THE SURPRISING EFFECTIVENESS OF SHARED MEMORY IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers apply the same layers several times per token, adding compute to improve quality without more parameters. Each recursion, however, writes its own key-value cache, so memory still grows with compute. Inference-time techniques can shrink this cache at a cost in quality. We pretrain looped language models to share memory: only the first recursion writes a cache, and later recursions read it while keeping a short window of their own. Surprisingly, we find that sharing memory does not cost quality and instead improves it. At 150M–1B parameters, our Looped Prediction Transformer (LPT) and its hybrid variant set a new quality–memory frontier for looped models: with five recursions, the hybrid lowers validation NLL on FineWeb-Edu by 0.104–0.114 nats relative to a same-size standard Transformer while using 76–79% less context memory. Through an extensive analysis, we investigate why memory sharing helps. Shared and local memory develop different representations, and later recursions attend mostly to the shared memory, which also acts as a gradient highway to the first recursion.

1 INTRODUCTION

Looped Transformers (LT) repeatedly apply the same layers several times before predicting the next token, adding computation without adding parameters (Dehghani et al., 2019; Zhu et al., 2025). In principle, they trade time for memory, matching the compute of a larger model with far fewer parameters. In standard formulations, however, context memory still grows with recursion depth, as each iteration keeps a separate key-value (KV) cache (Geiping et al., 2025; Zhu et al., 2025). At long

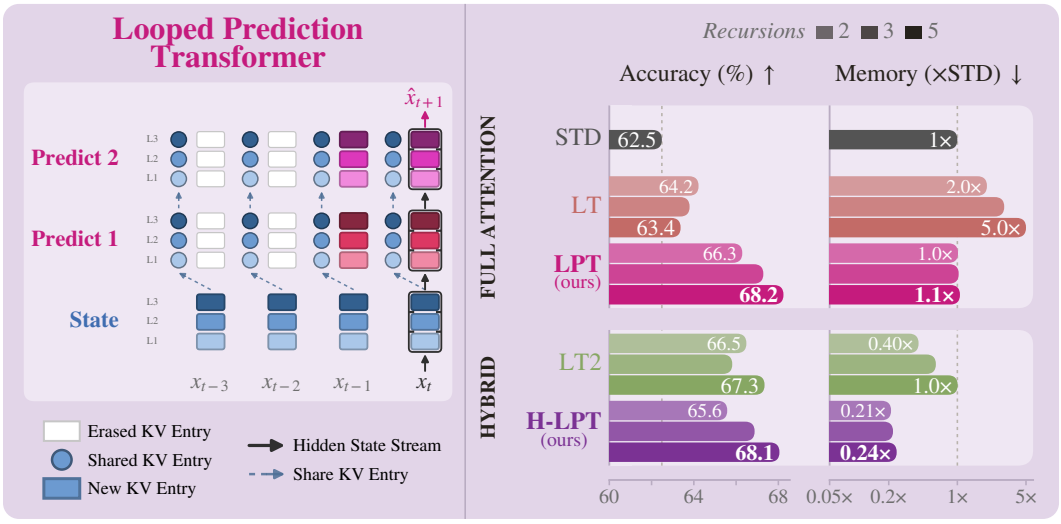


Figure 1: **Shared memory not only saves memory, it also improves quality. Left:** in our Looped Prediction Transformer (LPT), only the first recursion writes a full-context KV cache, later recursions read it and keep a short window. **Right:** five-task accuracy and context memory vs. a parameter-matched standard Transformer (STD), 1B parameters, two to five recursions. LT: looped Transformer with a full cache per recursion. LT2: its gated DeltaNet hybrid. H-LPT: LT2 with shared memory.

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.

contexts, this context memory can grow larger than the weights themselves, undoing the very reason to use a looped model.

Sharing caches across recursions during decoding can save substantial memory with little quality loss (Geiping et al., 2025; Zhu et al., 2025). So far, this reuse has been studied mainly as a way to save memory. Yet it also shows that memory written by one recursion can serve the others, even in models never trained to share it. This raises a question: *if sharing can work untrained, what can it do when models are trained for it?* To answer it, we train looped models with memory sharing from the start, investigating it as an architectural inductive bias.

We study a simple recipe for *Memory Sharing* in looped Transformers (Figure 1, left). At each full-attention layer, later recursions attend to both their own KV entries and the cache written by that layer in the first recursion, within the same attention operation. To further save memory, we truncate the cache of every recursion after the first to a short sliding window. The resulting Looped Prediction Transformer (LPT) keeps LT's sequential recurrence and uses context memory close to a standard (non-looped) Transformer's. Applied to the full-attention layers of a hybrid looped Transformer (Deng et al., 2026), the same rule gives the Hybrid Looped Prediction Transformer (H-LPT). We pretrain all models from scratch at 150M to 1B parameters.

Surprisingly, Memory Sharing improves quality over full-cache LT even when each later recursion's own cache covers less than 2% of the context. Its advantage grows with depth: LT stops improving in NLL after three recursions, while LPT improves through five. Across sizes, LPT lowers validation NLL by up to 0.099 nats and raises downstream accuracy by up to 4.8 percentage points over a matched-depth LT, with up to $4.7\times$ less context memory and nearly the same inference FLOPs. Sharing also improves the hybrid baseline: at five recursions, H-LPT outperforms a same-size standard Transformer in both language modeling and downstream accuracy while using 76–79$\%$ less context memory, moving the quality–memory frontier for looped models (Figure 2).

These results raise a further question: *why does memory sharing improve quality?* We find that later recursions rely heavily on the shared memory. LPT outperforms LT at every window size, even when later recursions keep no cache of their own, and later recursions attend mostly to shared memory even when their own cache covers the full context. Shared and local memory also develop distinct key representations, suggesting that they serve different purposes. Reading shared memory sends gradients directly to the first recursion, helping it learn what to store. This gradient highway bypasses the intermediate recursions: in LT, the magnitude of the gradient reaching the first recursion falls sharply with depth, while LPT recovers much of it through the shared memory. When we block this path, the model still reads shared memory but no longer matches LPT's quality and stability.

2 Looped Transformers

A standard LT shares its weights across recursions, while each recursion computes its own hidden states and KV entries. We use Ouro's full-stack formulation (Zhu et al., 2025), which builds on earlier recurrent-depth architectures (Dehghani et al., 2019; Geiping et al., 2025).

Let $\mathcal{M}^L = \mathcal{T}_\theta^{(L)} \circ \cdots \circ \mathcal{T}_\theta^{(1)}$ denote the shared stack of L Transformer blocks. For a sequence of length T and hidden dimension d, let $h_{\text{in}}^{(t)} \in \mathbb{R}^{T\times d}$ and $h^{(t)}$ be the residual streams entering and leaving recursion t. The stack is applied R times:

$$\begin{aligned} h_{\text{in}}^{(1)} &= \text{emb}(\mathbf{x}), \\ h^{(t)} &= \mathcal{M}^L\big(h_{\text{in}}^{(t)}\big), \\ h_{\text{in}}^{(t+1)} &= \mathcal{N}_{\text{rec}}\big(h^{(t)}\big), \\ \text{logits}^{(t)} &= \text{lm_head}\big(\mathcal{N}_{\text{out}}(h^{(t)})\big), \qquad t = 1, \ldots, R. \end{aligned} \tag{1}$$

Here $\mathcal{N}_{\text{rec}}$ and $\mathcal{N}_{\text{out}}$ are separately parameterized RMSNorms. Both act on $h^{(t)}$: the recurrent norm prepares the next recursion's input, while the readout norm feeds the language-modeling head. The effective depth is $L \cdot R$, with the parameters of a single L-layer stack.

Each layer computes its keys and values from the hidden states it receives at the current recursion. For a fixed layer, let $\mathcal{C}^{(t)}$ denote the cache written by recursion t, with entry $(k_j^{(t)}, v_j^{(t)})$ at position j.

We suppress the layer index throughout: for vanilla LT, each layer reads only its own entries. A query at position i reads the causal prefix of its recursion's cache, giving the attention set

$$\mathcal{A}^{(t)}_{\text{LT}}(i) = \mathcal{C}^{(t)}_{\leq i} = \{(k^{(t)}_j, v^{(t)}_j) : j \leq i\}. \tag{2}$$

At inference, each layer therefore retains R full-context caches, one per recursion, and context memory grows linearly with recursion depth even though the parameter count stays fixed. Some of these caches can be dropped or merged during decoding (Geiping et al., 2025; Zhu et al., 2025), but the model is still trained with a separate cache for every recursion.

Hybrid architectures reduce this memory cost by replacing most full-attention layers with linear attention, whose recurrent state does not grow with context length (Katharopoulos et al., 2020; Schlag et al., 2021; Yang et al., 2025). LT2 (Deng et al., 2026) loops such a stack, using Gated DeltaNet layers and one full-attention layer per five. This reduces the number of full-attention caches, but each remaining layer still needs a separate full-context cache at every recursion.

3 Looping with Shared Memory

We study *Memory Sharing*, in which later recursions read the memory written by the first recursion, as an inductive bias for looped models. The first recursion runs as in LT and writes a full-context cache $\mathcal{C}^{(1)}$ at each layer, and every later recursion attends to this cache together with its own KV entries, using a single softmax.

To also keep looping memory-efficient, we restrict each later recursion's own cache to a short window. The first recursion therefore acts as a memory phase that builds the model's only full-context memory, while later recursions use it to refine the prediction. We refer to the first recursion as the *state recursion* and to the later ones as *prediction recursions*, and call the resulting model the Looped Prediction Transformer (LPT; Figure 1). Like LT, LPT applies the same L-layer stack R times and passes the final output to the language-modeling head (Equation 1).

Let ω be the number of preceding positions kept in each prediction recursion's window, and write $\mathcal{C}^{(t)}_{[i-\omega,i]}$ for the entries from positions $i-\omega$ through i, including the current entry. The state recursion reads its own cache, as in LT (Equation 2). A prediction recursion reads the shared memory together with its own window. The entries read by a query at position i in recursion t are

$$\mathcal{A}^{(t)}_{\text{LPT}}(i) = \begin{cases} \mathcal{C}^{(1)}_{\leq i}, & t = 1, \\ \mathcal{C}^{(1)}_{\leq i} \cup \mathcal{C}^{(t)}_{[i-\omega,i]}, & t = 2, \ldots, R. \end{cases} \tag{3}$$

In prediction recursions, the two sets are attended to using a single softmax.

Setting $\omega = T$ gives every prediction recursion a full-context cache of its own, as in LT, and leaves access to shared memory as the only difference between the two models; we use this setting in Section 6 to separate the effect of sharing from that of the short window.

The same rule extends to hybrid looped architectures: H-LPT applies Equation 3 to the full-attention layers of LT2 (Deng et al., 2026) and leaves its gated linear layers and its gated residual connection between recursions unchanged.

Efficiency and Costs In training, we implement Equation 3 as a FlexAttention (Dong et al., 2025) block mask over the concatenated shared and local entries, which skips the fully masked blocks, so each query attends over the shared prefix and its ω window entries only (Appendix A).

