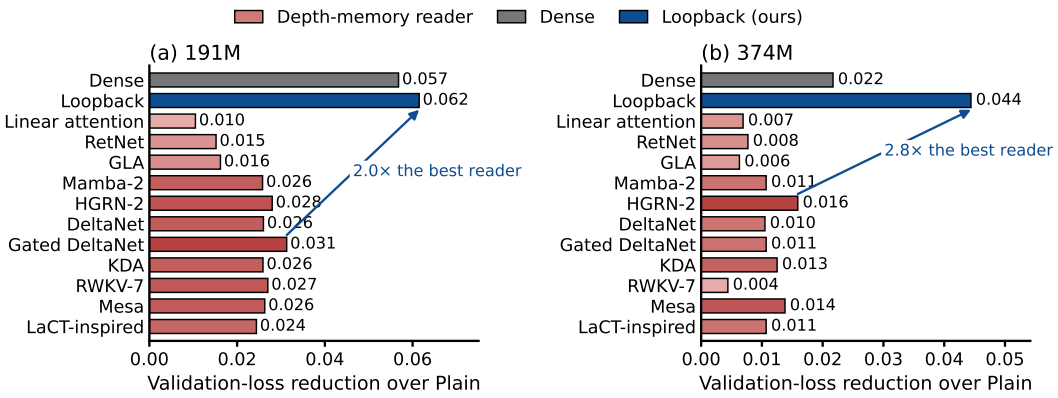


Figure 5: **Depth-memory readers.** ClimbMix validation-loss reduction over Plain at (a) 191M and (b) 374M; Dense and Loopback are references.

At 191M, updates outperform states in every tested family (Table 10), supporting our delta-based design, although heads, gates, and blends also differ. All update readers improve on Plain at both scales but trail Dense and Loopback (Fig. 5). Deltas are thus a useful starting point, not a complete residual design: Loopback also fixes the input path and controls total weight over block updates. At 374M we report only updates; these explicit-history tests measure quality, not recurrent speedups.

6.5 Efficiency: compute, throughput, and memory

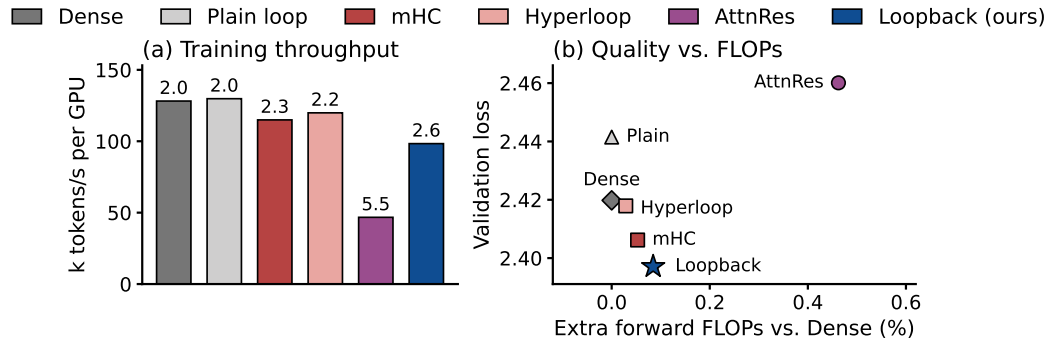


Figure 6: **Compute and efficiency at 374M.** AttnRes and Loopback run fused read kernels. (a) Per-GPU training throughput on eight H100s; bar labels give hours for the 20-TPP budget. (b) Validation loss versus forward FLOPs per token in excess of Dense's. Table 11 gives all three measured scales.

Loopback achieves its quality gains at near-Dense arithmetic cost: at matched executed depth, it adds only 0.04–0.08% forward FLOPs across three measured scales (Fig. 6b, Table 11). With a fused Triton reader (Section G), step-time overhead falls from 33% at 191M to 19% at 778M (Fig. 6a, Table 11); history reads still incur memory traffic. Fewer weights reduce persistent inference storage with the same KV-cache capacity (Table 12), but peak training memory remains higher (Table 11).

7 Conclusion

We present Loopback, a parameter-efficient looped Transformer that learns how to reuse the updates produced by shared computation. Building on depth-memory views, our analysis connects the choice of stored values and read rules to their effective update coefficients. Loopback translates this analysis into a fixed input anchor, a controlled total update weight, and learned allocation over individual updates. It improves aggregate quality over matched-depth Dense controls at 191M, 374M, and 778M, and offers a competitive quality–parameter trade-off at 1.4B. These results highlight a complementary direction for parameter-efficient modeling: beyond deciding which computations to share, designing how their outputs are retained and reused.

ETHICS STATEMENT

This work studies the architecture of language models. All models have at most 1.4B parameters and are trained on ClimbMix, a public pretraining corpus, with evaluation on public benchmarks; no new data were collected and no human subjects were involved. The models are research artifacts that inherit the biases of their training corpus and are not intended for deployment. The method reduces stored parameters at matched computation and does not alter the risks associated with language models in general.

REPRODUCIBILITY STATEMENT

Every training run is specified by the architecture, data, optimizer, schedule, and batch settings in Section E, including the per-scale configuration table and the residual settings of each baseline. Validation uses a fixed held-out ClimbMix shard packed as in training, and downstream results come from lm-evaluation-harness 0.4.12 with zero-shot prompts and full evaluation sets (Section E.2). Section D gives the exact form of Loopback's read and its initialization, Section C the memory-reader interfaces, and Section G the FLOP, throughput, memory, and prefill measurement protocols together with the tests applied to the fused kernel. The training code, configurations, model definitions for every residual rule, and the fused kernel with its tests will be released.

AI USE STATEMENT

Generative AI assisted with manuscript organization, mathematical exposition and verification scripts, figure preparation, language editing, and LaTeX formatting.

REFERENCES

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in artificial intelligence*, pp. 1352–1361. PMLR, 2021.

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

Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3028–3033, 2018.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, et al. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. *Advances in Neural Information Processing Systems*, 38, 2026.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

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.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018. URL https://arxiv.org/abs/1608.06993.

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.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Cheng Luo, Zefan Cai, and Junjie Hu. Delta attention residuals. *arXiv preprint arXiv:2605.18855*, 2026a.

Cheng Luo, Zefan Cai, and Junjie Hu. Multi-head attention residuals. *arXiv preprint arXiv:2607.27230*, 2026b.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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 *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks, 2015. URL https://arxiv.org/abs/1505.00387.

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.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Johannes Von Oswald, Maximilian Schlegel, Alexander Meulemans, Seijin Kobayashi, Eyvind Niklasson, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, et al. Uncovering mesa-optimization algorithms in transformers. *arXiv preprint arXiv:2309.05858*, 2023.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, 2024.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

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, 2024.

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.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William Freeman, and Hao Tan. Test-time training done right. In *International Conference on Learning Representations*, volume 2026, pp. 157604–157638, 2026.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *International Conference on Learning Representations*, volume 2025, pp. 97183–97219, 2025.

A CONDITIONAL EXPANSION OF RESIDUAL MEMORY

A.1 DELTA REPRESENTATION AND EFFECTIVE RETENTION

For $\boldsymbol{x}_s = \boldsymbol{M}_s \boldsymbol{p}_s$, define $\boldsymbol{v}_s^{(\chi)} = f_s(\boldsymbol{x}_s) - \chi \boldsymbol{x}_s$. Then $\boldsymbol{v}_s^{(\chi)} = \boldsymbol{\delta}_s + (1-\chi)\boldsymbol{M}_s \boldsymbol{p}_s$, so

$$\boldsymbol{M}_s \boldsymbol{W}_s + \boldsymbol{v}_s^{(\chi)} \boldsymbol{b}_s^\top = \boldsymbol{M}_s [\boldsymbol{W}_s + (1-\chi)\boldsymbol{p}_s \boldsymbol{b}_s^\top] + \boldsymbol{\delta}_s \boldsymbol{b}_s^\top .$$

This proves Eq. (7) for compatible, possibly content-dependent controls on the actual trajectory. Appending a delta uses $[\boldsymbol{I}, \boldsymbol{0}]$; appending a full output uses $[\boldsymbol{I}, \boldsymbol{p}_s]$. Freely reparameterized carry can absorb this difference, so the distinction is relative to the specified rule.

