STATELOOP: PERSISTENT STATE FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers offer a parameter-efficient approach to language modeling by repeatedly applying a shared backbone. Vanilla looped Transformers use each backbone output directly as the input to the next iteration. These outputs may contain information useful several iterations later, but retaining it across the loop depends entirely on the single hidden state. We introduce **StateLoop**, which maintains one additional persistent state per token to retain information across iterations. At each iteration, a learned gate controls how much of the previous state is retained and how much information from the backbone output is incorporated. The updated state is projected back and added to the backbone output for subsequent iterations and final prediction. This design reduces Pile perplexity and improves average accuracy across eight zero-shot tasks over Vanilla Looped. Its perplexity advantage over Vanilla Looped grows with trained loop depth, consistent with persistent state becoming increasingly useful as recurrent computation deepens.

1 INTRODUCTION

Looped Transformers increase computation by repeatedly applying a shared backbone (Dehghani et al., 2019). Saunshi et al. (2025) show that looped models improve language modeling over non-looped models with the same number of parameters. Ouro demonstrates this approach at larger scale, with 1.4B and 2.6B looped models achieving performance comparable to larger standard Transformers across downstream benchmarks (Zhu et al., 2025).

The design of connections between iterations is a central architectural consideration in looped Transformers. Vanilla looped Transformers use each backbone output directly as the input to the next iteration. As Pagliardini et al. (2024) show, DenseFormer achieves lower perplexity when later layers can access much earlier outputs than when access is restricted to recent outputs. This finding motivates the hypothesis that information produced in one iteration may likewise remain useful several iterations later. In vanilla looped Transformers, retaining such information across iterations relies entirely on the same hidden state that also supplies the input for subsequent backbone computation.

We therefore propose StateLoop, which maintains an additional persistent state for each token. The state is carried across iterations and updated by combining its previous value with a projection of the backbone output. A gate determined by the same output controls how much of the previous state is retained and how much new information is written. An optional state mixer uses causal attention or Mamba-2 to exchange information across token positions before readout. The updated state is projected back and added to the backbone output, making the retained information available to later iterations and final prediction.

We evaluate StateLoop at 410M and 1.4B scales to assess both its performance and the contribution of the persistent state. Compared with Vanilla Looped, StateLoop reduces Pile perplexity by 0.18 and improves average accuracy across eight zero-shot tasks by 0.97 percentage points at 1.4B. Adding causal attention and Mamba-2 state mixers further increases the accuracy gains over Vanilla Looped to 1.43 and 1.44 percentage points, respectively. We further analyze the role of the persistent state. Resetting the state at every iteration increases perplexity, supporting the value of carrying information across iterations. Moreover, StateLoop's advantage over Vanilla Looped grows with trained loop depth, suggesting that persistent state becomes increasingly useful as recurrent computation deepens.

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.

2 Related Work

Looped Transformers and recurrent depth. The Universal Transformer introduced recurrent computation across depth by repeatedly applying a shared block (Dehghani et al., 2019). Subsequent work shows that looped Transformers can learn iterative algorithms for in-context learning (Yang et al., 2024). Ouro extends this approach to large-scale language-model pretraining with learned depth allocation (Zhu et al., 2025).

Connections between iterations. Recent looped language models differ in how they carry information across iterations and construct the next backbone input. Parcae stabilizes the recurrent hidden state by constraining the spectral norm of its state-transition matrix (Prairie et al., 2026). PonderLM feeds back a weighted sum of token embeddings, with weights given by the predicted token probabilities (Zeng et al., 2026). Hyperloop carries multiple residual streams across iterations and uses loop-specific hyper-connections to read the backbone input from these streams and write the output back (Zeitoun et al., 2026). These methods modify or enrich the recurrent stream itself, whereas StateLoop retains the direct backbone stream and introduces a separate persistent state.