During decoding, each prediction recursion keeps a sliding buffer of at most ω preceding entries per layer; its current entry participates in attention before entering the buffer. In cached token positions across L full-attention layers, the inference-time KV cache sizes are

$$M^{\text{LT}}_{\text{KV}} = L\,R\,T, \qquad M^{\text{LPT}}_{\text{KV}} = L\,[T + (R-1)\omega]\,, \tag{4}$$

where the common head dimensions and datatype size are omitted. LPT uses the same parameters and recursion count as a matched-depth LT, but performs slightly more attention because each prediction recursion also reads its local window. At a full context of $T = 4{,}096$ with $\omega = 64$ and five recursions, these windows increase the number of attended KV entries per generated token by only $(R-1)\omega/(RT) = 1.25\%$, while reducing the full-attention cache relative to LT by $4.7\times$.

Table 1: **LPT and H-LPT reach lower NLL than their looped baselines with much smaller caches.** Main results across S, M, and L: validation NLL, mean five-task accuracy (%), and context memory and FLOPs relative to same-size STD. In the STD rows, $R = 1$ is the standard Transformer, and $R > 1$ is DEEPSTD. Elsewhere, R counts model passes. **Bold** and underline mark the best and second-best within each size and column group; our rows are shaded.

		$R=1$		$R=2$				$R=3$				$R=5$			
Size	Model	NLL ↓	Acc. ↑	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓
	STD	2.824	51.5	2.838	50.2	1.50	**1.19**	2.850	50.7	1.88	**1.31**	2.874	50.4	2.50	**1.54**
	LOOPIE			2.809	51.6	2.00	1.83	2.804	51.9	3.00	2.66	2.808	52.1	5.00	4.32
	LT			2.784	51.5	2.00	1.83	2.772	52.1	3.00	2.66	2.773	52.4	5.00	4.32
S	PLT			2.763	**52.7**	1.02	1.84	2.744	52.9	1.03	2.67	2.740	53.5	1.06	4.34
	LT2			2.759	51.9	0.35	1.29	2.740	52.5	0.52	1.86	2.720	53.7	0.87	2.98
	LPT			**2.756**	51.9	1.02	1.84	**2.726**	**54.0**	1.03	2.67	**2.702**	**54.9**	1.06	4.34
	H-LPT			2.757	52.4	**0.18**	1.29	2.735	53.1	**0.19**	1.86	2.710	54.6	**0.21**	2.98
	STD	2.625	56.1	2.657	55.5	1.44	**1.13**	2.673	55.4	1.78	**1.24**	2.701	54.4	2.34	**1.43**
	LOOPIE			2.617	56.9	2.00	1.92	2.621	55.8	3.00	2.84	2.620	56.6	5.00	4.69
	LT			2.588	57.5	2.00	1.92	2.576	59.4	3.00	2.84	2.592	57.9	5.00	4.69
M	PLT			2.565	59.1	1.02	1.93	2.540	60.0	1.03	2.85	2.538	60.5	1.06	4.71
	LT2			2.551	59.1	0.35	1.43	2.537	59.5	0.52	2.10	2.521	60.2	0.87	3.45
	LPT			2.558	**59.2**	1.02	1.93	**2.530**	60.2	1.03	2.85	2.512	**61.6**	1.06	4.70
	H-LPT			**2.550**	58.5	**0.18**	1.43	**2.530**	**60.3**	**0.19**	2.10	**2.511**	61.4	**0.21**	3.45
	STD	2.429	62.5	2.453	62.5	1.45	**1.14**	2.481	61.3	1.73	**1.17**	2.507	61.4	2.25	**1.32**
	LOOPIE			2.421	63.5	2.00	1.95	2.423	63.9	3.00	2.91	2.506	61.0	5.00	4.81
	LT			2.416	64.2	2.00	1.95	2.408	63.8	3.00	2.91	2.433	63.4	5.00	4.81
L	PLT			2.393	64.8	1.02	1.96	2.375	66.8	1.03	2.92	2.360	66.4	1.06	4.83
	LT2			2.360	**66.5**	0.40	1.53	2.348	65.8	0.60	2.27	2.334	67.3	1.01	3.75
	LPT			2.375	66.3	1.02	1.96	2.347	**67.3**	1.03	2.92	2.334	**68.2**	1.06	4.83
	H-LPT			**2.359**	65.6	**0.21**	1.53	**2.343**	66.9	**0.22**	2.27	**2.325**	68.1	**0.24**	3.75

4 EXPERIMENTAL SETUP

Baselines To study memory sharing, we compare LPT with LT and H-LPT with LT2. We also compare with the Parallel Loop Transformer (PLT) (Wu et al., 2025), which parallelizes recursions and uses gated dual attention over the first recursion's shared cache and each later recursion's local cache. Other baselines are a standard Transformer (STD), LOOPIE, which repeats each layer before advancing to the next (Gao et al., 2026), and DEEPSTD, a deeper, narrower Transformer without weight sharing that matches the looped models in effective depth and approximately in parameters.

Architecture All models use a decoder-only Transformer backbone with RMSNorm, SwiGLU, rotary positional embeddings, and tied input and output embeddings. We train S, M, and L models with approximately 150M, 450M, and 1B parameters; the main comparisons cover all three sizes, and most ablations and analyses use S and M. The context length is 4,096. Unless otherwise specified, LPT and H-LPT use a prediction window of $\omega = 64$. We study two, three, and five recursions. This range includes the two loop steps of LOOPIE (Gao et al., 2026) and surrounds the four recursions of Ouro and LT2 (Zhu et al., 2025; Deng et al., 2026).[1] All models use nanoGPT's default optimizer settings (Karpathy, 2022). Model dimensions, normalization, and training details are in Appendix A.

Data and Training We pretrain on FineWeb-Edu (Penedo et al., 2024) using 20B tokens for S and M, and 50B for L, roughly 2–7× the Chinchilla compute-optimal token budgets (Hoffmann et al., 2022). The main tables report one run per configuration, with the same initialization seed and data order across models. At S, we also train three more seeds per model and confirm that seed variation is far below the gaps we report (Appendix B).

We study two training objectives. *Final loss* computes the next-token NLL at the final recursion only. *Adaptive loss* follows Ouro (Zhu et al., 2025), weighting per-recursion NLLs with a learned per-token

[1] Nine recursions bring instability in several model families and limited accuracy gains, consistent with the loss spikes Ouro reports at eight (Zhu et al., 2025). We report these experiments in Appendix E.

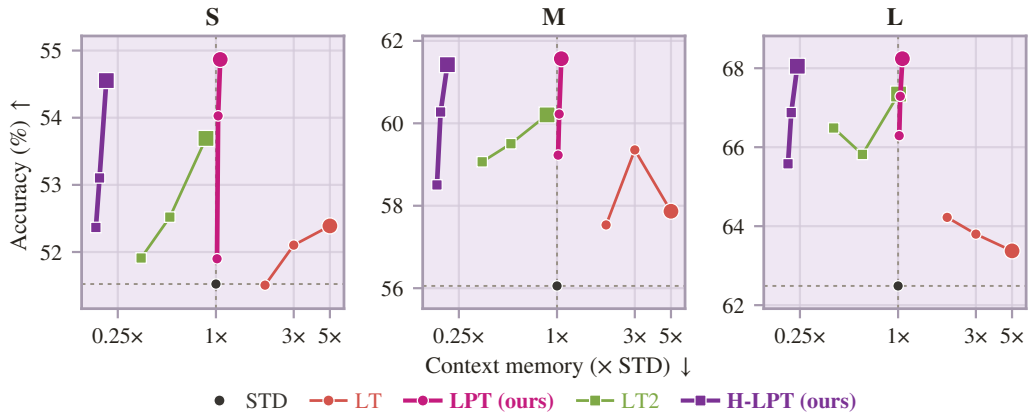


Figure 2: **LPT and H-LPT reach higher accuracy with much smaller caches than the looped baselines.** Mean five-task accuracy vs. context memory at each size. Curves follow two, three, and five recursions, with larger markers at five. Dashed lines mark single-pass STD.

halting distribution and adding KL regularization. Unless otherwise specified, LT uses adaptive loss and all other models use final loss.

Evaluation We report NLL on 10M held-out FineWeb-Edu tokens and mean zero-shot accuracy on ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and LAMBADA (Paperno et al., 2016), using the LM Evaluation Harness (Gao et al., 2023). We compute context memory and decoding FLOPs per generated token analytically at a 4,096-token context, relative to the same size STD. Context memory includes the KV cache and GDN's recurrent state.

5 Results

Memory Sharing Improves Quality Restricting the memory written by later recursions could hurt prediction. Surprisingly, our Memory Sharing approach improves quality. At matched size and recursion count, LPT lowers validation NLL by 0.028–0.099 nats and raises accuracy by 0.4–4.8 percentage points over LT, with 2.0–4.7× less KV memory and under 1% more FLOPs (Table 1). Measured on an H100, the memory savings are the same, so LPT fits 2.0–4.8× as many sequences in a batch as LT (Appendix F).

The NLL gain grows with recursion count (Table 1): LT stops improving beyond three recursions at every size, while LPT continues to improve through five. At comparable cache sizes, LPT also reaches lower NLL than PLT's parallel formulation. At low recursion counts, the hybrid LT2 improves on LT while using less memory and compute. Another factor that could explain this gap, and the instability of LT, is Ouro's normalization, which we use for both LT and LPT. Other normalizations make LT more robust, but none matches LPT (Appendix C). We keep Ouro's normalization for LT because it is the one in the original paper (Zhu et al., 2025). LT2's remaining full-attention caches still grow with depth, approaching STD's context memory at five recursions. H-LPT keeps this memory at 0.18–0.24× STD's across recursion counts, while also improving quality. At five recursions, it lowers NLL by 0.009–0.010 and raises accuracy by 0.8–1.2 points over LT2, with 4.1–4.2× less context memory and about 0.1% more inference FLOPs. Both hybrids improve with depth, but H-LPT reaches lower NLL at every size and recursion count.

Our seed sweep at S (Appendix B) confirms that these gaps are well beyond noise level: across three more seeds of every model, NLL varies by at most 0.005 nats, and paired tests confirm the gaps over LT and LT2 at $p < 0.05$.