A.2 CUMULATIVE STATES AND UPDATE COEFFICIENTS

For ordinary additive states $\boldsymbol{h}_j^{\mathrm{add}} = \boldsymbol{h}_0 + \sum_{i<j} \boldsymbol{\delta}_i$, a nonnegative normalized read gives

$$\sum_{j=0}^{s+1} \rho_j \boldsymbol{h}_j^{\mathrm{add}} = \boldsymbol{h}_0 + \sum_{i=0}^{s} \left(\sum_{j=i+1}^{s+1} \rho_j \right) \boldsymbol{\delta}_i, \qquad \rho_j \geq 0, \quad \sum_j \rho_j = 1. \tag{14}$$

The equality follows by exchanging the finite sums: update $\boldsymbol{\delta}_i$ appears in every state with $j > i$. Hence $c_{s,i} = \sum_{j=i+1}^{s+1} \rho_j$ and $1 \geq c_{s,0} \geq \cdots \geq c_{s,s} \geq 0$. Reading deltas separately removes this cumulative-sum restriction, but does not make the updates independent.

For a more general state history with expansion $\boldsymbol{v}_j = a_j \boldsymbol{h}_0 + \sum_i C_{j,i} \boldsymbol{\delta}_i$, a read with state weights $\boldsymbol{\rho}$ induces input coefficient $\sum_j \rho_j a_j$ and update weights $\boldsymbol{\rho}^\top \boldsymbol{C}$. The ordering above is specific to $a_j = 1$ and $C_{j,i} = \mathbf{1}_{i<j}$ with a nonnegative normalized read. It need not hold for reweighted trajectories, whose actual coefficient matrix $\boldsymbol{C}$ must be used.

A.3 STORED-VALUE WEIGHTING AND SCALAR DECAY

A scalar-weighted read separates the explicit input path from stored values:

$$\boldsymbol{h}_{s+1} = a_{s+1}^{\mathrm{in}} \boldsymbol{h}_0 + \sum_{i=0}^{s} w_{s,i} \boldsymbol{v}_i^{(\chi)}. \tag{15}$$

Here a_{s+1}^{in} weights only the explicit path, unlike the expanded coefficient a_{s+1} in Eq. (8), which also includes input carried inside values. Let $\boldsymbol{u}_{s+1} = \sum_{i=0}^{s} w_{s,i} \boldsymbol{v}_i^{(\chi)}$ and $\boldsymbol{u}_0 = 0$. Write scales $\alpha_s \geq 0$ and retention $0 \leq \gamma_s \leq 1$ give

$$\boldsymbol{u}_{s+1} = \gamma_s \boldsymbol{u}_s + \alpha_s \boldsymbol{v}_s^{(\chi)}, \qquad w_{s,i} = \alpha_i \prod_{j=i+1}^{s} \gamma_j. \tag{16}$$

Unrolling the recurrence yields the product, with an empty product equal to one. Ordinary addition uses delta values and unit coefficients; decay after addition sets $\alpha_s = \gamma_s$, scaling old and new terms together. This distinction also explains why Loopback's visit-end decay includes the newest update.

Fixed-query attention and normalization-induced decay. For fixed scores $e_i > 0$, define $Z_s = \sum_{i \leq s} e_i$ and $\boldsymbol{o}_s = Z_s^{-1} \sum_{i \leq s} e_i \boldsymbol{v}_i$. Splitting off the newest term gives

$$\boldsymbol{o}_s = \frac{Z_{s-1}}{Z_s} \boldsymbol{o}_{s-1} + \frac{e_s}{Z_s} \boldsymbol{v}_s.$$

The coefficients sum to one, with $Z_{s-1}/Z_s \in [0, 1)$; the first source has no old term. Changing the query generally invalidates this recurrence from the previous read alone, although an explicit bank can still be reweighted.

A.4 Expansion with additional value sources

This section proves the coefficient expansion in Eq. (9) and extends it to explicit input and position writes. Let $\boldsymbol{M}_s \in \mathbb{R}^{d\times m_s}$, $\boldsymbol{h}_s^{\text{in}} = \boldsymbol{M}_s\boldsymbol{p}_s$, and $\boldsymbol{\delta}_s = f_s(\boldsymbol{h}_s^{\text{in}}) - \boldsymbol{h}_s^{\text{in}}$. Write

$$\begin{aligned}\boldsymbol{M}_{s+1} &= \boldsymbol{M}_s\boldsymbol{A}_s + \boldsymbol{\delta}_s\boldsymbol{b}_s^\top + \boldsymbol{h}_0\boldsymbol{u}_s^\top + \boldsymbol{e}_s\boldsymbol{t}_s^\top,\\ \boldsymbol{h}_{s+1} &= \boldsymbol{M}_{s+1}\boldsymbol{r}_s.\end{aligned} \tag{17}$$

Here $\boldsymbol{A}_s \in \mathbb{R}^{m_s\times m_{s+1}}$, $\boldsymbol{b}_s, \boldsymbol{u}_s, \boldsymbol{t}_s, \boldsymbol{r}_s \in \mathbb{R}^{m_{s+1}}$, and $\boldsymbol{p}_s \in \mathbb{R}^{m_s}$. The optional writes inject the fixed input $\boldsymbol{h}_0$ or an extra value $\boldsymbol{e}_s$, such as a position vector; their write vectors are zero when absent. Rectangular transport permits a growing history bank.

Define $\boldsymbol{P}_{i+1:s} = \boldsymbol{A}_{i+1}\cdots\boldsymbol{A}_s$, with an empty product equal to identity, and $\boldsymbol{D}_i = \boldsymbol{\delta}_i\boldsymbol{b}_i^\top + \boldsymbol{h}_0\boldsymbol{u}_i^\top + \boldsymbol{e}_i\boldsymbol{t}_i^\top$. Then

$$\boldsymbol{M}_{s+1} = \boldsymbol{M}_0\boldsymbol{A}_0\cdots\boldsymbol{A}_s + \sum_{i=0}^{s}\boldsymbol{D}_i\boldsymbol{P}_{i+1:s}. \tag{18}$$

For $s = 0$ this is the update rule. Multiplying the expansion for $\boldsymbol{M}_s$ by $\boldsymbol{A}_s$ and adding $\boldsymbol{D}_s$ proves the induction step. The order of the factors is essential; they need not commute. With $\boldsymbol{M}_0 = \boldsymbol{h}_0\boldsymbol{a}_0^\top$, right-multiplication by $\boldsymbol{r}_s$ gives

$$\boldsymbol{h}_{s+1} = a_{s+1}\boldsymbol{h}_0 + \sum_{i=0}^{s}c_{s,i}\boldsymbol{\delta}_i + \sum_{i=0}^{s}d_{s,i}\boldsymbol{e}_i, \tag{19}$$

where

$$\begin{aligned}c_{s,i} &= \boldsymbol{b}_i^\top\boldsymbol{P}_{i+1:s}\boldsymbol{r}_s, \qquad d_{s,i} = \boldsymbol{t}_i^\top\boldsymbol{P}_{i+1:s}\boldsymbol{r}_s,\\ a_{s+1} &= \boldsymbol{a}_0^\top\boldsymbol{A}_0\cdots\boldsymbol{A}_s\boldsymbol{r}_s + \sum_{i=0}^{s}\boldsymbol{u}_i^\top\boldsymbol{P}_{i+1:s}\boldsymbol{r}_s.\end{aligned} \tag{20}$$

Removing the optional writes yields Eq. (9). Setting the initial memory to zero and writing $\boldsymbol{v}_i$ instead of $\boldsymbol{\delta}_i$ also gives the recurrent read in Eq. (24b).

All operators are evaluated along the generated trajectory. This is an exact value decomposition, not a globally linear model or its Jacobian: derivatives also pass through keys, gates, and reads. Independent heads satisfy the identity headwise; learned feature projections require operator-valued coefficients. Arbitrary nonlinear value maps are outside the scalar expansion.