Persistent state and memory. LSTMs, GRUs, and selective state-space models update recurrent state along a sequence (Hochreiter & Schmidhuber, 1997; Cho et al., 2014; Gu & Dao, 2024), while Recurrent Memory Transformers carry memory tokens between sequence segments (Bulatov et al., 2022). In looped architectures, Adaptive Loops and Memory augments repeated computation with gated access to static learned memory banks (Frey et al., 2026), while Universal Transformers Need Memory studies dynamically updated memory tokens as a computational scratchpad for recursive reasoning (Sapunov, 2026). MeSH externalizes recurrent state management into a multi-slot buffer with iteration-specific read and write routers (Yu et al., 2026). StateLoop explores a simpler design point: it maintains a single continuously updated state per token, retains the main recurrent hidden stream as a direct path, and shares all state-update and feedback parameters at every iteration.

3 Method

StateLoop adds a persistent state to each token. At each iteration, a gated update incorporates the backbone output into the state, which is then read to form the next backbone input or the final prediction. Figure 1 shows this computation.

3.1 Backbone Feedback and Persistent State

Let $x = (x_1, \ldots, x_n)$ be a token sequence and T the shared Transformer backbone, with backbone width d and state width d_s. We use $K \geq 1$ loop iterations in total, indexed by $k = 0, \ldots, K-1$. Initialization is

$$E = \sqrt{d}\,\mathrm{Embed}(x), \qquad S^0 = 0, \qquad H^0 = T(E). \tag{1}$$

Here $H^k \in \mathbb{R}^{n\times d}$ is the backbone output at iteration k, and $S^k \in \mathbb{R}^{n\times d_s}$ is the state after k updates, before incorporating H^k. Updating it with H^k produces S^{k+1}. Linear projections act on each token's feature vector.

At iteration $k = 0, \ldots, K-1$, the state branch receives the backbone output normalized with RMSNorm (Zhang & Sennrich, 2019),

$$U^k = \mathrm{RMSNorm}_h(H^k). \tag{2}$$

The state update $\mathcal{V}_\phi$ combines S^k and U^k independently at each token. The operator $\mathcal{X}_\psi$ then applies identity or causal token mixing. The resulting state is normalized for readout, giving

$$\widetilde{S}^{k+1} = \mathcal{V}_\phi(S^k, U^k), \qquad S^{k+1} = \mathcal{X}_\psi(\widetilde{S}^{k+1}), \qquad N^{k+1} = \mathrm{RMSNorm}_s(S^{k+1}). \tag{3}$$

Section 3.2 defines the state update and the mixing options.

For $k = 0, \ldots, K-2$, the backbone output H^k and the projected state read form the input $Z^{k+1} \in \mathbb{R}^{n\times d}$ to the next iteration. The backbone then computes its next output from this input,

$$\boxed{\begin{aligned} Z^{k+1} &= H^k + N^{k+1} W_{s\to z}^\top, \\ H^{k+1} &= T(Z^{k+1}). \end{aligned}} \tag{4}$$