More Quality from the Same Parameters Both memory-sharing models improve on STD with little additional memory, or much less for H-LPT (Figure 2). At five recursions, LPT lowers NLL by

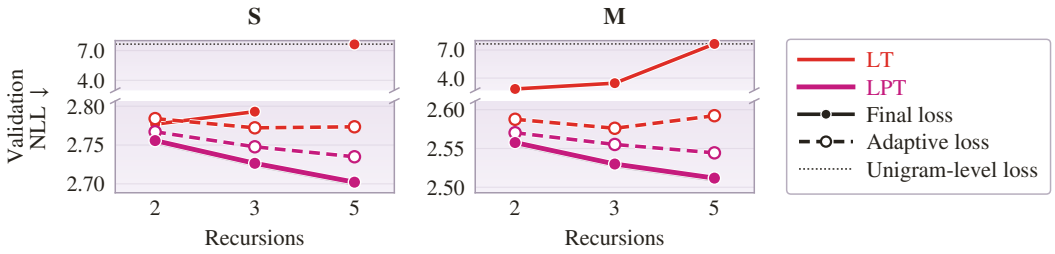


Figure 3: **LPT trains best with final loss, while LT consistently worsens with more recursions without adaptive loss.** Validation NLL (↓). Filled markers are final loss, hollow ones adaptive loss.

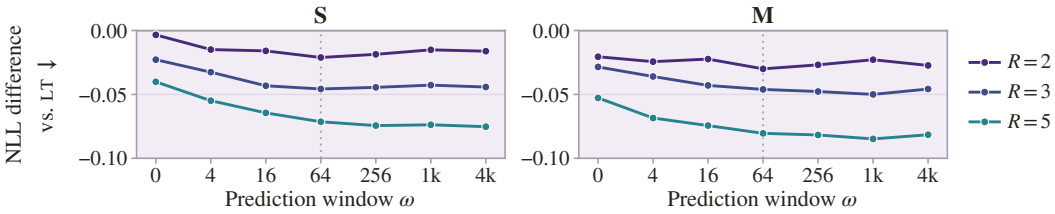


Figure 4: **A short prediction window is enough.** Validation NLL of LPT relative to LT (nats, ↓) vs. window size, LT at its better objective per depth. The dotted line marks $\omega = 64$.

0.095–0.122 and raises accuracy by 3.4–5.7 points with 6.25% more context memory. H-LPT lowers NLL by 0.104–0.114 and raises accuracy by 3.1–5.6 points with 76–79% less memory. This smaller cache lets H-LPT fit 3.6–5.0× as many sequences as STD on one GPU. With three to five recursions, H-LPT lowers NLL by 0.086–0.114 while beating or staying close to STD's throughput (1.4–1.5× at three, 81–88% at five, Appendix F). Both models' accuracy gains grow from two to five recursions at every size. LPT also outperforms LOOPIE and DEEPSTD, neither of which improves consistently with depth.

> **Takeaway 1.** Through Memory Sharing, LPT and especially H-LPT improves the quality–memory frontier for looped models (Figure 2).

6 Why Memory Sharing Helps

We use S- and M-size models with two, three, and five recursions to study why this inductive bias improves quality (Appendix D).

Memory Sharing Stabilizes Deep Recursion We compare final and adaptive loss in LT and LPT (Figure 3). Without intermediate supervision, adding recursions consistently worsens LT's NLL, and the model collapses to unigram-level loss at five recursions. LPT stays stable, and final loss lowers its NLL by 0.011–0.033 compared with adaptive loss. Its intermediate recursions can refine the hidden state without having to predict early.

> **Takeaway 2.** LPT stays stable and trains best with final loss.

Prediction Recursion Needs Local Memory We train LPT with prediction windows ranging from $\omega = 0$ to the full context of 4,096 tokens (Figure 4). LPT outperforms LT at every window size, across both sizes and all tested depths. This includes $\omega = 0$, where prediction recursions keep no local history and read shared memory alongside only their current KV entry. At the other extreme, $\omega = 4{,}096$, LT and LPT differ only in access to shared memory. LPT still lowers NLL by 0.016–0.082 nats relative to LT with the better of its two training objectives at each depth, close to the 0.021–0.080 gain at $\omega = 64$. Overall, window size has a modest effect on quality, and what really matters is the access to the shared memory. To balance quality and efficiency, we adopt $\omega = 64$ as the default for our memory sharing models.

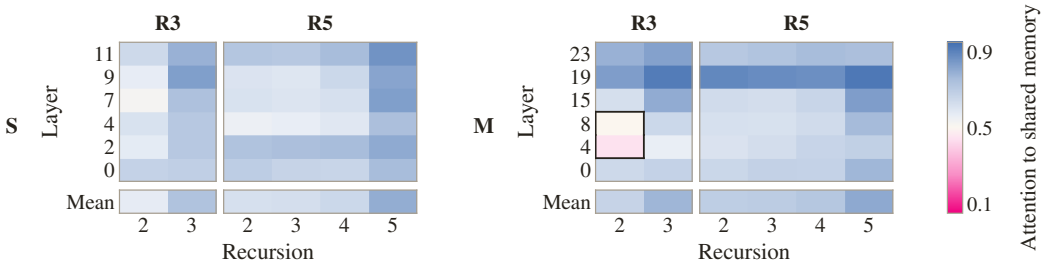


Figure 5: **Prediction recursions attend mostly to shared memory.** Attention mass assigned to shared KV entries in LPT, averaged over queries and heads. For clarity, only six layers are displayed. The bottom row averages all layers.

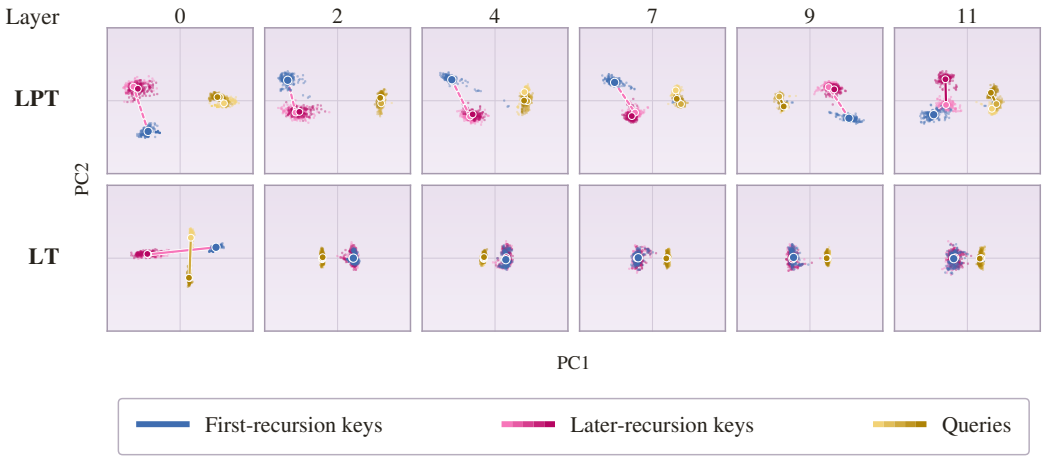


Figure 6: **Shared and local keys separate in LPT.** Joint query/key PCA, fitted separately for each panel. Dots are tokens and large circles their means; darker shades mark later recursions.

Takeaway 3. Shared memory improves quality across prediction-window sizes, even when later recursions keep no local history.

Later Recursions Attend More to Shared Memory We next measure how much attention later recursions assign to shared memory. We evaluate each model on the same 4,096-token validation sequence and average the attention assigned to shared keys over queries and heads (Figure 5 and Appendix D). Shared memory receives most attention at every prediction recursion in both sizes. Its share is 57–71% in earlier prediction recursions and rises to 73–80% in the final recursion.[2]

Takeaway 4. Later recursions attend more to shared memory than to their own caches, with a stronger preference when getting closer to the prediction.

Shared and Local Memory Develop Different Representations Shared keys serve every later recursion, whereas local keys serve only the recursion that writes them. To see whether these roles are reflected in their representations, we compare S-size LT and LPT with three recursions, both trained with final loss, the only tested setting with at least three recursions where LT trains with final loss without a substantial drop in quality (Figure 3). We evaluate both models on the same 4,096-token validation sequence and extract queries and keys before rotary encoding. We concatenate the attention heads and fit principal component analysis (PCA) jointly to queries and keys across tokens and recursions per model and layer (Appendix D).

[2]Additional comparisons in the appendix show that shared memory still receives most attention with full prediction caches, while adaptive loss makes its use more uniform across recursions (Figure 10).

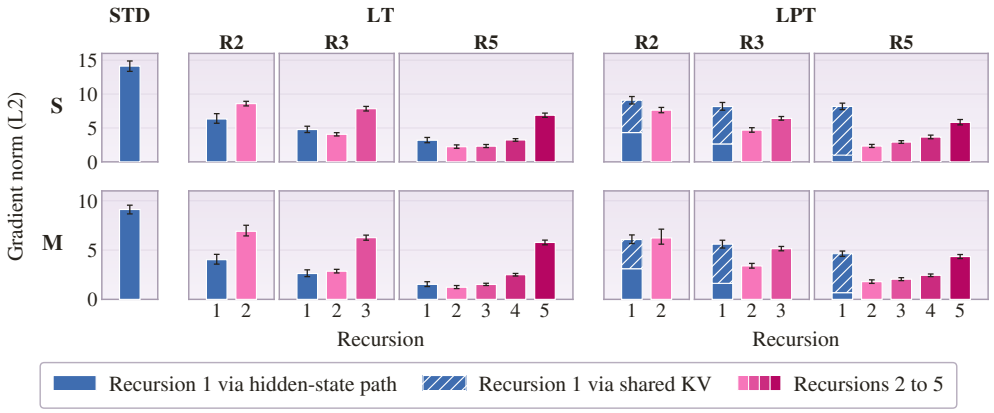


Figure 7: **Shared memory restores much of the gradient reaching the first recursion.** Mean gradient norm contributed by each recursion, with single-pass STD as reference. The first LPT bar is split into shared-KV (hatched) and hidden-state projections. LT is trained with adaptive loss, the others with final loss. Whiskers are 95% sequence-bootstrap intervals.

Table 2: **Detaching shared-cache gradients hurts learning.** Validation NLL (↓) at S and M. Bold marks the best per column.

	S			M		
Model	$R = 2$	$R = 3$	$R = 5$	$R = 2$	$R = 3$	$R = 5$
LT (final loss)	2.777	2.793	7.623	2.840	3.458	7.647
LPT (detached KV)	2.793	2.782	3.503	2.630	7.599	7.331
LPT	**2.756**	**2.726**	**2.702**	**2.558**	**2.530**	**2.512**

Shared and local keys consistently form distinct clusters in LPT, whereas keys from different recursions overlap more in LT (Figure 6). In the appendix, we show similar separation in LPT across S and M at three and five recursions (Figure 9). We also show that with full prediction caches, the clusters overlap more but still remain distinguishable, even though shared and local keys cover the same positions (Figure 8). Prior work attributes the small cost of cache reuse to the similarity of keys and values across recursions (Bae et al., 2025), as we see in LT. With Memory Sharing, shared and local keys are instead clearly different. This fits the freedom that final loss gives: recursions need not all produce prediction-ready states, so the first can specialize in building memory and the later ones in using it. This specialization may help explain the quality gains, similar to how Monea et al. (2026) show that Transformers benefit from assigning memory construction and next-token prediction to separate residual streams.

> **Takeaway 5.** Shared and local memory diverge in their local representations, so sharing is not reliant on recursions writing similar memories.