B Mappings for the evaluated residual adapters

B.1 Stream transport and explicit history

On the delta axis, our HC and iHC adapters write the branch difference $\boldsymbol{\delta}_s$, whereas mHC and Hyperloop write the full branch output $f_s(\boldsymbol{M}_s\boldsymbol{p}_s)$ together with a position value $\boldsymbol{e}_s$. Substitution of $f_s(\boldsymbol{M}_s\boldsymbol{p}_s) = \boldsymbol{M}_s\boldsymbol{p}_s + \boldsymbol{\delta}_s$ gives

$$\begin{aligned}\boldsymbol{M}_{s+1} &= \boldsymbol{M}_s\boldsymbol{W}_s + [f_s(\boldsymbol{M}_s\boldsymbol{p}_s) + \boldsymbol{e}_s]\boldsymbol{b}_s^\top\\ &= \boldsymbol{M}_s(\boldsymbol{W}_s + \boldsymbol{p}_s\boldsymbol{b}_s^\top) + \boldsymbol{\delta}_s\boldsymbol{b}_s^\top + \boldsymbol{e}_s\boldsymbol{b}_s^\top.\end{aligned} \tag{21}$$

This is the $\chi = 0$ case of Eq. (7); its full transport is $\boldsymbol{A}_s = \boldsymbol{W}_s + \boldsymbol{p}_s\boldsymbol{b}_s^\top$. In value-by-stream storage, mHC uses $\boldsymbol{W}_s = \boldsymbol{H}_{\text{res}}^\top$ and constrains this carry by Sinkhorn normalization; the additional branch-input path is not part of that constraint. Hyperloop uses a diagonal carry. iHC instead fixes $\boldsymbol{A}_s = \boldsymbol{W}_s = \boldsymbol{I}$ and learns nonnegative write gates, input reads, and the final-coda read. Its native sublayer rule writes the branch output $g_s(\boldsymbol{x}_s)$; our whole-block adapter writes the net update $\boldsymbol{\delta}_s$.

For a history bank, appending $\boldsymbol{\delta}_s$ uses $\boldsymbol{A}_s = [\boldsymbol{I}, \boldsymbol{0}]$ and a write vector selecting the new column. Appending the full output instead uses $\boldsymbol{A}_s = [\boldsymbol{I}, \boldsymbol{p}_s]$. Attention Residuals jointly normalizes the embedding and earlier sublayer outputs; MHAR reads loop states per head with the blend in Eq. (31). Loopback reads individual block messages with a separate unit-weight input path. Exact placements are given in Table 7.

B.2 Attention reads and recurrent memory cores

Explicit attention. A depth-wise softmax reads earlier computations at one token position:

$$\boldsymbol{o}_s = \sum_{i=0}^{s} \pi_{s,i} \boldsymbol{v}_i^{(\chi)}, \qquad \pi_{s,i} = \mathrm{softmax}_i(\boldsymbol{q}_s^\top \boldsymbol{k}_i / \sqrt{d_k} + b_{s,i}). \tag{22}$$

Score bias $b_{s,i}$, such as a log prior, differs from write vector $\boldsymbol{b}_i$. With fixed old scores, appending a source rescales the old read; a changed query generally needs additional history (Section A.3). Full AttnRes includes the input as an additional source and uses a learned destination query with RMS-normalized keys (Eq. (2)).

Finite-feature attention. Linear attention summarizes values $\boldsymbol{v}_i = \boldsymbol{v}_i^{(\chi)}$ in sufficient statistics (Katharopoulos et al., 2020; Schlag et al., 2021). For a finite feature map $\phi : \mathbb{R}^{d_k} \to \mathbb{R}^m$ with $\kappa(\boldsymbol{q}, \boldsymbol{k}) = \phi(\boldsymbol{q})^\top \phi(\boldsymbol{k}) \geq 0$, associativity gives

$$\boldsymbol{o}_s = \frac{\sum_{i=0}^{s} \kappa(\boldsymbol{q}_s, \boldsymbol{k}_i) \boldsymbol{v}_i}{\sum_{i=0}^{s} \kappa(\boldsymbol{q}_s, \boldsymbol{k}_i)} = \frac{\boldsymbol{S}_s \phi(\boldsymbol{q}_s)}{\boldsymbol{z}_s^\top \phi(\boldsymbol{q}_s)}, \qquad \boldsymbol{S}_s = \sum_{i=0}^{s} \boldsymbol{v}_i \phi(\boldsymbol{k}_i)^\top, \quad \boldsymbol{z}_s = \sum_{i=0}^{s} \phi(\boldsymbol{k}_i). \tag{23}$$

Updating $\boldsymbol{S}_s \in \mathbb{R}^{d \times m}$ and $\boldsymbol{z}_s \in \mathbb{R}^m$ avoids storing past values for either χ. The denominator must be positive; this finite-feature identity is not a general softmax equivalence.

With feature maps absorbed into keys and queries in $\mathbb{R}^m$, gating and key-specific correction fit an *unnormalized* memory core:

$$\boldsymbol{S}_i = \boldsymbol{S}_{i-1} \boldsymbol{A}_i + \boldsymbol{v}_i \boldsymbol{b}_i^\top, \qquad \boldsymbol{S}_{-1} = 0, \tag{24a}$$

$$\boldsymbol{o}_s = \boldsymbol{S}_s \boldsymbol{q}_s = \sum_{i=0}^{s} (\boldsymbol{b}_i^\top \boldsymbol{A}_{i+1} \cdots \boldsymbol{A}_s \boldsymbol{q}_s) \boldsymbol{v}_i. \tag{24b}$$

Linear attention uses $\boldsymbol{A}_i = \boldsymbol{I}$ and $\boldsymbol{b}_i = \boldsymbol{k}_i$; its normalized form divides by $\boldsymbol{z}_s^\top \boldsymbol{q}_s$. Other recurrent cores in Table 1 allow noncommuting transports and signed coefficients. Residual deltas specify *values*; DeltaNet instead corrects memory predictions (Yang et al., 2024). Depth-wise reader implementations are detailed in Section C.

Our recurrent depth adapters use the unnormalized core: keys and queries are L2-normalized with an epsilon floor, but reads omit the scalar kernel normalizer. Coefficients may be negative; the positivity conditions of Eq. (23) are therefore not required (Section C). In Table 1, $\boldsymbol{M}_{s+1}$, $\boldsymbol{W}_s$, and $\boldsymbol{r}_s$ correspond to $\boldsymbol{S}_s$, $\boldsymbol{A}_s$, and $\boldsymbol{q}_s$ in Eq. (24). For the update interface ($\chi = 1$), carry also equals effective transport; for full-output writes, effective transport adds $\boldsymbol{p}_s \boldsymbol{b}_s^\top$ (Eq. (7)).

Decay and key-specific correction. In Table 1, Mamba-2 couples decay and write scale through $\gamma_s = \exp(-a\tau_s)$ with $a, \tau_s > 0$, whereas HGRN-2 writes $\mathbf{1} - \boldsymbol{g}_s$, the complement of retention. The delta rule corrects a memory prediction; it does not denote the block output-minus-input transformation. Its target value can itself be either a full output or a residual update. For these families, let $\bar{\boldsymbol{S}}_i = \boldsymbol{S}_{i-1} \boldsymbol{D}_i$:

$$\boldsymbol{S}_i = \bar{\boldsymbol{S}}_i + \eta_i (\boldsymbol{v}_i - \bar{\boldsymbol{S}}_i \boldsymbol{k}_i) \boldsymbol{k}_i^\top, \qquad \boldsymbol{D}_i = \begin{cases} \boldsymbol{I} & \text{DeltaNet,} \\ \gamma_i \boldsymbol{I} & \text{Gated DeltaNet,} \\ \mathrm{Diag}(\boldsymbol{g}_i) & \text{KDA.} \end{cases} \tag{25}$$

Decay acts before correction toward the new value (Yang et al., 2024; 2025; Team et al., 2025). Thus KDA uses $\boldsymbol{A}_i = \mathrm{Diag}(\boldsymbol{g}_i)(\boldsymbol{I} - \eta_i \boldsymbol{k}_i \boldsymbol{k}_i^\top)$ in value-by-key orientation; reversing the factors generally changes the operator. RWKV-7 subtracts a rank-one correction from diagonal transport. Its normalized removal direction $\boldsymbol{\kappa}_s$ and coordinatewise strength $\boldsymbol{a}_s$ need not align with the write key $\boldsymbol{k}_s$ (Peng et al., 2025).

B.3 Regression readers and their additional state

The nine memory cores in Table 1 directly use Eq. (24a). Two further readers require additional state: Mesa tracks inverse covariance, while the LaCT-inspired reader revisits a stored prefix.