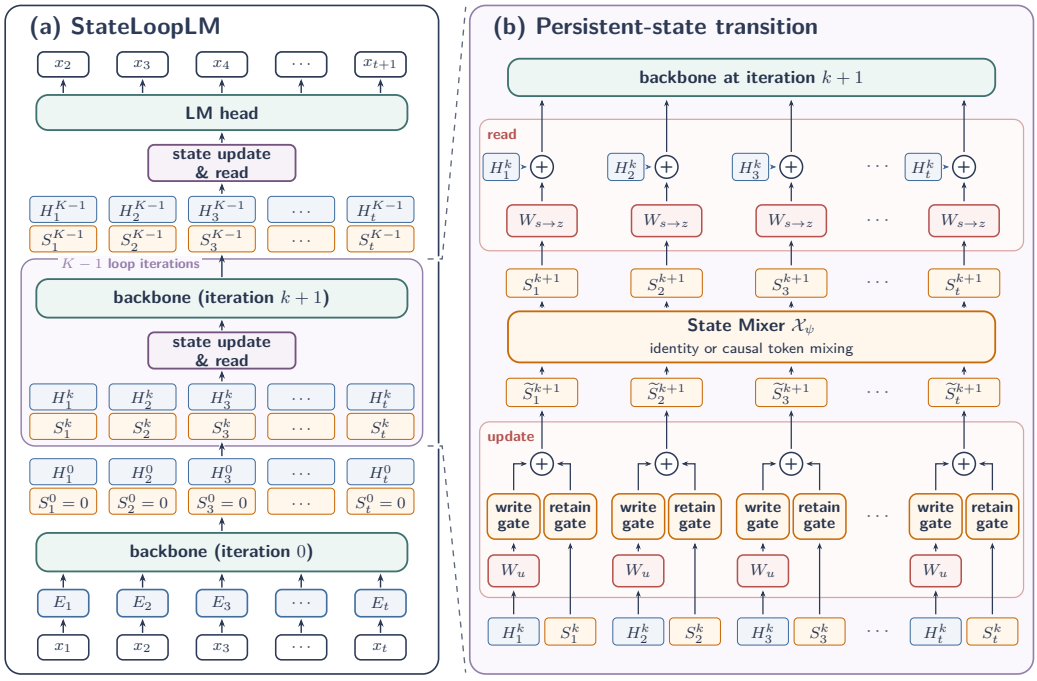


Figure 1: **StateLoop architecture.** (a) The looped model with a persistent state per token. (b) State update, optional token mixing, and readout. $W_{s\to z}$ feeds the next iteration, whereas $W_{s\to out}$ supplies the final prediction.

The projection $W_{s\to z} \in \mathbb{R}^{d\times d_s}$ maps the state read to the backbone width. The backbone, state update, feedback projection, and optional token mixer share their parameters across iterations.

3.2 State Update

3.2.1 Selective SSM Update

The selective-SSM recurrence operates along the loop axis independently for each token, while $\mathcal{X}_\psi$ provides optional mixing along the token axis. A learned projection maps the normalized backbone output to the state width, giving

$$P^k = U^k W_u^\top, \qquad W_u \in \mathbb{R}^{d_s\times d}. \tag{5}$$

The normalized backbone output U^k determines an input-dependent step size, following selective SSMs (Gu & Dao, 2024). We use the update

$$\begin{aligned} \Delta^k &= \mathrm{softplus}\big(U^k W_\Delta^\top + b_\Delta\big), & \lambda &= \exp(A_{\log}),\\ \widetilde{S}^{k+1} &= \exp(-\lambda\odot\Delta^k)\odot S^k + \big(1-\exp(-\lambda\odot\Delta^k)\big)\odot P^k. \end{aligned} \tag{6}$$

Here $W_\Delta \in \mathbb{R}^{d_s\times d}$, $b_\Delta, A_{\log}\in\mathbb{R}^{d_s}$, and $\odot$ denotes elementwise multiplication. The positive rate λ is learned per channel. The retention gate $g^k = \exp(-\lambda\odot\Delta^k)$ weights the previous state, while the write gate $\eta^k = 1-g^k$ weights the projected output P^k. Because Δ^k depends on U^k, these weights can vary with the token and the iteration.

We initialize $W_\Delta = 0$ and $b_\Delta = \log(e^{0.3}-1)$, so that $\Delta = 0.3$ initially, and initialize $\lambda = \exp(A_{\log})$ logarithmically from 0.01 to 4 across channels. These parameters are shared across iterations and remain trainable. The resulting initial write rate is $\eta_c = 1-\exp(-0.3\lambda_c)$.

3.2.2 Optional Token-Axis Mixing

StateLoop uses identity mixing, so the updated state is normalized and read out directly, with

$$S^{k+1} = \mathcal{X}_{\mathrm{id}}(\widetilde{S}^{k+1}) = \widetilde{S}^{k+1}. \tag{7}$$

Adding causal attention or Mamba-2 gives StateLoop + Attn Mixer or StateLoop + Mamba-2 Mixer, respectively, allowing updated states to exchange information across token positions. Both preserve the shape $n \times d_s$ and compute the output at position t from the updated-state prefix $\widetilde{S}^{k+1}_{\leq t}$.