Shared Memory Is a Gradient Highway Reading shared memory gives the final loss a direct gradient path to the first recursion, bypassing the intermediate recursions. We evaluate the STD, LT, and LPT models of Table 1 on the same 32 validation sequences of 4,096 tokens and backpropagate the final loss from 32 fixed positions in each. We measure each recursion's contribution to the gradient on the Transformer block weights, and split the first recursion's contribution into its hidden-state and shared-KV paths by repeating the backward pass with shared KV detached (Appendix D).

In LT, the first-recursion gradient norm falls sharply with depth. LPT recovers much of the signal magnitude relative to the single-pass STD reference, with a much smaller decline as depth increases (Figure 7). At five recursions, most of this gradient comes through the shared KV cache. The same holds for LPT with adaptive loss or full prediction caches (Figure 11).

To test whether this highway helps learning, we train LPT from scratch with shared KV detached at later reads. The forward computation and hidden-state gradient path are unchanged. NLL rises in both sizes and at every depth, and deeper detached models often fail to train (Table 2).

Takeaway 6. The memory sharing mechanism creates a gradient highway from the final loss to the first recursion that facilitates learning.

7 Related Work

Separating Memory from Prediction In a standard Transformer, each hidden state does two jobs. It predicts the next token, and it writes the keys and values that later positions read. Several architectures separate these jobs across token streams, as in the two-stream attention of XLNet (Yang et al., 2019). The State-Prediction Separation Transformer (Monea et al., 2026) gives memory construction and next-token prediction separate residual streams and improves language modeling, and Langford et al. (2026) study a variant of this split. A second family separates them across layers. The first layers write one cache once, and the remaining layers only read it (Sun et al., 2024; 2026; DeepSeek-AI, 2026; Capps, 2026). Instead, LPT decouples memory from prediction across the recursions of a looped Transformer: the first recursion writes the model's only full-context memory, and every later recursion reads it while it predicts, keeping just a short window of its own. Looped models thus offer a third place for the split, with no extra stream or layer, and our analysis finds evidence that the recursions learn these roles (Section 6).

Looped Transformers and Memory Sharing Looped Transformers apply the same layers several times (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025), and recent work differs in which layers repeat and how recursions connect (Gao et al., 2026; Wang et al., 2026a; Huang et al., 2026; Mozer et al., 2026; Wang et al., 2026b; Deng et al., 2026). In standard formulations, each recursion writes its own KV cache, so context memory grows with depth (Geiping et al., 2025; Zhu et al., 2025). Sharing a cache across recursions, like sharing it across layers (Brandon et al., 2024), has been studied as a way to save memory. Without retraining, Huginn (Geiping et al., 2025) reuses caches cyclically with little quality loss, and Ouro (Zhu et al., 2025) can keep only the final or an averaged cache. Keeping only the first recursion's cache sharply reduces Ouro's quality, which suggests that shared memory has to be trained. Trained variants reuse the first recursion's keys and values at a slight cost in quality (Bae et al., 2025), compute them once in a separate prelude (Capps, 2026), or learn gated or compressed caches (Vendrell et al., 2026; O'Neill & Reid, 2026). PLT is the closest design. It shares the first recursion's cache with gated local caches and runs recursions in parallel (Wu et al., 2025), while LPT keeps them sequential and reads both caches in one softmax. All of these treat sharing as a cost to contain. We instead treat it as an inductive bias that separates memory from prediction and show that it improves quality (Section 5 and Section 6).

8 Discussion

We investigate memory sharing as an architectural inductive bias for looped models, and find that, in both full attention and hybrid models, it not only cuts context memory but also improves quality. Our analyses suggest that sharing helps recursions learn different roles. The first recursion builds memory that later recursions use to refine the prediction. Those reads also give the final loss a direct gradient path to the first recursion. The different representations of shared and local memory support this division of roles, and cutting this gradient path hurts learning. Shared memory changes how later recursions read information and how the first recursion learns to write it.

Like other looped Transformers, LPT is not compute-optimal: it trades time for quality, but unlike LT, its memory barely grows with depth. At five recursions it costs 4.3–$4.8\times$ the decoding FLOPs of STD, about the same as LT, and it spends the same number of recursions on every token. Choosing that number per token is future work. All our comparisons are parameter-matched, including DEEPSTD, and we do not train a standard Transformer matched in decoding FLOPs instead, which at five recursions would have several times the parameters. We train up to 1B parameters on one corpus with $4{,}096$-token contexts and do not test long-context tasks or, given our limited pretraining, very advanced tasks. Our analyses use S and M models, since not every configuration they need was trained at L, and only the detach ablation looks at training dynamics.

Overall, our findings suggest that shared memory is a useful inductive bias for looped models, helping them make better use of computation within a limited memory budget.

AI USE STATEMENT

We used generative AI tools to polish the writing and as a coding assistant for analysis and plotting scripts. All AI-generated text and code were reviewed and edited by the authors, who take full responsibility for the content of this paper.

ETHICS STATEMENT

This work pretrains research-scale language models, from 150M to 1B parameters, on FineWeb-Edu (Penedo et al., 2024), a public, filtered educational web corpus. It does not involve human subjects. We do not identify societal impacts specific to LPT beyond those of language-model pretraining in general.

REPRODUCIBILITY STATEMENT

Section 4 specifies the baselines, the data, the training objectives, and the evaluation protocol. Appendix A gives the model dimensions, the attention implementation, the data processing, and the optimizer settings, Appendix B the seed protocol and the statistical tests, and Appendix D the setup of every analysis. The pretraining corpus is public. Our code will be released upon publication.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8b08bbf8b420faa6eeb4020720582ec7-Abstract-Conference.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/9e23d020c18e4c40d81c6a0fc7a46f68-Abstract-Conference.html.

Chad A. Capps. CART: Context-anchored recurrent transformer – a parameter-efficient architecture with learned stability, 2026. URL https://arxiv.org/abs/2606.01495.

Chen Chen and Lai Wei. Post-LayerNorm is back: Stable, ExpressivE, and deep, 2026. URL https://arxiv.org/abs/2601.19895.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 Reasoning Challenge, 2018. URL https://arxiv.org/abs/1803.05457.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

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

Chunyuan Deng, Yizhe Zhang, Rui-Jie Zhu, Yuanyuan Xu, Jiarui Liu, T. S. Eugene Ng, and Hanjie Chen. LT2: Linear-time looped transformers, 2026. URL https://arxiv.org/abs/2605.20670.

Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, and Jie Tang. CogView: Mastering text-to-image generation via transformers. In *Advances in Neural Information Processing Systems*, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/a4d92e2cd541fca87e4620aba658316d-Abstract.html.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. FlexAttention: A programming model for generating fused attention variants. In *Proceedings of Machine Learning and Systems*, 2025. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/61a9278dfef5f871b5e472389f8d6fa1-Abstract-Conference.html.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo, version v0.4.0, December 2023. URL https://zenodo.org/records/10256836.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the Loopies!, 2026. URL https://arxiv.org/abs/2607.16051.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

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, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

Benhao Huang, Chufan Shi, Junlin Chen, Shicheng Wen, Zhengzhong Liu, Eric Xing, and Xuezhe Ma. Towards looped models done right—Part I: Topology, input injection, recurrent-state design. Institute of Foundation Models technical blog, 2026. URL https://ifm-research.notion.site/3ade511912ec8128987dfeb7a5580043. Accessed September 16, 2026.

Andrej Karpathy. nanoGPT. GitHub repository, commit 325be85, 2022. URL https://github.com/karpathy/nanoGPT.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

John Langford, Nathan Godey, Giovanni Monea, Yoav Artzi, Harry Dong, Ying Fan, Gustavo de Rosa, and Zheng Zhan. Almost free state prediction separation, 2026. URL https://arxiv.org/abs/2609.03807.

Giovanni Monea, Nathan Godey, Kianté Brantley, and Yoav Artzi. The state-prediction separation hypothesis. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2607.01218.

Michael C. Mozer, Shoaib Ahmed Siddiqui, Danny Sawyer, Sunny Sanyal, and Rosanne Liu. Recirculation, 2026. URL https://arxiv.org/abs/2608.17981.

James O'Neill and Fergal Reid. Looped latent attention: Cross-loop KV compression for looped transformers, 2026. URL https://arxiv.org/abs/2607.15456.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016. URL https://aclanthology.org/P16-1144/.

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. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 30811–30849, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/370df50ccfdf8bde18f8f9c2d9151bda-Abstract-Datasets_and_Benchmarks_Track.html.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Noam Shazeer. GLU variants improve Transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

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://proceedings.neurips.cc/paper_files/paper/2024/hash/0df38cd13520747e1e64e5b123a78ef8-Abstract-Conference.html.

Yutao Sun, Li Dong, Tianzhu Ye, Shaohan Huang, Jianyong Wang, and Furu Wei. Universal YOCO for efficient depth scaling, 2026. URL https://arxiv.org/abs/2604.01220.

Victor Conchello Vendrell, Arnau Padrés Masdemont, Niccolò Grillo, Jordi Ros-Giralt, Arash Behboodi, and Fabio Valerio Massoli. Memory-efficient looped transformer: Decoupling compute from memory in looped language models, 2026. URL https://arxiv.org/abs/2605.07721.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. SMELT: Scaling laws for compute-matched MoE looped transformers, 2026a. URL https://arxiv.org/abs/2609.01343.

Xi Wang, Ziyang Cai, Zheng Zhan, Harry Dong, Ying Fan, Gustavo de Rosa, Tim Pearce, and John Langford. Full-bandwidth transformer, 2026b. URL https://arxiv.org/abs/2608.08888.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017. URL https://aclanthology.org/W17-4413/.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, Siyuan Qiao, and Xingyan Bin. Parallel loop transformer for efficient test-time computation scaling, 2025. URL https://arxiv.org/abs/2510.24824.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10524–10533, 2020. URL https://proceedings.mlr.press/v119/xiong20b.html.

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

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, pp. 29687–29707, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019. URL https://aclanthology.org/P19-1472/.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

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 Architecture and Training Details

Architecture Table 3 lists the base model dimensions. The decoder-only backbone uses RMSNorm pre-normalization (Zhang & Sennrich, 2019), SwiGLU feed-forward networks (Shazeer, 2020), no biases in linear layers, and tied input and output embeddings. Full-attention layers use rotary positional embeddings (Su et al., 2024).

Table 3: **Base model configurations.** Looped models reuse the listed L-layer stack.

	S	M	L
Layers (L)	12	24	36
Hidden size (d)	768	1,024	1,280
Heads	12	16	20
FFN size	3,072	4,096	5,120
Parameters	151.9M	454.2M	1.01B

LT and LPT use Ouro's normalization and recurrent transition (Zhu et al., 2025). LT2 and H-LPT use the Full+GDN layer pattern and gating setup of Deng et al. (2026), with one full-attention layer per five layers. The full-attention layers read the complete causal context in LT2. In H-LPT, later recursions read the first recursion's full-context cache and their own local prediction window, as in Equation 3.