Mesa. Let $\boldsymbol{K} = [\boldsymbol{k}_0, \dots, \boldsymbol{k}_s]$, $\boldsymbol{V} = [\boldsymbol{v}_0, \dots, \boldsymbol{v}_s]$, $\boldsymbol{\Omega} = \mathrm{Diag}(\omega_0, \dots, \omega_s)$ with $\omega_i \geq 0$, and $\mu > 0$. Differentiating the weighted ridge objective gives

$$\begin{aligned} \boldsymbol{S}_s &= \arg\min_{\boldsymbol{W}} \frac{1}{2}\|(\boldsymbol{W}\boldsymbol{K} - \boldsymbol{V})\boldsymbol{\Omega}^{1/2}\|_F^2 + \frac{\mu}{2}\|\boldsymbol{W}\|_F^2 \\ &= \boldsymbol{V}\boldsymbol{\Omega}\boldsymbol{K}^\top(\boldsymbol{K}\boldsymbol{\Omega}\boldsymbol{K}^\top + \mu\boldsymbol{I})^{-1}. \end{aligned} \tag{26}$$

Its recursive form starts from $\boldsymbol{S}_{-1} = 0$ and $\boldsymbol{R}_{-1} = \mu^{-1}\boldsymbol{I}$:

$$\begin{aligned} \boldsymbol{\ell}_i &= \frac{\omega_i \boldsymbol{R}_{i-1}\boldsymbol{k}_i}{1 + \omega_i \boldsymbol{k}_i^\top \boldsymbol{R}_{i-1}\boldsymbol{k}_i}, \\ \boldsymbol{S}_i &= \boldsymbol{S}_{i-1}(\boldsymbol{I} - \boldsymbol{k}_i\boldsymbol{\ell}_i^\top) + \boldsymbol{v}_i\boldsymbol{\ell}_i^\top, \qquad \boldsymbol{R}_i = \boldsymbol{R}_{i-1} - \boldsymbol{R}_{i-1}\boldsymbol{k}_i\boldsymbol{\ell}_i^\top. \end{aligned} \tag{27}$$

The rank-one inverse identity yields $\boldsymbol{R}_i = (\mu\boldsymbol{I} + \sum_{j\leq i}\omega_j\boldsymbol{k}_j\boldsymbol{k}_j^\top)^{-1}$. Substituting this into the ridge normal equations gives the update for $\boldsymbol{S}_i$. Hence $\boldsymbol{A}_i = \boldsymbol{I} - \boldsymbol{k}_i\boldsymbol{\ell}_i^\top$ and $\boldsymbol{b}_i = \boldsymbol{\ell}_i$, with $\boldsymbol{R}_i$ as control state. Our implementation evaluates the equivalent weighted Gram solve.

LaCT-inspired reader. Let $\boldsymbol{K}_i = [\boldsymbol{k}_0, \dots, \boldsymbol{k}_i]$, $\boldsymbol{V}_i = [\boldsymbol{v}_0, \dots, \boldsymbol{v}_i]$, and $\boldsymbol{E}_i = \mathrm{Diag}(\eta_0, \dots, \eta_i)$. A gradient step on the weighted prefix prediction error, followed by the implemented normalization, is

$$\begin{aligned} \overline{\boldsymbol{S}}_i &= \boldsymbol{S}_{i-1}(\boldsymbol{I} - \boldsymbol{K}_i\boldsymbol{E}_i\boldsymbol{K}_i^\top) + \boldsymbol{V}_i\boldsymbol{E}_i\boldsymbol{K}_i^\top, \\ \boldsymbol{S}_i &= \nu_i\overline{\boldsymbol{S}}_i, \qquad \nu_i = (\|\overline{\boldsymbol{S}}_i\|_F^2 + \varepsilon^2)^{-1/2}, \qquad \boldsymbol{S}_{-1} = 0. \end{aligned} \tag{28}$$

Because old values are written again, $\boldsymbol{S}_i$ alone is insufficient for a single-new-value update. Set $\boldsymbol{M}^- = [\boldsymbol{S}_{i-1}, \boldsymbol{V}_{<i}]$ and $\boldsymbol{G}_i = \boldsymbol{I} - \boldsymbol{K}_i\boldsymbol{E}_i\boldsymbol{K}_i^\top$. The augmented operators

$$\boldsymbol{A}_i = \begin{bmatrix} \nu_i\boldsymbol{G}_i & 0 & 0 \\ \nu_i\boldsymbol{E}_{<i}\boldsymbol{K}_{<i}^\top & \boldsymbol{I} & 0 \end{bmatrix}, \qquad \boldsymbol{b}_i^\top = [\nu_i\eta_i\boldsymbol{k}_i^\top, \boldsymbol{0}^\top, 1] \tag{29}$$

give $\boldsymbol{M}^+ = \boldsymbol{M}^-\boldsymbol{A}_i + \boldsymbol{v}_i\boldsymbol{b}_i^\top = [\boldsymbol{S}_i, \boldsymbol{V}_i]$ by direct multiplication. Prefix blocks are empty at $i = 0$. The scalar normalization preserves the value span but remains in the derivative graph. This reader and Mesa use different objectives and update schedules; neither mapping establishes a bound on language-model loss.

C VISIT-LEVEL READER IMPLEMENTATIONS

The two interfaces in Table 10 read at visit boundaries and initialize to the plain loop. Both use unprojected values, L2 normalization with an epsilon floor, and no learned Q/K/V projections.

C.1 CUMULATIVE STATES AND WHOLE-VISIT UPDATES

Write $\widetilde{\boldsymbol{h}}_{(r+1)K}$ for the output of visit r before its boundary combination. The paired visit update is

$$\boldsymbol{\Delta}_r = \widetilde{\boldsymbol{h}}_{(r+1)K} - \boldsymbol{h}_{rK} = \sum_{k=0}^{K-1} \boldsymbol{\delta}_{rK+k}. \tag{30}$$

The equality telescopes under within-visit addition, as used here; block-level reweighting would also enter the visit difference.

The state interface stores $\boldsymbol{h}_0$ and each visit output before its boundary blend. Denote these stored states by $\boldsymbol{v}_j$. Within each of eight heads,

$$\boldsymbol{k}_j = \mathrm{L2Norm}(\boldsymbol{v}_j), \qquad \boldsymbol{v}_j = \boldsymbol{v}_j, \qquad \boldsymbol{q} = \mathrm{L2Norm}(\widetilde{\boldsymbol{h}}_{(r+1)K}).$$

There are at most $R + 1$ sources. The current key and query coincide, so their inner product is one when the normalization floor is inactive. The memory output $\boldsymbol{o}_r$ is combined with the current proposal as

$$\boldsymbol{h}_{(r+1)K} = (1 - \zeta)\widetilde{\boldsymbol{h}}_{(r+1)K} + \zeta\boldsymbol{o}_r, \qquad \zeta_{\text{init}} = 0. \tag{31}$$

The update interface instead stores $\boldsymbol{\Delta}_0, \ldots, \boldsymbol{\Delta}_r$ and uses one full-width head, with

$$\boldsymbol{k}_j = \mathrm{L2Norm}(\boldsymbol{\Delta}_j), \qquad \boldsymbol{v}_j = \boldsymbol{\Delta}_j, \qquad \boldsymbol{q} = \mathrm{L2Norm}(\widetilde{\boldsymbol{h}}_{(r+1)K}).$$

It exposes at most R values and adds the input outside the memory read:

$$\boldsymbol{h}_{(r+1)K} = (1-\zeta)\widetilde{\boldsymbol{h}}_{(r+1)K} + \zeta(\boldsymbol{h}_0 + \boldsymbol{o}_r), \qquad \zeta_{\text{init}} = 0. \tag{32}$$

In both interfaces ζ is an unconstrained learned scalar. Zero blend recovers Plain, not Loopback's fixed-skip initialization. Changing the interface also changes head count, source count, gate shapes, and output blend; it is not an isolated storage ablation.

C.2 Gates and memory evaluation

In our adapters, gates are content-independent learned parameters indexed by junction, head, and, when required, channel. Key-addressed families write along $\boldsymbol{k}_j$; HGRN-2 writes the gate complement instead. For RWKV-7, the removal direction $\boldsymbol{\kappa}_j$ in Table 1 is the normalized visit update in the update interface and the normalized difference of adjacent stored states in the state interface. Mesa and the LaCT-inspired reader follow Section B.3, including the latter's prefix normalization.