The causal-attention mixer uses self-attention (Vaswani et al., 2017) with pre-normalization and a residual connection, giving

$$S^{k+1} = \widetilde{S}^{k+1} + \mathrm{CausalAttention}_{\psi}\left(\mathrm{RMSNorm}_{m}(\widetilde{S}^{k+1})\right). \tag{8}$$

The Mamba-2 mixer applies a residual Mamba-2/SSD block along the token axis (Dao & Gu, 2024) to obtain

$$S^{k+1} = \mathrm{Mamba2Block}_{\psi}(\widetilde{S}^{k+1}). \tag{9}$$

3.3 Final Prediction and Training

At the final iteration $k = K-1$, H^{K-1} updates S^{K-1} to S^{K}. A separate readout projection $W_{s\to out} \in \mathbb{R}^{d\times d_s}$ combines the normalized state with H^{K-1} to produce

$$\boxed{Y = H^{K-1} + \mathrm{RMSNorm}_{s}(S^{K})W^{\top}_{s\to out} = H^{K-1} + N^{K}W^{\top}_{s\to out}.} \tag{10}$$

The LM head maps Y to vocabulary logits. All parameters are trained end-to-end by minimizing the standard autoregressive loss,

$$p_{\theta}(\cdot \mid x_{\leq t}) = \mathrm{softmax}\big(\mathrm{LMHead}(Y_t)\big), \tag{11}$$

$$\mathcal{L}_{\mathrm{LM}} = -\frac{1}{n-1}\sum_{t=1}^{n-1}\log p_{\theta}(x_{t+1} \mid x_{\leq t}). \tag{12}$$

4 Experiments

We evaluate language modeling and downstream accuracy, examine performance across trained loop depths, and measure autoregressive decoding efficiency. Section 5 examines the design and behavior of the persistent state.

4.1 Experimental Setup

Training configuration. All models are trained from scratch on the Pile (Gao et al., 2020) with a LLaMA-style backbone (Touvron et al., 2023). All 1.4B comparisons use the same GPT-NeoX tokenizer, data pipeline, 30B-token training budget, batch size 256, sequence length 2,048, and cosine-decay schedule with peak learning rate 2×10^{-4}. The shared Transformer backbone contains 24 layers, with hidden width $d = 2048$, SwiGLU intermediate width 5504, and 32 attention heads of dimension 64. StateLoop uses persistent-state width $d_s = 2048$, and the full-backbone recurrent models use $K = 4$ iterations of the shared backbone. The 410M studies use 25B training tokens and a peak learning rate of 3×10^{-4}, with the same batch size, sequence length, and cosine-decay schedule. Training budgets count original language-model tokens.

State-branch configuration. Both token-axis mixers consist of a single residual module whose parameters are shared across iterations. In the 1.4B model, the attention mixer uses pre-normalized causal self-attention with rotary position embeddings (RoPE), 32 heads of dimension 64, and no FFN. The Mamba-2 mixer uses an expansion factor of 2, an internal SSM state dimension of 128, a convolution width of 4, a head dimension of 64, and 8 groups, without additional positional embeddings.

Comparison models. All baselines are implemented in the same codebase. Vanilla Looped (Dehghani et al., 2019) and PonderLM (Zeng et al., 2026) apply the full backbone for four iterations. Vanilla Looped + FFN increases the SwiGLU intermediate width from 5,504 to 5,818, yielding approximately the same parameter count as StateLoop + Mamba-2 Mixer. Our Hyperloop (Zeitoun

et al., 2026), MeSH (Yu et al., 2026), and Parcae (Prairie et al., 2026) baselines retain the partial-loop structures used in their original studies. Hyperloop follows its original begin–middle–end structure, with only the middle block applied recurrently. For MeSH and Parcae, we use eight layers each for the prelude, recurrent core, and coda, with ten core repetitions, matching the 96 layer applications of four StateLoop iterations, each with 24 layers. Pause Token inserts three pause tokens after each input token to provide additional computation before predicting the next token (Goyal et al., 2024).