PLT uses the same block, recurrent, and readout normalization as LT and LPT. Its recurrent transition adds the input embedding to the previous recursion's normalized output shifted by one token.

Attention Implementation During training, each prediction recursion concatenates its keys and values with the shared keys and values along the sequence dimension, separately at each full-attention layer. A block mask implements Equation 3, allowing full causal attention over shared entries and causal sliding-window attention over local entries. FlexAttention (Dong et al., 2025) skips fully masked blocks.

Data Processing We tokenize with the GPT-2 tokenizer (Radford et al., 2019) and pack documents into $4{,}096$-token sequences, inserting an end-of-sequence token between documents. Attention is masked across document boundaries, and boundary tokens are excluded from the language-modeling loss.

Optimization We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and gradient clipping at 1.0. The global batch contains 96 sequences of $4{,}096$ tokens, and training uses bfloat16 mixed precision. For L-size models, AdamW optimizer state is sharded across DDP ranks (ZeRO-1). The learning rate increases linearly to 6×10^{-4} over the first 50M tokens, stays constant, then decreases linearly to 6×10^{-5} over the final 10% of training. These are nanoGPT's default optimizer settings, used unchanged for every model, size, and objective: we did not tune the learning rate or the warmup for any model, and we did not test whether a lower peak learning rate or a longer warmup changes how final-loss LT trains at depth (Figure 3).

For adaptive loss, we use Ouro's adaptive-exit objective (Zhu et al., 2025). A learned per-token halting distribution weights the losses across recursions, with a KL regularization weight of 0.1. Final loss applies the language-modeling loss only at the last recursion.

B Seed Variance and Statistical Tests

Retraining every configuration at every size with several seeds is prohibitive at pretraining cost, so we run the check at S. Each S-size model of Table 1 is trained with four seeds: the run reported in the table and three more, each changing the initialization and the data order together. All other settings stay at their main-table values. Table 4 lists the mean and 95% confidence interval (Student-t) of the final validation NLL for every model and recursion count. Table 5 tests each gap of interest with a one-sided paired t-test on the same-seed differences, with the alternative that the first model has lower NLL. The largest standard deviation across seeds is 0.005 nats, against gaps of 0.026–0.068 nats between LPT and LT, and every difference keeps its sign at every seed.

Table 4: **Seed variance at S.** Final validation NLL, mean $\pm$ 95% confidence interval (Student-t) over four seeds. In the STD row, $R = 1$ is the standard Transformer and $R > 1$ the deeper, narrower parameter-matched Transformer. SPS is the two-recursion baseline of Appendix E.

Model	$R = 1$	$R = 2$	$R = 3$	$R = 5$
STD	2.824 ± 0.002	2.835 ± 0.003	2.849 ± 0.004	2.870 ± 0.008
SPS		2.780 ± 0.005		
LOOPIE		2.808 ± 0.002	2.808 ± 0.005	2.809 ± 0.001
LT		2.783 ± 0.002	2.773 ± 0.003	2.772 ± 0.002
PLT		2.762 ± 0.003	2.744 ± 0.002	2.738 ± 0.002
LT2		2.758 ± 0.001	2.739 ± 0.001	2.719 ± 0.001
LPT		2.757 ± 0.005	2.727 ± 0.003	2.704 ± 0.003
H-LPT		2.756 ± 0.002	2.735 ± 0.000	2.710 ± 0.001

Table 5: **Paired tests at S.** Same-seed NLL gap of the first model over the second, mean $\pm$ 95% confidence interval over four seeds, the gap in units of the standard deviation of the same-seed differences, and the one-sided paired t-test for the first model having lower NLL.

Comparison	R	Gap (NLL)	Gap/SD	p
LPT vs. LT	2	-0.026 ± 0.004	11	1.1×10^{-4}
	3	-0.046 ± 0.003	26	7.7×10^{-6}
	5	-0.068 ± 0.004	30	4.9×10^{-6}
H-LPT vs. LT2	2	-0.002 ± 0.001	4	1.6×10^{-3}
	3	-0.004 ± 0.001	5	1.0×10^{-3}
	5	-0.009 ± 0.001	10	1.4×10^{-4}

C Normalization between Recursions

Our LT and LPT use the Ouro paper's normalization (Zhu et al., 2025). Sandwich norms normalize the output of every sublayer before the residual addition, and the recurrent norm $\mathcal{N}_{\text{rec}}$ normalizes each recursion's output before the next recursion (Equation 1). With final loss, this LT degrades or diverges at depth (Figure 3), so the main tables train it with adaptive loss. We did not tune the normalization of either model. To test whether Ouro's normalization holds LT back, we also train S and M LT with final loss and five other normalizations, each shared across recursions:

- *Pre-norm only* (Xiong et al., 2020) removes both, so $h_{\text{in}}^{(t+1)} = h^{(t)}$ and each block keeps only its pre-norms.
- *Sandwich only* keeps the sandwich norms (Ding et al., 2021) and removes $\mathcal{N}_{\text{rec}}$.
- *Input norm* removes the sandwich norms and keeps $\mathcal{N}_{\text{rec}}$, which it also applies to the embeddings.
- *Update norm* removes the sandwich norms and normalizes only each recursion's update, so the carried state stays unnormalized: $h_{\text{in}}^{(t+1)} = h_{\text{in}}^{(t)} + \mathcal{N}\big(h^{(t)} - h_{\text{in}}^{(t)}\big)$.
- *KEEL* removes the sandwich norms and applies the highway residual of Chen & Wei (2026) across recursions: $h_{\text{in}}^{(t+1)} = \mathcal{N}_2\big(R\, h_{\text{in}}^{(t)} + \mathcal{M}^L(\mathcal{N}_1(h_{\text{in}}^{(t)}))\big)$.

Here $\mathcal{N}$, $\mathcal{N}_1$, and $\mathcal{N}_2$ are RMSNorms. Data, seed, batch, and learning-rate schedule match the main runs. LPT keeps Ouro's normalization and final loss.

At every size and depth in Table 6, LPT with Ouro's normalization reaches lower NLL than the best LT, and the best LT reaches lower NLL than our main LT. The alternatives are also more robust: every finished run trains with final loss at every size and depth, while Ouro's normalization degrades or diverges. No single normalization is best for LT: update norm wins three settings and KEEL the other three. Choosing the best one lowers LT's NLL by 0.008–0.048 nats relative to our main LT. LPT still leads the best LT by 0.020–0.026 nats at S and by 0.005–0.035 nats at M. The smallest gap, 0.005 nats at M with two recursions, is within the seed variation we measure at S (Appendix B). At S, every alternative keeps improving through five recursions, unlike our main LT. At M, the best LT stops improving after three recursions.

Table 6: **LPT outperforms LT with the best normalization we tried.** Validation NLL (↓) after 20B tokens. **Bold** marks the best per column and underline the best LT. Values above 7 are unigram-level loss. A dash marks a run that did not finish.

			S			M		
Model	Normalization	Loss	$R = 2$	$R = 3$	$R = 5$	$R = 2$	$R = 3$	$R = 5$
LT	Ouro (main)	Adaptive	2.784	2.772	2.773	2.588	2.576	2.592
	Ouro	Final	2.777	2.793	7.623	2.840	3.458	7.647
	Pre-norm only	Final	2.791	2.773	2.759	2.587	2.570	2.553
	Sandwich only	Final	2.788	2.763	2.753	–	–	–
	Input norm	Final	2.780	2.760	2.742	2.570	2.557	2.568
	Update norm	Final	2.776	2.752	2.732	2.567	2.546	2.549
	KEEL	Final	2.780	2.757	2.725	2.563	2.549	2.547
LPT	Ouro	Final	**2.756**	**2.726**	**2.702**	**2.558**	**2.530**	**2.512**

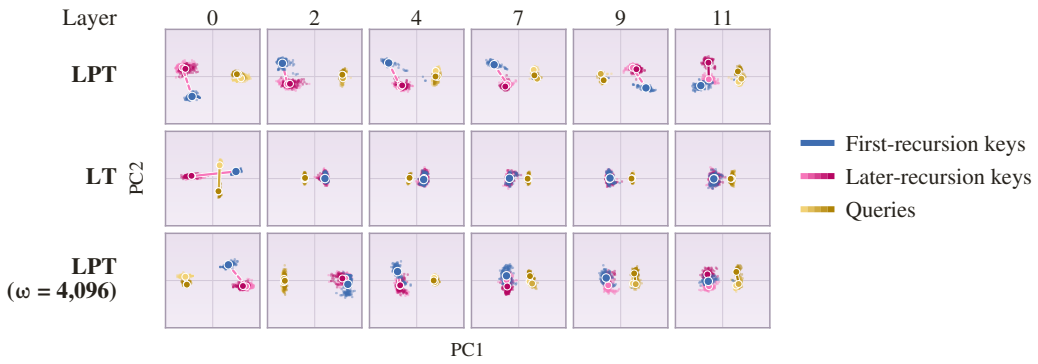


Figure 8: **Shared and local keys remain distinguishable with full prediction caches.** Expanded comparison for Figure 6, including $\omega = 4{,}096$.

Although this comparison favors LT, LPT still performs best. LT gets the best of six normalizations, chosen on the same validation NLL we report, while LPT keeps Ouro's. The same search for LPT would include Ouro's normalization, so it could only keep or lower LPT's NLL. Since normalization is not the focus of our investigation, we keep Ouro's for the main LT, as in the original paper.

D Additional Analysis Details

Attention Representations For Figure 6, we use S-size models with three recursions and final loss. We evaluate each model on the same 4,096-token validation sequence. At each layer, we concatenate the attention heads into one query vector and one key vector per token, before rotary encoding. We pool these vectors across all tokens and recursions, subtract their common mean, and fit PCA by singular value decomposition, separately for each model and layer. We do not normalize the vectors or rescale their features. All recursions within a panel therefore share one basis. Axes are not aligned across models or layers.

We display the first two principal components. For readability, the panels show one token in 24, while PCA and the large centroid markers use all tokens. Lines connect centroids across recursions, separately for queries and keys. Axis limits are shared across layers within each model, with equal scaling on both axes.

Figure 8 adds LPT with full prediction caches. Figure 9 compares LPT across sizes and recursion counts, matching layers by relative depth. Figure 10 compares attention to shared memory across window sizes and training objectives. Each layer's attention mass is averaged over queries and heads. We display six layers at matched relative depths, but compute the mean over all 12 layers for S and all 24 for M.