The implementations evaluate short histories using the factorization $\boldsymbol{S}_s = \sum_i \boldsymbol{v}_i \boldsymbol{w}_{s,i}^\top$, where $\boldsymbol{w}_{s,i}$ is a write vector after subsequent transitions, or replay the required prefixes. These evaluate the stated reads without establishing an end-to-end recurrent speedup.

D Loopback implementation and derivations

D.1 Mixture replacement and initialization

After appending the newest update and applying anchored decay at visit ends (the boundary skip), write the current proposal as $\boldsymbol{h}_{s+1}^{\text{base}} = \boldsymbol{h}_0 + \sum_i \bar{c}_{s,i}\boldsymbol{\delta}_i$. The implementation reaches the target in Eq. (13) by adding only the coefficient difference:

$$\boldsymbol{h}_{s+1} = \boldsymbol{h}_{s+1}^{\text{base}} + \sum_i (c_{s,i} - \bar{c}_{s,i})\boldsymbol{\delta}_i. \tag{33}$$

The implementation uses skip parameter $\beta = 1 - \lambda$ and tracks the actual coefficients after prior reads to avoid double-counting contributions.

To make the initial correction vanish, target coefficients are evaluated as

$$\boldsymbol{c}_s = \boldsymbol{c}_s^0 + B_s \left[\mathrm{softmax}(\boldsymbol{z}_s + \log \boldsymbol{c}_s^0) - \mathrm{softmax}(\log \boldsymbol{c}_s^0)\right]. \tag{34}$$

For a positive unfloored prior this equals $B_s \,\mathrm{softmax}(\boldsymbol{z}_s + \log \boldsymbol{c}_s^0)$ in exact arithmetic. Zero queries cancel the bracket; causal induction then recovers the decay-only trajectory of Section 5.1. The first read has only one source and requires no query. The $(KR-1)d$ learned parameters total 11,520 at $K = 4$, $R = 4$, $d = 768$ (the 191M setting).

The actual log prior uses $\log \max(c_{s,i}^0, 10^{-8})$ in both softmax terms. If the floor activates, the centered form still preserves the total weight in exact arithmetic and cancels at zero queries, but its coefficients need not remain nonnegative. All reported configurations satisfy $\lambda^R > 10^{-8}$, so the floor is inactive.

D.2 Allocation gradient

With the message bank, prior, and total B fixed, write $\pi_i = c_i/B$ and $\boldsymbol{\mu} = \sum_j \pi_j \boldsymbol{\delta}_j$, so $\boldsymbol{h} = \boldsymbol{h}_0 + B\boldsymbol{\mu}$. The softmax derivative $\partial \pi_j / \partial z_i = \pi_j(\mathbf{1}_{i=j} - \pi_i)$ gives $\partial \boldsymbol{h}/\partial z_i = c_i(\boldsymbol{\delta}_i - \boldsymbol{\mu})$. Contraction with $\boldsymbol{g} = \nabla_{\boldsymbol{h}}\mathcal{L}$ yields

$$\frac{\partial \mathcal{L}}{\partial z_i} = c_i \boldsymbol{g}^\top(\boldsymbol{\delta}_i - \boldsymbol{\mu}), \qquad \sum_i \frac{\partial \mathcal{L}}{\partial z_i} = 0. \tag{35}$$

The sum vanishes because $\sum_i c_i \boldsymbol{\delta}_i = B\boldsymbol{\mu}$. This local gradient redistributes a fixed total according to each message's alignment with the downstream loss gradient. Actual score changes are coupled through the query; full training also changes the keys and messages.

D.3 Retention sensitivity

Fix the generated messages, scores z_i, and boundary ages b_i at one destination, with numerical floors inactive. Let $\eta = \log \lambda$ and define

$$B = \sum_i e^{\eta b_i}, \qquad \pi_i^0 = e^{\eta b_i}/B, \qquad \pi_i = \mathrm{softmax}_i(z_i + \eta b_i), \qquad \boldsymbol{\mu} = \sum_i \pi_i \boldsymbol{\delta}_i.$$

Writing $\bar{b}_0 = \sum_i \pi_i^0 b_i$ and $\bar{b}_\pi = \sum_i \pi_i b_i$, differentiation gives $\partial_\eta B = B\bar{b}_0$ and $\partial_\eta \pi_i = \pi_i(b_i - \bar{b}_\pi)$. Applying the product rule to $\boldsymbol{u} = B\boldsymbol{\mu}$ gives

$$\frac{\partial \boldsymbol{u}}{\partial \eta} = B\left[\bar{b}_0 \boldsymbol{\mu} + \sum_i \pi_i (b_i - \bar{b}_\pi)\boldsymbol{\delta}_i\right]. \tag{36}$$

The two terms change the total weight and the allocation across ages, respectively. This derivative holds the messages and scores fixed; it excludes their changes during training.

E Training and evaluation protocols

Table 4 lists the architecture, batch composition, and budget at every scale; the subsections below give the full 191M recipe and what changes at larger scales. The per-GPU sequence counts in Table 11 are these training micro-batches, and timing runs use the same micro-batch and accumulation.

Table 4: **Configuration at each scale.** Blocks and heads apply to Dense and to every looped arm; head width is 64 throughout. Sequences per optimizer step are the per-GPU micro-batch times gradient-accumulation steps, on 8 ranks, at 4,096 tokens per sequence. Peak learning rates are Dense / looped arms.

Scale	Width	Blocks	Heads	SwiGLU	Loopback	Seq./step	Micro×acc.	Steps	Tokens	LR ($\times 10^{-3}$)
191M	768	20	12	2,048	[2L → 4L (×4) → 2L]	128	16 × 1	7,280	3.82B	1 / 3
374M	1,024	24	16	2,816	[2L → 10L (×2) → 2L]	256	8 × 4	7,120	7.47B	2 / 2
778M	1,536	24	24	4,096	[2L → 8L (×2) → 6L]	512	8 × 8	7,440	15.60B	1 / 1.5
1.4B	2,048	25	32	5,632	[2L → 9L (×2) → 5L]	960	8 × 15	7,200	28.31B	1 / 1

E.1 Architecture and training protocol

Scale names use Dense's total parameters, which set the common 20-TPP budget; reported Transformer counts exclude embeddings and the vocabulary head. At 191M, all nine models train from scratch for 7,280 steps using Tables 5 and 6, differing only in topology, residual rule, and peak learning rate. An additional eight-block Dense control matches Plain's stored parameters (56.64M), not its twenty applications; under the same protocol and tokens, its loss is 2.7376 versus Plain's 2.6802. Budgets are rounded to a 20-step grid. During the final tenth of training, the learning-rate multiplier is $1 - \sqrt{u}$ for decay progress $u \in [0, 1]$. All results use the final checkpoint.

E.2 Residual rules and evaluation

Table 7 specifies our depth-wise adapters. Trunk weights are shared across visits; connections and queries are untied where the rule requires.

Held-out language modeling. Validation uses disjoint ClimbMix shard 3583. As in training, documents are tokenized with beginning-of-sequence markers, concatenated, and packed into 4,096-token sequences. All models use the first 1,024 sequences and final BF16 checkpoints. Loss averages next-token cross-entropy over 4,095 predicted positions per sequence after shifting labels. It is reported in nats; perplexity is its exponential.

Table 5: **191M architecture.** Looped topology: $[P\mathrm{L} \to K\mathrm{L}\,(\times R) \to C\mathrm{L}]$.

Setting	Value
Dense structure	Twenty independent blocks
Looped structure	$[2\mathrm{L} \to 4\mathrm{L}\,(\times 4) \to 2\mathrm{L}]$; eight distinct blocks, twenty applications
Attention	Full-sequence causal attention; 12 heads of width 64; no GQA
Hidden / feed-forward width	768 / 2,048; SwiGLU feed-forward layers
Normalization	Pre-RMSNorm; $\epsilon = 10^{-6}$
Position encoding	RoPE; $\theta = 10^{6}$
Context / vocabulary	4,096 tokens / 32,000 Llama tokens
Embedding / vocabulary head	Untied; omitted from reported Transformer counts
Trunk initialization	Linear and embedding weights: $\mathcal{N}(0, 0.02^2)$
Other trunk options	Bias-free attention projections; no QK normalization, logit softcap, or zero-initialized residual-output projections

Table 6: **Training recipe at the 20-TPP endpoint.** Only peak learning rates differ across arms.