Evaluation metrics. We report perplexity (PPL) on Pile validation data, LAMBADA-OpenAI (LAMBADA-OAI), and WikiText-2. PPL is the exponential of mean negative log-likelihood (NLL). Pile and LAMBADA-OAI use token PPL, while WikiText-2 uses word PPL (Paperno et al., 2016; Radford et al., 2019; Merity et al., 2017). Downstream evaluation covers LAMBADA-OAI, ARC-Easy, ARC-Challenge, WinoGrande, PIQA, HellaSwag, SciQ, and RACE-high (RACE in the tables) under both zero-shot and five-shot settings (Clark et al., 2018; Sakaguchi et al., 2020; Bisk et al., 2020; Zellers et al., 2019; Welbl et al., 2017; Lai et al., 2017). We report task accuracy and Avg-8, the unweighted average across these eight tasks.

4.2 Main Results

Tables 1 and 2 compare three StateLoop variants with the baselines under the common 1.4B training setup. The variants share the selective-SSM update and feedback rule and apply identity, causal attention, or Mamba-2 to the updated state.

Relative to Vanilla Looped, StateLoop lowers perplexity on all three datasets and improves average downstream accuracy under both prompting settings (Tables 1 and 2). Both token mixers further improve Avg-8, with similar scores in each setting.

In this parameter-control comparison, allocating the additional parameters to the state branch yields better performance than widening the FFN. StateLoop outperforms Vanilla Looped + FFN in Pile PPL and both downstream averages with fewer parameters, while StateLoop + Mamba-2 Mixer does so at a comparable parameter count (Tables 1 and 2).

4.3 Loop Depth

We next test whether the benefit of the additional state grows with the number of iterations, holding state width fixed. With $d = d_s = 1024$, we train paired 410M Vanilla Looped and StateLoop models using $K = 3, \ldots, 7$ iterations. Each model uses 25B training tokens and is evaluated at the depth used during training. With identity mixing, StateLoop has 411.10M parameters at every tested loop depth.

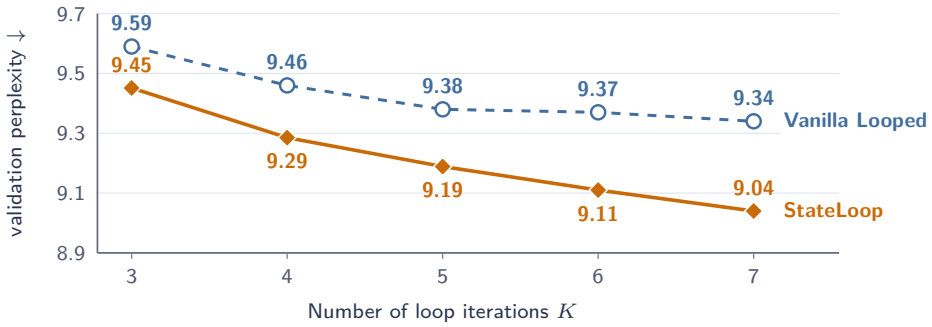


Figure 2: **Loop depth and Pile PPL at 410M.** Models are trained for 25B tokens at each plotted K. StateLoop uses identity mixing and $d = d_s = 1024$. The PPL advantage over Vanilla Looped grows with trained loop depth.

Both models benefit from more iterations, with larger gains for StateLoop (Figure 2). Its PPL advantage over Vanilla Looped grows from approximately 0.14 at $K = 3$ to 0.30 at $K = 7$, consistent with persistent state becoming increasingly useful as recurrent depth increases.

Table 1: **Language-modeling PPL at 1.4B (30B training tokens).** Full-backbone models use $K = 4$, with MeSH and Parcae matched at 96 layer applications. LAMBADA-OpenAI uses zero-shot token PPL and WikiText-2 uses word PPL.