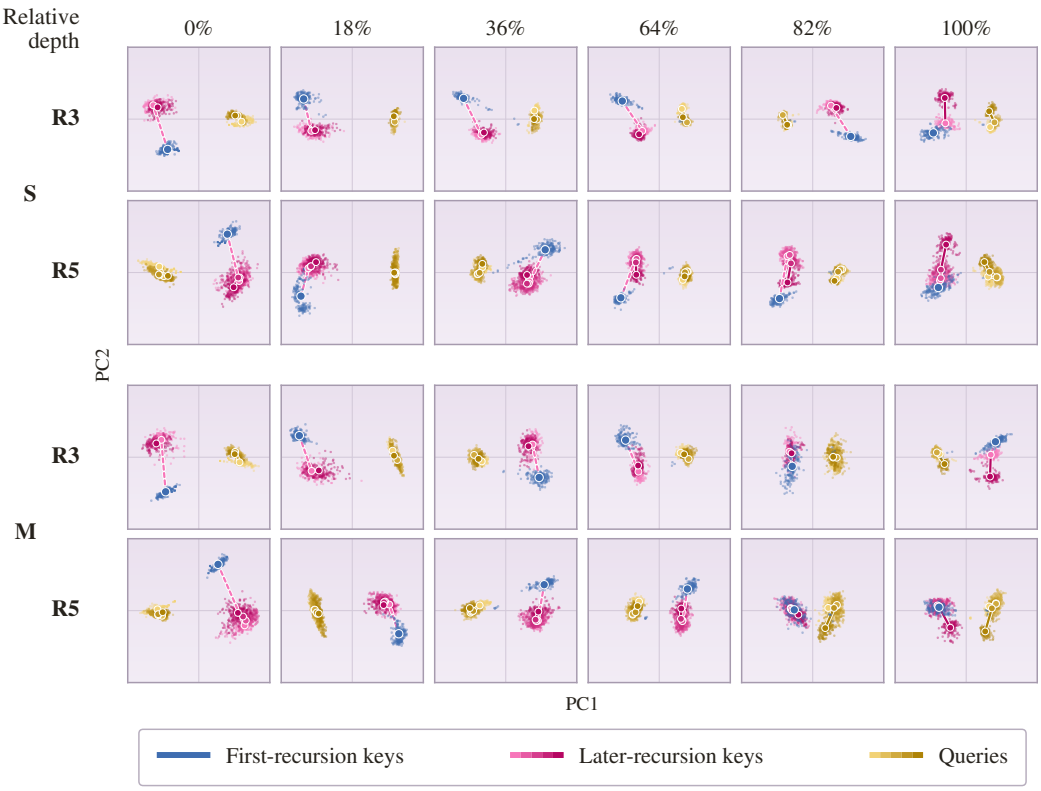


Figure 9: **Query/key geometry across LPT models.** S and M models at three and five recursions, with final loss and $\omega = 64$. Columns match relative layer depth.

Gradient Feedback through Shared Memory We differentiate the mean cross-entropy of the final recursion at 32 sampled positions in each of 32 validation sequences of length 4,096. The sequence contents and loss positions are identical across conditions. Figure 7 uses the main-comparison checkpoints: LT trained with adaptive loss and LPT with final loss. STD provides a single-pass reference, measured through its native forward pass on the same data. Figure 11 adds LPT with full prediction caches and LPT trained with adaptive loss. Every probe differentiates the loss at the final recursion, regardless of the training objective. We measure gradients on Transformer block parameters, excluding embeddings, readout, recurrent normalization, and the halting gate. Each model retains its trained normalization.

For the backward computation, we give each recursion an independent parameter copy with the same values. This leaves the forward pass unchanged and isolates each recursion's contribution to the shared-parameter gradient. Detaching shared KV at later reads isolates the first recursion's hidden-state path contribution, g_{hidden}. Subtracting it from the ordinary first-recursion gradient $g^{(1)}$ gives the shared-cache contribution, g_{KV}. We verify that contributions from all parameter uses sum to the ordinary shared-parameter gradient.

The bars show arithmetic mean gradient norms across sequences. To split the first bar, we project each path onto its total gradient within each sequence:

$$p_a = \frac{\langle g_a, g^{(1)} \rangle}{\|g^{(1)}\|_2}, \qquad a \in \{\text{KV}, \text{hidden}\}, \qquad p_{\text{KV}} + p_{\text{hidden}} = \|g^{(1)}\|_2. \tag{5}$$

We then average these projected contributions. Their sum equals the mean first-recursion norm. The two path norms themselves are not additive. The projections are nonnegative in all plotted measurements, without clipping. Whiskers are 95% sequence-bootstrap intervals for the full bar, not for its segments or for variation across training seeds.

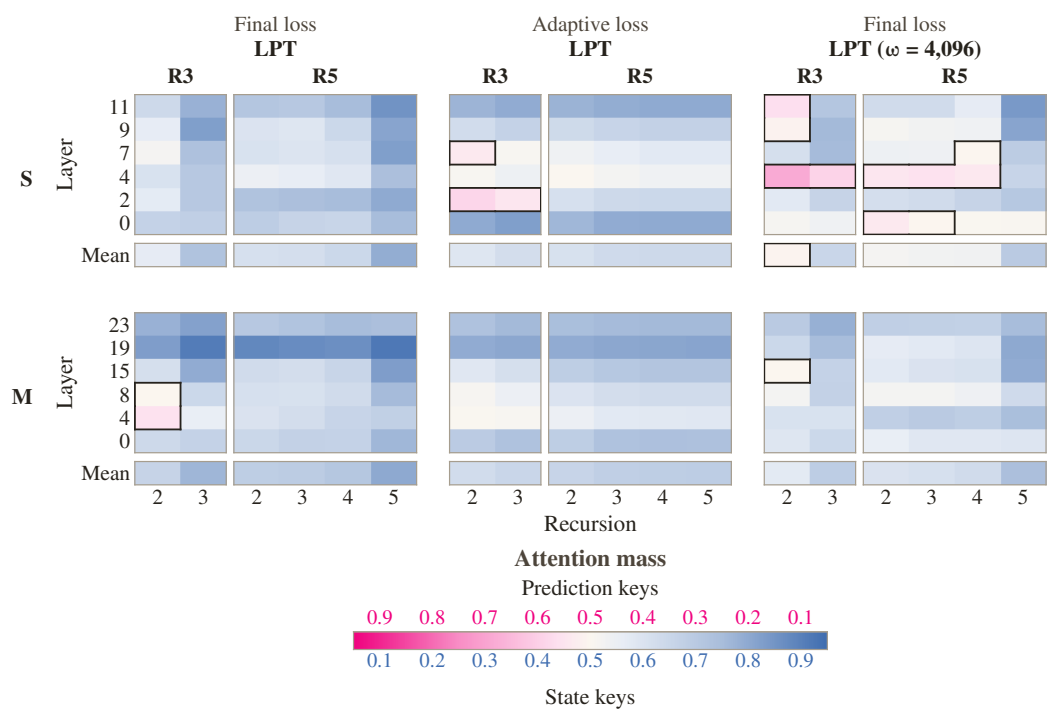


Figure 10: **Attention to shared memory across training objectives and window sizes.** Expanded comparison for Figure 5. Mean rows average all layers.

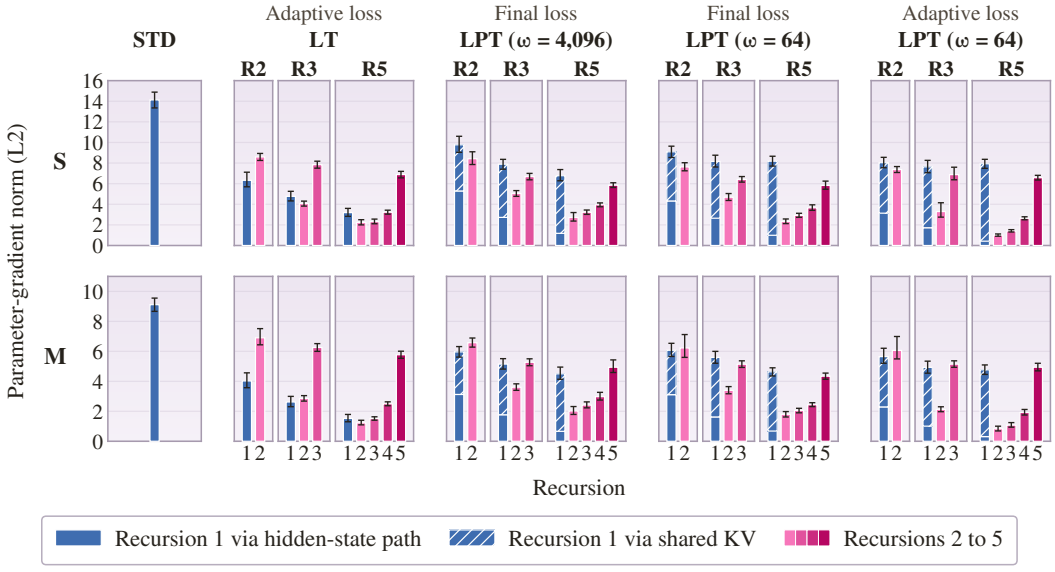


Figure 11: **Shared-cache feedback persists across training objectives and window sizes.** Expanded version of Figure 7, with the same sequences and final-loss gradient probe in every condition.

Figure 11 shows that the shared-cache contribution remains strong with a full prediction window and under adaptive loss. This is consistent across S and M. The compact main-paper comparison therefore does not depend on using final loss or a short prediction window for LPT.

Gradient Paths at Initialization Figure 12 compares the norm ratio $\|g_{\mathrm{KV}}\|_2/\|g_{\mathrm{hidden}}\|_2$ before and after training, using geometric means over the same 32 sequences. Fresh initialization is measured

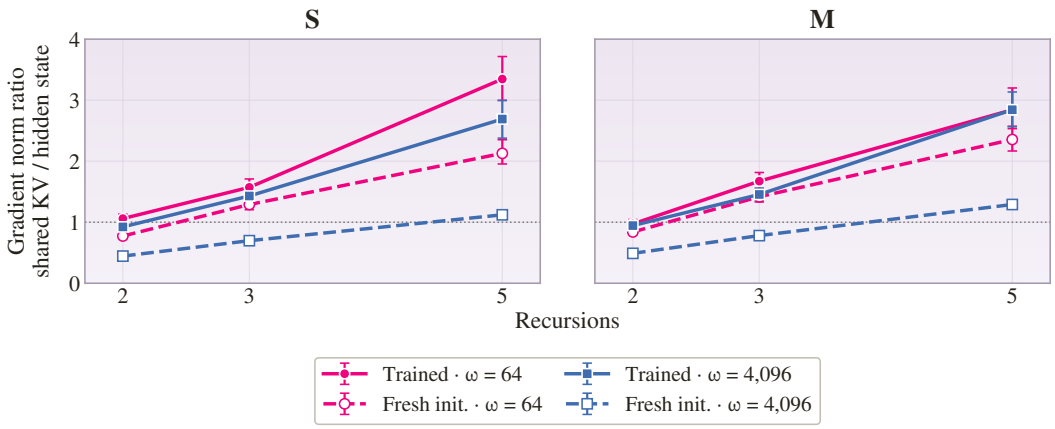


Figure 12: **Shared-KV feedback relative to the hidden-state path.** Gradient-norm ratios at fresh initialization and after training. Whiskers: 95% sequence-bootstrap intervals.

at two, three, and five recursions for both windows and sizes. It uses separate random weights, not saved initial checkpoints. The ratio increases with depth even at initialization, but is larger in the trained models. More recursions add more shared-cache reads, so this trend alone does not establish that the feedback helps learning. The detachment experiment tests that separately.