Setting	Value
Training data	ClimbMix (Diao et al., 2026); training shards 0–239
Input construction	Online tokenization; concatenated documents packed into fixed 4,096-token sequences
Global batch	128 sequences; 8 GPU ranks $\times 16$ sequences per rank
Gradient accumulation	1; 524,288 input tokens per optimizer step
Training budget	7,280 steps; 3,816,816,640 input tokens
Optimizer	AdamW; $(\beta_1, \beta_2) = (0.9, 0.95)$; $\epsilon = 10^{-15}$
Weight decay / gradient clip	0 / global norm 1.0
Peak learning rate	Dense: 10^{-3}; all looped arms: 3×10^{-3}
Learning-rate schedule	Linear warmup: 128 steps; stable peak: 6,424 steps; square-root decay to zero: 728 steps
Hardware allocation	One node with eight GPU ranks for every arm
Mixed precision	BF16 parameter computation; FP32 gradient reductions
Execution	FSDP; no activation checkpointing or model compilation

Table 7: **Residual settings for the main nine-model comparison.** The stream dimension is four for every stream-mixing baseline.

Model	Residual configuration
Dense / Plain	Ordinary residual addition; unshared / shared trunk, respectively
mHC	One connection per visit; full stream mixing with 20 Sinkhorn iterations
HC	One connection per visit; dynamic four-stream mixing
Hyperloop	One connection per visit; diagonal residual transport; residual bias 0
iHC	One identity-carry Hy4 connection per middle-block application; magnitude 2; gate offset 10^{-6}; learned final stream readout
MHAR	Eight-head attention over the loop states; zero-initialized queries and blend gate
AttnRes	Per-sublayer placement: every sublayer attends over the token embedding and all earlier sublayer outputs, one zero-initialized pseudo-query per read
Loopback	Single-head attention over raw block updates; fixed $\beta = 0.75$; zero-initialized destination queries and geometric prior

Downstream tasks. The same final checkpoints are evaluated using lm-evaluation-harness 0.4.12 (Gao et al., 2021), zero-shot prompts, full evaluation sets, and 1,000 bootstrap iterations. HellaSwag, OpenBookQA, ARC-Easy, and ARC-Challenge use length-normalized accuracy; the other accuracy tasks use the harness's standard accuracy. MMLU is the harness aggregate over 57 subjects and counts as one task in the unweighted thirteen-task mean. WikiText word perplexity is reported separately and excluded from that mean.

Table 16 reports taskwise diagnostics. Bootstrap resampling measures variability over evaluation items, not over training runs.

E.3 374M COMPARISON

At width 1024, Dense has twenty-four blocks and looped models use $[2\text{L} \rightarrow 10\text{L}\,(\times 2) \rightarrow 2\text{L}]$, storing fourteen blocks for twenty-four applications. All arms receive 7.47B tokens in 7,120 steps of 1,048,576 tokens (256 sequences: eight per GPU with four accumulation steps) and follow the data, evaluation, and schedule conventions in Section E.1. All arms, including the memory readers of Table 10, use a peak learning rate of 2×10^{-3}. For the dense control, this rate was selected from $\{1, 1.5, 2\} \times 10^{-3}$ in a preliminary 5-TPP comparison; the upper two rates differed by about 0.001 nats there. A parameter-matched dense control with fourteen distinct blocks (179.86M Transformer parameters, the looped models' stored count) trained under the same protocol and tokens reaches loss 2.4833, higher than Plain's 2.4415.

E.4 778M SCALE-UP

At width 1536, Dense has twenty-four blocks, 24 attention heads, SwiGLU width 4096, and 777.9M total parameters (679.55M Transformer). Loopback uses $[2\text{L} \rightarrow 8\text{L}\,(\times 2) \rightarrow 6\text{L}]$: sixteen stored blocks, twenty-four applications, and 453.06M Transformer parameters (66.7% of Dense). Both receive 15.60B tokens in 7,440 steps of 2,097,152 tokens (512 sequences: eight per GPU with eight accumulation steps).

Data, evaluation, and schedule follow Section E.3. Dense uses a peak learning rate of 10^{-3}, the best of $\{0.5, 1, 2\} \times 10^{-3}$ for a 1.4B-parameter dense model in a preliminary 2-TPP comparison. Loopback uses a peak rate of 1.5×10^{-3}; in a preliminary 5-TPP comparison on $[2\text{L} \rightarrow 10\text{L}\,(\times 2) \rightarrow 2\text{L}]$, 2×10^{-3} changed its validation loss by 0.0005 nats.

E.5 1.4B SCALE-UP

At width 2048, Dense has twenty-five blocks, 32 attention heads, and 1,415.68M total parameters (1,284.61M Transformer). Loopback uses $[2\text{L} \rightarrow 9\text{L}\,(\times 2) \rightarrow 5\text{L}]$, $\lambda = 1/4$, and 822.19M Transformer parameters (64% of Dense), with the same twenty-five applications. Both receive 28.31B tokens in 7,200 steps of 3,932,160 tokens (960 sequences). Data, evaluation, and schedule follow Section E.3. These runs use H200 GPUs; losses are compared only within scale. Every model uses a peak learning rate of 10^{-3}: in preliminary 2-TPP comparisons at this scale, 10^{-3} was the best of $\{0.5, 1, 2\} \times 10^{-3}$ for the dense control and 2×10^{-3} raised Loopback's loss; Loopback's rate was not tuned further.

F SENSITIVITY AND MEMORY-READER RESULTS

F.1 DENSE LEARNING-RATE COMPARISON

Table 8: **Learning-rate selection for the dense control.** The unshared 20-block model uses the same machine and budget as Table 13. We select the best of four tested peak rates, 10^{-3}; looped models use their best tested rate, 3×10^{-3}.

Peak learning rate	5×10^{-4}	7×10^{-4}	10^{-3}	2×10^{-3}
ClimbMix loss	2.6666	2.6420	**2.6234**	2.6245

F.2 Decay strength

A sweep on a second machine under the 191M protocol of Table 3 (Dense: 2.6188) gives Loopback losses of 2.6195, 2.6158, 2.6259, and 2.6296 at $\lambda = 1/5$, $1/4$, $1/3$, and $1/2$, favoring $\lambda = 1/4$. With zero queries, $\lambda = 1/4$ gives 2.6245 and $\lambda = 1$ (Plain) 2.6798, so learned allocation gains 0.0087 nats there. These runs are not pooled with Table 3.

F.3 Decay controls

Two fixed rules at 191M keep one of decay's two effects. Recency only rescales decay's geometric weights to Plain's growing total $s + 1$; total only spreads decay's bounded total B_s evenly over the stored updates. Both use zero queries and the protocol of Table 3.

Table 9: **Decay controls at 191M.** Validation loss (↓), gain over Plain, and the share of decay's gain that each control recovers. All rows use zero queries.

Rule	Loss	Gain over Plain	Share of decay's gain
Plain ($\lambda = 1$)	2.6802	0.0000	0%
Recency only	2.6687	0.0115	20%
Total only	2.6350	0.0452	78%
Decay only ($\lambda = 1/4$)	2.6220	0.0582	100%

F.4 Initialization schemes

For Table 3, the default linear and embedding weights follow $\mathcal{N}(0, 0.02^2)$. The GPT-2 scheme (Radford et al., 2019) keeps this standard deviation except in each block's two residual-writing projections, the attention output and MLP down-projection, where weights follow $\mathcal{N}\big(0, (0.02/\sqrt{2L})^2\big)$ for $L = 20$. The PyTorch scheme uses the module defaults: Kaiming-uniform linear weights with $a = \sqrt{5}$ and $\mathcal{N}(0, 1)$ token embeddings. Residual-rule parameters keep their own initialization in every scheme; Loopback's queries start at zero. All runs retain the learning rates of Table 2: 10^{-3} for Dense and 3×10^{-3} for looped models, without retuning. Loopback has the lowest loss in every scheme and a smaller range than Dense or mHC, the strongest residual baseline in the main comparison. The default is also Dense's best scheme among those tested. This measures initialization sensitivity, not repeated-seed variability under a fixed protocol.

F.5 MEMORY-READER RESULTS

Table 10: **Memory readers at two scales.** ClimbMix validation loss (↓), with matched machine and budget within each scale (Table 2). Bold marks the best reader in each column and the best reference model at each scale. Updates and states use one and eight heads, respectively, with different output blends (Section C); only update readers are reported at 374M.