	Parameters		Perplexity ↓		
Model	Params (M)	vs. Vanilla Looped	Pile	LAMBADA OAI	WikiText-2
Vanilla Looped	1420.92	0.00%	7.79	6.94	16.50
Vanilla Looped + FFN	1467.22	+3.26%	7.75	7.00	16.34
Hyperloop	1421.32	+0.03%	7.77	6.84	16.66
MeSH	1421.51	+0.04%	7.74	6.77	16.43
PonderLM	1420.92	0.00%	7.85	7.31	16.72
Parcae	1429.32	+0.59%	7.87	7.38	16.83
Pause Token	1420.92	0.00%	8.20	7.74	17.87
StateLoop	1437.71	+1.18%	7.61	6.71	15.96
StateLoop + Attn	1454.49	+2.36%	**7.60**	**6.64**	**15.93**
StateLoop + Mamba-2	1467.23	+3.26%	**7.60**	6.88	15.95

Table 2: **Zero-shot and five-shot accuracy (%) at 1.4B after 30B training tokens.** Full-backbone models use $K = 4$; MeSH and Parcae match their 96 layer applications. Avg-8 is the unweighted eight-task mean. LAMBADA-OAI denotes LAMBADA-OpenAI.

Model	LAMBADA OAI	ARC-E	ARC-C	WinoG.	PIQA	Hella.	SciQ	RACE	Avg-8
Zero-shot									
Vanilla Looped	58.24	52.57	28.16	54.85	70.02	50.39	77.90	34.35	53.31
Vanilla Looped + FFN	59.05	50.93	26.71	56.91	69.42	50.50	75.60	33.68	52.85
Hyperloop	**61.28**	51.05	27.56	55.72	70.46	49.69	80.70	34.16	53.83
MeSH	60.94	50.88	26.62	**59.67**	69.70	50.52	78.10	34.55	53.87
PonderLM	58.26	50.59	28.58	55.41	68.93	48.96	78.00	33.68	52.80
Parcae	57.40	50.04	26.71	53.83	70.46	49.34	78.00	34.45	52.53
Pause Token	57.05	49.45	25.77	53.12	68.28	45.42	75.00	32.25	50.79
StateLoop	60.68	52.27	28.33	56.43	70.13	51.19	80.60	34.64	54.28
StateLoop + Attn	59.91	53.79	**29.35**	56.43	70.40	51.61	**81.50**	**34.93**	54.74
StateLoop + Mamba-2	58.99	**55.39**	27.90	59.04	**71.06**	**52.14**	79.20	34.26	**54.75**
Five-shot									
Vanilla Looped	51.64	60.23	29.69	55.17	69.53	50.02	90.90	33.78	55.12
Vanilla Looped + FFN	53.15	60.06	29.52	57.06	70.62	50.85	91.40	31.67	55.54
Hyperloop	**55.81**	58.84	31.66	58.25	69.80	50.45	92.00	34.35	56.39
MeSH	54.94	59.81	29.86	58.41	69.21	51.55	**92.70**	**35.50**	56.50
PonderLM	52.53	57.95	29.35	55.33	68.93	49.36	91.80	33.30	54.82
Parcae	50.94	59.51	30.63	54.93	70.08	49.59	91.50	33.01	55.02
Pause Token	50.79	55.98	28.16	53.28	68.50	45.45	88.60	32.44	52.90
StateLoop	53.89	61.49	30.55	58.09	70.02	52.24	92.20	34.35	56.60
StateLoop + Attn	54.71	60.86	**31.91**	**59.04**	71.11	**52.62**	91.90	35.02	**57.15**
StateLoop + Mamba-2	53.81	**62.58**	30.80	58.80	**71.71**	52.18	92.40	34.74	57.13

4.4 Decoding Efficiency

Table 3 compares the three StateLoop variants and wider-FFN baseline at $K = 4$ with Vanilla Looped at $K = 4$ and $K = 5$, measuring the cost of the state branch against an additional backbone iteration.

During autoregressive decoding, StateLoop inherits the per-loop backbone KV caches of Vanilla Looped. In addition, its identity-mixer variant maintains only a transient d_s-dimensional state for

Table 3: **Decoding efficiency at 1.4B.** Measurements use BF16, batch size 1, and a 1,024-token prefill followed by a 1,024-token continuation on one GPU. Latency is mean ± standard deviation over five runs; changes are relative to Vanilla Looped at $K = 4$.