Shared-Cache Gradient Ablation We train S and M LPT models with $R \in \{2, 3, 5\}$, $\omega = 64$, and final loss, changing only whether the shared KV entries are detached at each read by a later recursion. The first recursion still receives gradients through the hidden-state path, and all shared parameters remain trainable. This removes direct gradient feedback through shared-cache reads, not all training of the first recursion.

Table 2 compares final-loss LT, detached LPT, and LPT after 20B training tokens. Detachment raises NLL in every comparison. Detached S at five recursions finishes well above LPT, while M at three and five recursions remains near unigram-level loss. The experiment supports a role for shared-cache gradients in learning and stability, but does not distinguish shorter gradient paths from changes in gradient scale or direction.

E Experiments at Nine Recursions

Table 7 and Figure 13 extend the comparisons to nine recursions. We also list SPS (Monea et al., 2026) in the $R = 2$ column, since its memory and FLOPs match those of LPT with two recursions. From five to nine, LPT gains 0.7 accuracy points at S and 0.2 at M, but loses 1.7 at L. NLL improves at S and worsens at M and L. LT and PLT also worsen in NLL at every size. Both hybrids continue to improve: H-LPT gains 0.4, 0.2, and 0.6 accuracy points at S, M, and L, respectively, and reaches the lowest NLL of all models at every size. At nine recursions, LPT still outperforms LT at every size, and H-LPT outperforms LT2 in both NLL and accuracy.

Higher recurrent depth can also complicate training. Ouro reports loss spikes and gradient oscillations at eight recursions and reduces its recurrent depth to four (Zhu et al., 2025). LT2 also uses four recursions (Deng et al., 2026).

Table 7: **Quality and efficiency through nine recursions.** One table per size. Metrics and formatting follow Table 1: in the STD row, $R = 1$ is the standard Transformer and $R > 1$ is DEEPSTD. SPS appears in the $R = 2$ column, whose memory and FLOPs it matches.

S	$R=1$		$R=2$				$R=3$				$R=5$				$R=9$			
Model	NLL ↓	Acc. ↑	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓
STD	2.824	51.5	2.838	50.2	1.50	**1.19**	2.850	50.7	1.88	**1.31**	2.874	50.4	2.50	**1.54**	2.909	49.1	3.38	**1.81**
SPS			2.785	52.6	1.02	1.84												
LOOPIE			2.809	51.6	2.00	1.83	2.804	51.9	3.00	2.66	2.808	52.1	5.00	4.32	2.833	51.0	9.00	7.64
LT			2.784	51.5	2.00	1.83	2.772	52.1	3.00	2.66	2.773	52.4	5.00	4.32	2.782	52.2	9.00	7.64
PLT			2.763	**52.7**	1.02	1.84	2.744	52.9	1.03	2.67	2.740	53.5	1.06	4.34	2.742	53.2	1.12	7.68
LT2			2.759	51.9	0.35	1.29	2.740	52.5	0.52	1.86	2.720	53.7	0.87	2.98	2.707	54.4	1.56	5.23
LPT			**2.756**	51.9	1.02	1.84	**2.726**	**54.0**	1.03	2.67	**2.702**	**54.9**	1.06	4.34	2.695	**55.6**	1.12	7.68
H-LPT			2.757	52.4	**0.18**	1.29	2.735	53.1	**0.19**	1.86	2.710	54.6	**0.21**	2.98	**2.692**	55.0	**0.25**	5.23

M	$R=1$		$R=2$				$R=3$				$R=5$				$R=9$			
Model	NLL ↓	Acc. ↑	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓
STD	2.625	56.1	2.657	55.5	1.44	**1.13**	2.673	55.4	1.78	**1.24**	2.701	54.4	2.34	**1.43**	2.747	53.4	3.09	**1.63**
SPS			2.565	58.9	1.02	1.93												
LOOPIE			2.617	56.9	2.00	1.92	2.621	55.8	3.00	2.84	2.620	56.6	5.00	4.69	2.760	52.9	9.00	8.37
LT			2.588	57.5	2.00	1.92	2.576	59.4	3.00	2.84	2.592	57.9	5.00	4.69	2.620	56.9	9.00	8.37
PLT			2.565	59.1	1.02	1.93	2.540	60.0	1.03	2.85	2.538	60.5	1.06	4.71	2.555	59.5	1.12	8.41
LT2			2.551	59.1	0.35	1.43	2.537	59.5	0.52	2.10	2.521	60.2	0.87	3.45	2.512	60.4	1.56	6.14
LPT			2.558	**59.2**	1.02	1.93	**2.530**	60.2	1.03	2.85	2.512	**61.6**	1.06	4.70	2.517	**61.8**	1.12	8.41
H-LPT			**2.550**	58.5	**0.18**	1.43	**2.530**	**60.3**	**0.19**	2.10	**2.511**	61.4	**0.21**	3.45	**2.498**	61.6	**0.25**	6.14

L	$R=1$		$R=2$				$R=3$				$R=5$				$R=9$			
Model	NLL ↓	Acc. ↑	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓	NLL ↓	Acc. ↑	KV ↓	FLOPs ↓
STD	2.429	62.5	2.453	62.5	1.45	**1.14**	2.481	61.3	1.73	**1.17**	2.507	61.4	2.25	**1.32**	2.539	59.8	3.15	**1.63**
SPS			2.377	65.5	1.02	1.96												
LOOPIE			2.421	63.5	2.00	1.95	2.423	63.9	3.00	2.91	2.506	61.0	5.00	4.81	2.445	63.2	9.00	8.63
LT			2.416	64.2	2.00	1.95	2.408	63.8	3.00	2.91	2.433	63.4	5.00	4.81	2.527	60.2	9.00	8.63
PLT			2.393	64.8	1.02	1.96	2.375	66.8	1.03	2.92	2.360	66.4	1.06	4.83	2.378	66.1	1.12	8.66
LT2			2.360	**66.5**	0.40	1.53	2.348	65.8	0.60	2.27	2.334	67.3	1.01	3.75	2.323	68.1	1.81	6.71
LPT			2.375	66.3	1.02	1.96	2.347	**67.3**	1.03	2.92	2.334	**68.2**	1.06	4.83	2.386	66.5	1.12	8.66
H-LPT			**2.359**	65.6	**0.21**	1.53	**2.343**	66.9	**0.22**	2.27	**2.325**	68.1	**0.24**	3.75	**2.313**	**68.7**	**0.28**	6.72

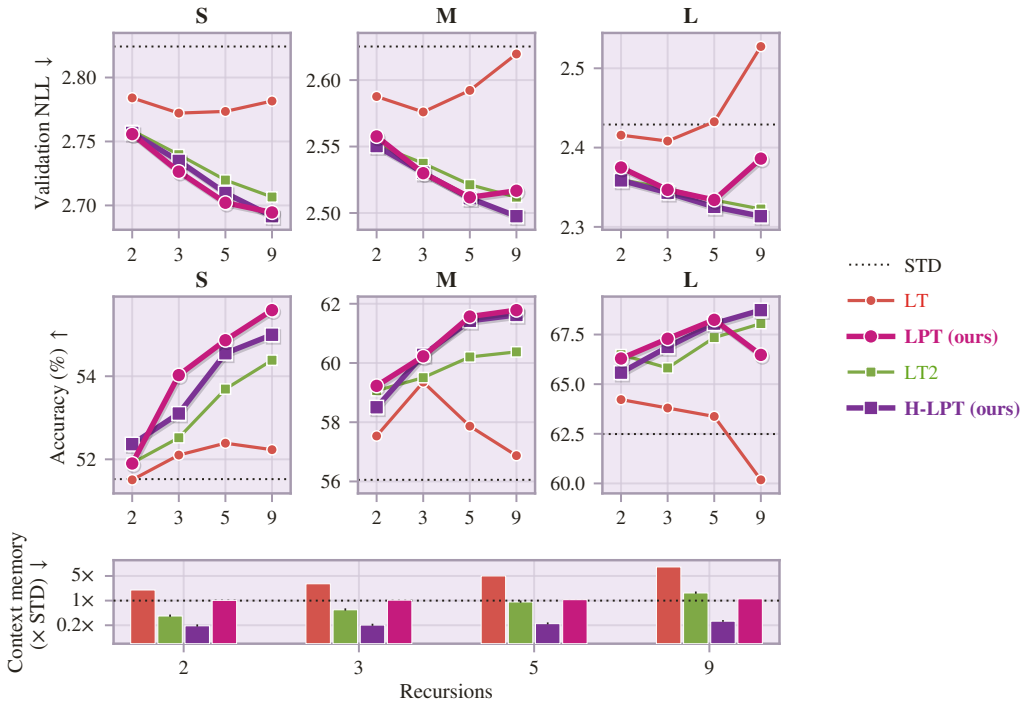


Figure 13: **Quality and memory through nine recursions.** Validation NLL, mean five-task accuracy, and context memory relative to STD.

Intermediate Readouts We apply the model's readout normalization and language-modeling head to the output of each recursion. The measurements use S-size LPT checkpoints trained for 20B tokens, averaged over 16 validation sequences of 4,096 tokens. With nine recursions and final loss, intermediate-readout NLL ranges from 4.47 to 6.92, falling to 2.75 at the last recursion. With adaptive loss, it is already 2.79 by the fourth recursion and remains near that value. Under final loss, the head is trained only on the last recursion: high intermediate NLL measures poor prediction through this head, not the absence of predictive information in the hidden state. These probe-set losses are separate from the full validation-set NLL in Table 7.

Complete Quality–Memory Sweep Figure 14 includes every family, size, and recursion count in Table 7 except SPS and PLT. Figure 15 compares nine-recursion models with STD across sizes.