	191M		**374M**
Reader	Updates	States	Updates
Gated DeltaNet	**2.6489**	2.6685	2.4308
HGRN-2	2.6522	2.6619	**2.4256**
RWKV-7	2.6532	2.6791	2.4371
Mesa	2.6539	**2.6603**	2.4277
DeltaNet	2.6542	2.6664	2.4310
KDA	2.6543	2.6612	2.4290
Mamba-2	2.6544	2.6759	2.4308
LaCT-inspired	2.6558	2.6774	2.4308
GLA	2.6640	2.6773	2.4352
RetNet	2.6650	2.6827	2.4338
Linear attention	2.6697	2.6846	2.4346
Reference models			
Plain loop		2.6802	2.4415
Dense		2.6234	2.4198
Loopback		**2.6187**	**2.3971**

G COMPUTE AND EFFICIENCY MEASUREMENTS

FLOP accounting. Table 11 covers three scales. PyTorch's operator counter measures forward FLOPs on one 4,096-token bf16 sequence, including the vocabulary head. Scaled-dot-product attention substitutes for the uncounted flash kernel; uncounted elementwise work is added analytically. For a read at destination s over $s+1$ stored updates, the score and weighted-sum matmuls contribute $4(s+1)d$ FLOPs per token, excluding normalization, softmax, and mixture replacement. Anchored decay costs $3d$ per token at each visit end; AttnRes reads over up to $2L+1$ sublayer outputs at every one of its $2L+1$ reads; the stream rules cost $O(n^2 d)$ per connection for $n = 4$ streams.

Training measurements. Measurements use one eight-H100 node, FSDP, bf16 compute, and each scale's micro-batch. AttnRes and Loopback use fused reads measured in 160-step timing runs; other entries come from the training logs. Throughput uses median step time, and memory is the trainer's reported peak. Estimated hours divide the token budget by eight times the per-GPU throughput. Activations dominate sharded model state here, so fewer stored weights need not lower peak memory.

Inference measurements. One H100 runs each model eagerly with bf16 weights and no gradients; AttnRes and Loopback use fused reads. Prefill is one forward pass over a batch of eight 4,096-token sequences that keeps only the last position's logits and materializes no KV cache, so its measured peak covers weights and transient activations. For Loopback these include the depth history, which exists only within a forward pass and is never cached across decoding steps. Loopback's prefill throughput relative to Dense is 0.68, 0.72, and 0.81 at 191M, 374M, and 778M. Table 12 instead counts *persistent* memory analytically: all stored weights plus the KV cache, whose capacity is equal at matched executed depth. The two views differ by construction: at 374M the measured prefill peak is 3.0 GiB for Loopback versus 2.1 GiB for Dense, whereas the persistent total for the same eight sequences is 3.46 versus 3.70 GiB.

Table 11: **Compute and efficiency of the residual rules.** Parameters are Transformer-only; extra forward FLOPs are relative to Dense. Training throughput and peak memory are per GPU; hours are estimated from throughput. Prefill uses one H100.

Model	Params (M)	Extra FLOPs	Train tok/s/GPU	Hours per 20 TPP	Peak GiB/GPU	Prefill tok/s
191M, $[2\mathrm{L} \to 4\mathrm{L}\,(\times 4) \to 2\mathrm{L}]$**, 3.82B tokens, 16 sequences per GPU**						
Dense	141.59	–	222k	0.6	38	885k
Plain loop	56.64	+0.00%	226k	0.6	38	886k
mHC	56.93	+0.15%	169k	0.8	42	594k
HC	56.65	+0.07%	186k	0.7	42	671k
iHC	57.04	+0.17%	116k	1.1	67	370k
Hyperloop	56.79	+0.08%	186k	0.7	42	743k
MHAR	56.64	+0.01%	204k	0.6	41	810k
AttnRes	56.67	+0.45%	80k	1.7	50	509k
Loopback	56.65	+0.08%	167k	0.8	42	601k
374M, $[2\mathrm{L} \to 10\mathrm{L}\,(\times 2) \to 2\mathrm{L}]$**, 7.47B tokens, 8 sequences per GPU**						
Dense	308.33	–	128k	2.0	26	492k
Plain loop	179.86	+0.00%	130k	2.0	26	491k
mHC	180.06	+0.05%	115k	2.3	29	422k
HC	179.87	+0.02%	120k	2.2	30	439k
iHC	180.53	+0.15%	68k	3.8	52	220k
Hyperloop	179.96	+0.03%	120k	2.2	29	457k
MHAR	179.86	+0.00%	126k	2.1	27	480k
AttnRes	179.91	+0.46%	47k	5.5	40	294k
Loopback	179.88	+0.08%	98k	2.6	32	352k
778M, $[2\mathrm{L} \to 8\mathrm{L}\,(\times 2) \to 6\mathrm{L}]$**, 15.60B tokens, 8 sequences per GPU**						
Dense	679.55	–	75k	7.2	38	269k
Plain loop	453.04	+0.00%	76k	7.2	37	269k
mHC	453.33	+0.04%	68k	7.9	42	240k
Loopback	453.06	+0.04%	63k	8.6	44	217k

Table 12: **Persistent inference memory of Dense and Loopback.** Analytical bf16 storage for all weights, including embeddings and the vocabulary head, plus the KV cache at 4,096 tokens. Totals exclude temporary activations and workspaces; they are not measured peaks.

				KV cache per	Weights + KV (GiB)	
Scale	Model	Params (M)	Weights (GiB)	sequence (MiB)	1 sequence	8 sequences
191M	Dense	190.74	0.36	240	0.59	2.23
	Loopback	105.80	0.20	240	0.43	2.07
374M	Dense	373.87	0.70	384	1.07	3.70
	Loopback	245.42	0.46	384	0.83	3.46
778M	Dense	777.86	1.45	576	2.01	5.95
	Loopback	551.36	1.03	576	1.59	5.53
1.4B	Dense	1,415.68	2.64	800	3.42	8.89
	Loopback	953.26	1.78	800	2.56	8.03

The fused read. Each read uses one Triton kernel, streaming updates once per token tile while accumulating the scored and prior softmaxes together. Backward saves one score per token and update rather than normalized keys, then reloads each update and writes its gradient once. Tests compare the kernel with a direct PyTorch implementation of Eq. (13). In FP32, the output, the installed coefficients, and every gradient (query, updates through the value and key paths, proposal, input, and previous coefficients) agree to within 10^{-6} relative error, measured as the largest absolute difference over the largest reference magnitude, across widths 768–3,072, 2–20 stored updates, ragged token counts, and boundary and in-visit reads. With bf16 inputs, both implementations are compared with an FP64 evaluation of the same rounded inputs, and the fused error is no larger than the reference's. At zero queries the output and coefficients match the decay-only reference, which reduces to Plain only at $\lambda = 1$. An end-to-end comparison of the two implementations on a small model is bitwise at initialization and within 10^{-6} after perturbing the queries, forward and backward. Most remaining step-time overhead is in backward, as each update receives gradients from later reads. The kernel and tests will be released with the code.

H Complete downstream results

All tables use the evaluation protocol in Section E.2. Accuracy is in percent; ↓ marks lower-is-better metrics and bold marks the best value within a scale. The seven-task mean covers PIQA, HellaSwag, WinoGrande, OpenBookQA, ARC-Easy, ARC-Challenge, and LAMBADA; the thirteen-task mean uses all accuracy tasks. Both exclude WikiText perplexity. Transformer counts include adapters and final normalization, count shared weights once, and exclude embeddings and the vocabulary head. At 374M and 778M, Dense's taskwise bootstrap standard errors range from 0.4 to 3.0 accuracy points (RTE: 3.0; OpenBookQA: 2.1).

Table 13: **Complete downstream results at 191M.** Twenty block applications and 3.82B tokens; shared conventions are in Section H.