Model	Params (M)	K	Latency (ms/token) ↓	Throughput (tok/s) ↑	Latency change
Vanilla Looped	1420.92	4	121.224 ± 1.099	8.249	baseline
Vanilla Looped	1420.92	5	151.527 ± 1.064	6.599	+24.997%
Vanilla Looped + FFN	1467.22	4	121.167 ± 1.293	8.253	−0.047%
StateLoop	1437.71	4	122.943 ± 1.483	8.134	+1.418%
StateLoop + Attn	1454.49	4	123.778 ± 1.348	8.079	+2.106%
StateLoop + Mamba-2	1467.23	4	123.397 ± 0.923	8.104	+1.792%

the current token across recurrent iterations. Attention and Mamba-2 mixers additionally maintain loop-specific token-axis caches.

The state branch adds only 1.42%–2.11% latency at $K = 4$, compared with 25.00% for a fifth Vanilla Looped iteration. StateLoop thus has 18.86% lower latency than the $K = 5$ baseline.

5 Ablation and Analysis

We analyze 410M models at $K = 7$, using identity mixing for the StateLoop variants. We first vary state width, then compare feedback designs, state persistence, and update rules. Interventions on a trained model examine how retained information and continued state updates contribute to prediction.

5.1 State Width

We vary the persistent-state width $d_s \in \{128, 256, 512, 1024\}$ to test whether a smaller state preserves the language-modeling gains. Backbone width $d = 1024$ and loop depth $K = 7$ are fixed. StateLoop uses the selective-SSM update with identity mixing. Figure 3 reports PPL and parameter cost.

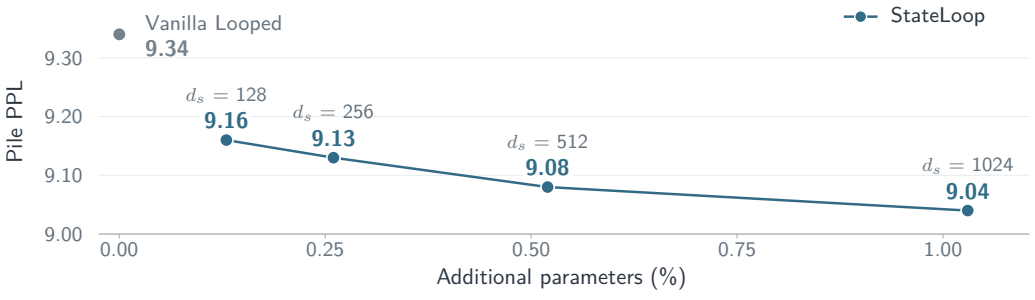


Figure 3: **State width at 410M ($K = 7$, 25B tokens).** Pile PPL versus parameter overhead for identity mixing and backbone width $d = 1024$. Points are labeled with state width d_s and PPL; overhead is relative to the 406.90M-parameter Vanilla Looped baseline.

Halving the state width preserves most of the PPL gain while roughly halving the added parameter cost (Figure 3). Even the narrowest state improves over Vanilla Looped, showing that a compact persistent state is sufficient to provide gains.

Table 4: **Persistent-state ablations on Pile at 410M.** All models use $K = 7$ and 25B training tokens. State variants use $d_s = 1024$ and identity mixing. State-only feedback uses the recurrent state as the next backbone input, whereas reset state sets the previous state to zero before each update. The GRU variant replaces the selective-SSM update while keeping feedback and final readout unchanged. Parameter counts are in millions.

Ablation	Variant	Params (M)	PPL ↓
Feedback	State only	409.00	9.44
	StateLoop	411.10	**9.04**
Persistence	Reset state	411.10	9.25
	Persistent state	411.10	**9.04**
Update rule	Vanilla Looped	406.90	9.34
	StateLoop (GRU)	416.34	9.09
	StateLoop	411.10	**9.04**

5.2 Persistent-State Design

We examine which aspects of the persistent-state design contribute to these gains, considering feedback design, state persistence, and the update rule. All models are trained from scratch at 410M scale with $K = 7$ iterations and 25B training tokens. The StateLoop variants use identity mixing.