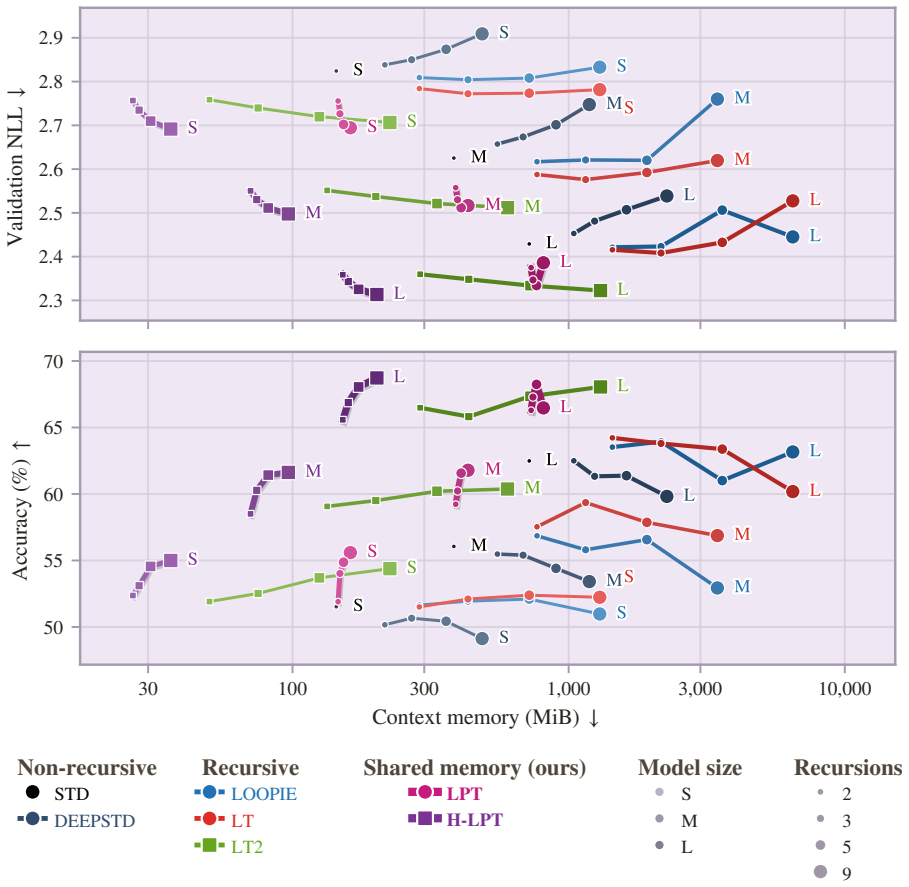


Figure 14: **Quality versus context memory across model sizes.** Curves vary recursion count. Marker size denotes recursions, and shade denotes model size. S/M train for 20B tokens and L for 50B.

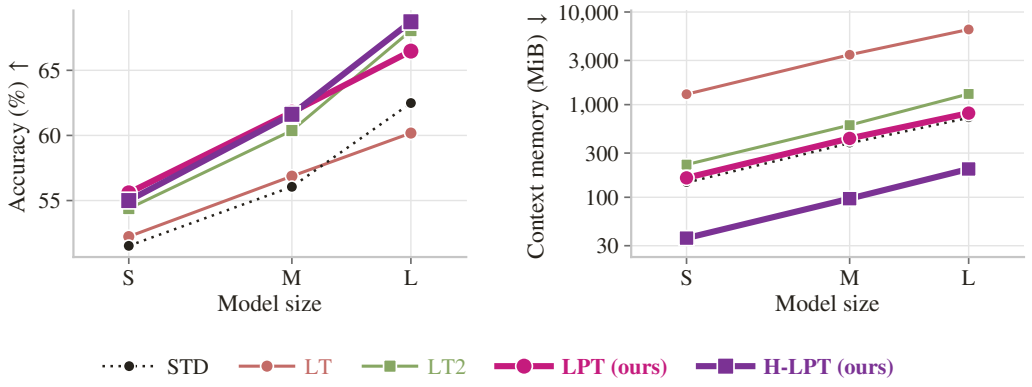


Figure 15: **Quality and memory across model sizes.** Mean five-task accuracy and context memory for STD and looped models at nine recursions.

F DECODING SPEED AND MEMORY ON A GPU

We test whether the analytic memory savings translate into larger batches and higher decoding throughput on a GPU. A smaller cache lets more sequences share a GPU, which can raise throughput (Brandon et al., 2024; Sun et al., 2024). As O'Neill & Reid (2026) do for a looped model, we measure the largest batch that fits at a fixed context and the throughput at that batch.

Setup We decode on one NVIDIA H100 with 80 GB of memory, in bf16, at a 4,096-token context: each step generates one token per sequence from a cache already filled to 4,068 tokens. All models share one decoding implementation in PyTorch: a preallocated cache per attention layer, FlexAttention (Dong et al., 2025) for attention, and CUDA graphs that replay a whole decoding step, all recursions and layers included. A later recursion of LPT or H-LPT attends to its shared and local caches in one softmax, which we compute by merging the two FlexAttention outputs by their log-sum-exp. Gated DeltaNet layers update their recurrent state one token at a time, with the kernels of flash-linear-attention (Yang & Zhang, 2024). STD runs the same code with one recursion, and LT runs all its recursions, without early exit. Weights are random, since speed and memory do not depend on them. On the same GPU, decoding reproduces the full forward pass to bf16 precision. The maximum batch is the largest batch whose decoding steps run without running out of memory, and we measure throughput at this batch, or at 95% of it when the CUDA graph does not fit. These measurements come from one research implementation, not an optimized serving engine, which would change the absolute speeds. Every model runs the same code, so the comparison holds within this implementation.

Capacity Table 8 shows that the analytic savings carry over to the GPU. LPT's cache per sequence is at most 6.25% larger than STD's, so it fits 93–98% of STD's batch. LT's cache grows with depth, so LPT fits 2.0, 2.9, and 4.7–4.8 times as many sequences as LT at two, three, and five recursions. H-LPT holds 20–27% of STD's cache and fits 3.6–5.0 times as many sequences as STD, and 1.8–3.8 times as many as LT2. The hybrids' measured caches are 3–17% larger than the analytic values in Table 1, because the kernel keeps the recurrent state in fp32 and we also count the short-convolution windows.

Speed The larger batches raise LPT's peak throughput above LT's by 7–91%, most at L with five recursions, where LT fits only 21 sequences. At equal batch, LPT is 3–8% slower than LT, since each later recursion still reads the whole shared cache, and its own window too. As Brandon et al. (2024) note for sharing across layers, a shared cache saves memory, not the time to read it. Both looped models pay for depth in time: at batch 1, a decoding step of LT takes about 2.3, 3.4, and 5.7 times as long as STD's at two, three, and five recursions, and LPT's takes 8–12% longer than LT's. The hybrids read a full-context cache in only one layer of five, so they decode faster. At its maximum batch, H-LPT generates 2.1–2.2 times as many tokens per second as STD with two recursions, 1.4–1.5 times with three, and 0.8–0.9 times with five, and it is 2–3% slower than LT2 at equal batch.

Throughput and Quality Figure 16 plots quality against throughput. At every size and recursion count, LPT has higher throughput, lower NLL, and higher accuracy than LT, and H-LPT has higher throughput and lower NLL than LT2. Against STD, H-LPT lowers NLL by 0.086–0.095 with three recursions while decoding 1.4–1.5 times as many tokens per second, and by 0.104–0.114 with five, at 81–88% of STD's throughput.

Table 8: **Decoding capacity and speed on one H100.** One table per size. bf16, 4,096-token context. Batch: the largest batch that decodes without running out of the 80 GB. Tok/s: generated tokens per second at that batch. Lat.: milliseconds per decoding step at batch 1. Our rows are shaded.

S	$R=1$			$R=2$			$R=3$			$R=5$		
Model	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.
STD	552	18,570	2.1									
LT				275	8,414	4.8	184	5,172	7.1	110	2,688	11.8
LPT				540	8,975	5.2	533	5,894	7.9	514	3,544	13.3
LT2				1,513	39,006	2.6	1,007	25,470	3.9	603	14,348	6.4
H-LPT				2,746	40,066	2.7	2,531	26,692	4.0	2,190	15,900	6.7

M	$R=1$			$R=2$			$R=3$			$R=5$		
Model	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.
STD	206	6,478	4.2									
LT				103	2,703	9.7	68	1,527	14.5	41	763	24.1
LPT				202	3,074	10.5	199	2,000	16.0	192	1,201	27.0
LT2				563	13,804	5.3	377	8,511	8.0	224	4,502	13.2
H-LPT				1,029	14,549	5.5	949	9,622	8.2	821	5,706	13.6

L	$R=1$			$R=2$			$R=3$			$R=5$		
Model	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.	Batch	Tok/s	Lat.
STD	108	3,162	6.7									
LT				54	1,203	15.3	36	691	22.8	21	303	37.9
LPT				106	1,480	16.4	104	976	25.0	101	578	42.3
LT2				258	5,986	8.8	172	3,585	13.1	103	1,776	21.7
H-LPT				475	6,612	9.0	444	4,333	13.6	390	2,553	22.6

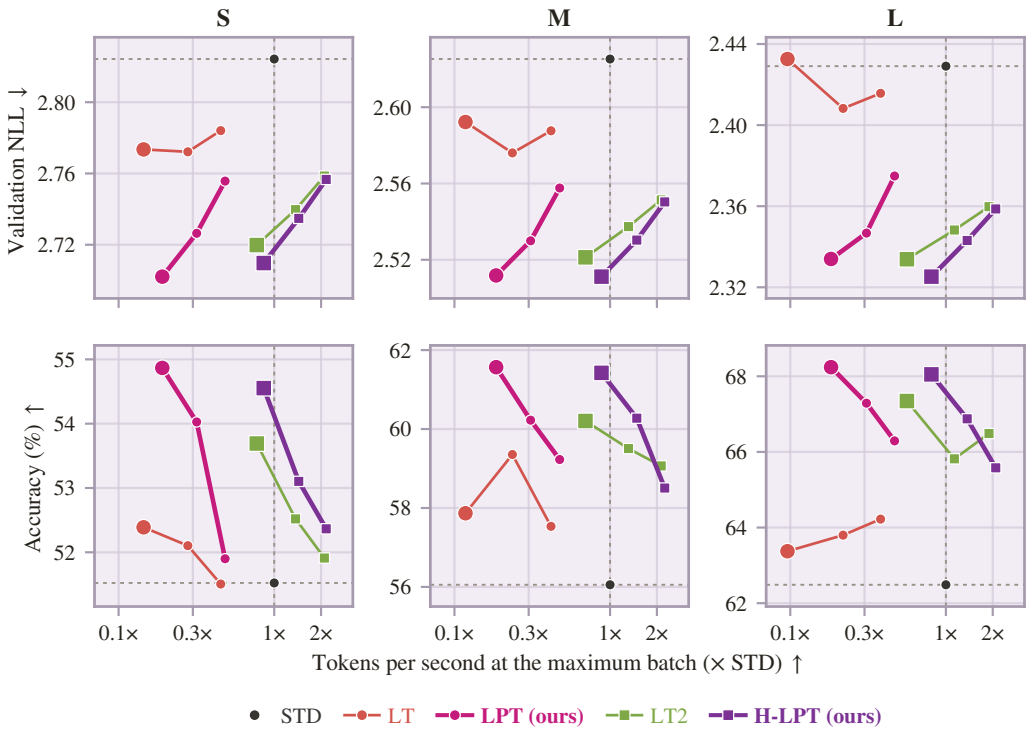


Figure 16: **H-LPT is better than STD at similar or higher throughput.** Validation NLL (top) and mean five-task accuracy (bottom) against generated tokens per second at each model's maximum batch, relative to same-size STD, on one H100 at a 4,096-token context. Curves follow two, three, and five recursions, with larger markers at five. Dashed lines mark STD.