Benchmark	Dense	Plain	mHC	HC	iHC	Hyperloop	MHAR	AttnRes	**Loopback**
Topology	20	[2L → 4L (×4) → 2L] (all looped models)							
Transformer params (M)	141.59	56.64	56.93	56.65	57.04	56.79	56.64	56.67	56.65
ClimbMix loss ↓	2.6234	2.6802	2.6250	2.6449	2.6564	2.6489	2.6564	2.6647	**2.6187**
PIQA	65.45	64.91	65.29	64.58	65.34	64.47	64.91	65.29	**65.56**
HellaSwag	33.11	31.93	33.59	32.90	32.70	32.94	32.83	32.48	**33.62**
WinoGrande	51.46	51.30	51.62	51.85	52.64	48.70	49.72	**53.04**	51.62
OpenBookQA	31.60	29.80	31.20	29.00	**32.40**	32.20	31.00	30.60	31.00
CommonsenseQA	21.46	18.67	20.88	20.15	19.82	**21.62**	19.98	20.23	20.64
BoolQ	**62.20**	47.03	**62.20**	58.32	**62.20**	60.83	61.38	60.52	61.50
RACE	28.61	28.13	28.04	**29.47**	28.71	28.71	29.28	27.46	28.23
ARC-Easy	**47.18**	46.89	46.68	46.13	46.42	45.41	45.62	46.17	46.93
ARC-Challenge	**26.54**	26.45	25.26	25.94	24.66	26.11	26.11	26.19	**26.54**
MMLU (57 subjects)	23.61	**25.46**	24.42	24.20	24.41	24.13	23.69	25.09	23.78
ANLI-R1	29.80	31.80	29.90	**32.60**	31.10	31.30	31.10	31.10	30.90
RTE	49.82	48.01	53.43	53.43	**54.87**	52.71	51.62	51.99	53.79
LAMBADA	23.31	21.97	25.19	23.99	23.50	23.99	23.05	24.35	**25.89**
WikiText word ppl ↓	46.30	50.30	45.86	47.52	48.72	47.87	48.30	48.31	**45.74**
Mean accuracy (13 tasks)	38.01	36.33	38.28	37.89	38.37	37.93	37.72	38.04	**38.46**

Table 14: **Complete downstream results at 374M.** Twenty-four block applications and 7.47B tokens; settings are in Section E.3.

Benchmark	Dense	Plain	mHC	HC	iHC	Hyperloop	MHAR	AttnRes	**Loopback**
Topology	24	$[2L \to 10L\,(\times 2) \to 2L]$ (all looped models)							
Transformer params (M)	308.33	179.86	180.06	179.87	180.53	179.96	179.86	179.91	179.88
ClimbMix loss ↓	2.4198	2.4415	2.4062	2.4249	2.4217	2.4179	2.4223	2.4601	**2.3971**
PIQA	68.44	68.93	67.08	68.93	**69.31**	68.82	68.66	68.50	69.04
HellaSwag	39.17	38.88	40.32	39.52	39.59	39.33	39.46	37.85	**40.62**
WinoGrande	51.07	51.22	52.80	50.59	49.88	52.25	52.25	51.93	**53.28**
OpenBookQA	32.00	**33.60**	33.00	32.00	33.20	33.00	32.20	**33.60**	33.00
CommonsenseQA	20.15	20.31	**20.97**	18.76	19.16	20.15	19.16	18.43	20.48
BoolQ	55.50	58.90	59.88	55.23	55.17	61.59	58.93	52.54	**61.71**
RACE	29.57	29.95	31.20	31.00	30.14	30.72	30.43	30.53	**32.34**
ARC-Easy	52.74	51.43	**54.92**	54.50	52.48	53.91	53.37	51.64	52.78
ARC-Challenge	28.84	29.52	**30.12**	29.78	28.16	29.52	27.73	28.41	28.16
MMLU (57 subjects)	24.46	**26.46**	23.13	26.28	24.51	25.11	24.59	25.73	24.80
ANLI-R1	32.70	31.40	33.30	**33.90**	31.90	31.60	30.20	31.20	33.20
RTE	**55.23**	53.79	51.26	54.51	49.10	54.87	52.35	53.43	50.18
LAMBADA	30.68	29.98	30.95	30.70	30.84	31.61	31.81	28.27	**33.01**
WikiText word ppl ↓	33.71	34.88	32.77	33.94	33.78	33.54	33.83	35.46	**32.41**
Mean accuracy (13 tasks)	40.04	40.34	40.69	40.44	39.50	40.96	40.09	39.39	**40.97**

Table 15: **Complete downstream results at larger scales.** The 778M/1.4B settings use 24/25 block applications and 15.60B/28.31B tokens, respectively (Sections E.4 and E.5).

	778M		1.4B	
Benchmark	Dense	**Loopback**	Dense	**Loopback**
Topology	24 blocks	$[2L \to 8L\,(\times 2) \to 6L]$	25 blocks	$[2L \to 9L\,(\times 2) \to 5L]$
Transformer params (M)	679.55	453.06	1,284.61	822.19
ClimbMix loss ↓	2.2521	**2.2334**	**2.1266**	2.1376
PIQA	**72.20**	71.76	**73.94**	73.61
HellaSwag	47.49	**48.82**	**54.65**	54.32
WinoGrande	52.49	**53.91**	**55.80**	53.35
OpenBookQA	**35.20**	35.00	38.00	**38.80**
CommonsenseQA	18.51	**21.21**	19.66	**19.90**
BoolQ	**57.49**	55.20	61.19	**63.55**
RACE	31.10	**31.77**	**32.92**	**32.92**
ARC-Easy	59.47	**60.52**	**64.77**	64.56
ARC-Challenge	31.74	**33.11**	**36.60**	35.92
MMLU (57 subjects)	**25.59**	25.22	**25.72**	**25.72**
ANLI-R1	33.20	**34.70**	32.50	**33.10**
RTE	54.51	**54.87**	51.99	**55.96**
LAMBADA	36.21	**37.96**	**41.20**	40.40
WikiText word ppl ↓	25.80	**25.03**	**21.15**	21.38
Mean (7 main tasks)	47.83	**48.73**	**52.14**	51.57
Mean accuracy (13 tasks)	42.71	**43.39**	45.30	**45.55**

H.1 TASKWISE DOWNSTREAM DIAGNOSTICS

Table 16: **Taskwise variation across the nine models.** Spread is maximum minus minimum accuracy; s.e. is the mean bootstrap standard error, both in accuracy points. Ratio is spread/s.e.; ρ is the Spearman correlation of accuracy with validation loss (negative favors lower loss). These evaluation-item diagnostics are not pairwise significance tests or training-run uncertainty estimates.

	191M, [2L → 4L (×4) → 2L]				374M, [2L → 10L (×2) → 2L]			
Task	Spread	S.e.	Ratio	ρ	Spread	S.e.	Ratio	ρ
PIQA	1.1	1.1	1.0	−0.45	2.2	1.1	2.1	+0.03
HellaSwag	1.7	0.5	3.6	−0.96	2.8	0.5	5.7	−0.73
WinoGrande	4.3	1.4	3.1	+0.03	3.4	1.4	2.4	−0.56
OpenBookQA	3.4	2.1	1.6	−0.25	1.6	2.1	0.8	+0.32
CommonsenseQA	2.9	1.2	2.6	−0.66	2.5	1.1	2.2	−0.72
BoolQ	15.2	0.9	17.8	−0.57	9.2	0.9	10.6	−0.77
RACE	2.0	1.4	1.4	−0.12	2.8	1.4	1.9	−0.48
ARC-Easy	1.8	1.0	1.7	−0.39	3.5	1.0	3.4	−0.53
ARC-Challenge	1.9	1.3	1.5	−0.21	2.4	1.3	1.8	−0.04
MMLU	1.9	0.4	5.1	+0.64	3.3	0.4	9.1	+0.63
ANLI-R1	2.8	1.5	1.9	+0.58	3.7	1.5	2.5	−0.50
RTE	6.9	3.0	2.3	−0.39	6.1	3.0	2.0	+0.23
LAMBADA	3.9	0.6	6.6	−0.55	4.7	0.6	7.4	−0.75

I LIMITATIONS

Our experiments focus on from-scratch language-model pretraining on ClimbMix at the evaluated scales and token budgets. The balance between parameter savings and quality can depend on scale, topology, and training recipe. Broader data mixtures, longer training, and post-training settings remain to be explored.

We compare complete residual designs, including their memory sources and adapter placement. Further controlled comparisons could separate the effects of these choices more precisely.

Loopback preserves executed depth while reducing stored parameters. Its explicit update history introduces memory traffic and temporary activations, so parameter savings do not necessarily translate into lower latency or peak memory (Section 6.5). More compact history representations and adaptive loop depths are possible extensions.