Feedback design. The state-only alternative uses a gated state itself as the next backbone input Z^{k+1}. From $X^0 = 0$ and $H^0 = T(E)$, its recurrence for $k = 0, \ldots, K-2$ is

$$\begin{aligned} X^{k+1} &= \mathrm{Cell}(H^k, X^k), \\ Z^{k+1} &= X^{k+1}, \qquad H^{k+1} = T(Z^{k+1}). \end{aligned} \tag{13}$$

Here $X^k \in \mathbb{R}^{n \times d}$ has the backbone width, and Cell applies the selective-SSM update in Equation 6 at this width, with H^k as its input and X^k as its previous state. In StateLoop, Z^{k+1} instead combines H^k with a projected read from the additional persistent state, as in Equation 4.

StateLoop achieves lower PPL than state-only feedback (Table 4). This result suggests an advantage for retaining the direct connection between iterations and adding a state read, rather than replacing that path with the recurrent state.

State persistence. We test persistence by setting the old state to zero before every update. The reset and persistent variants follow

$$\begin{aligned} \text{Reset state} \quad & S^{k+1} = \eta^k \odot P^k, \\ \text{Persistent state} \quad & S^{k+1} = (1 - \eta^k) \odot S^k + \eta^k \odot P^k. \end{aligned} \tag{14}$$

Resetting the state still improves over Vanilla Looped (9.25 vs. 9.34), showing that the state branch within each iteration can improve performance even without state persistence across iterations. Carrying the state across iterations yields a further improvement to 9.04 under the same parameter count and training setup, isolating an additional benefit from persistence (Table 4).

State update rule. We retain the same projected input $P^k = U^k W_u^\top$ and replace the selective-SSM update in Equation 6 with a GRU (Cho et al., 2014), keeping the feedback, final readout, and identity mixer unchanged. The GRU variant preserves most of the improvement over Vanilla Looped (Table 4), indicating that the benefit is not specific to the selective-SSM parameterization. The selective-SSM update achieves lower PPL with fewer parameters.

5.3 Analysis of the Persistent State

We analyze a trained 410M StateLoop with identity mixing and $K = 7$ on the full Pile validation set to examine how the persistent state supports later iterations and why continued state updates matter. All analyses share an unmodified baseline of 9.04 PPL.

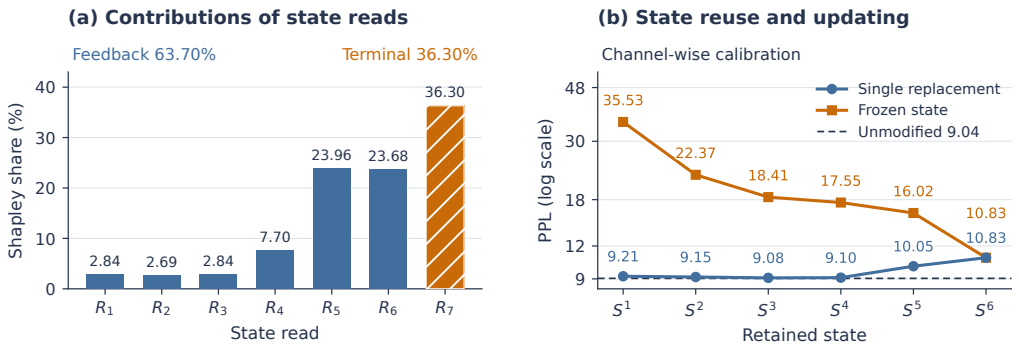


Figure 4: **State analysis at 410M ($K = 7$, identity mixing).** Evaluated on the full Pile validation set. (a) Shapley shares of the NLL increase from masking reads. (b) Calibrated single replacement versus frozen-state reuse; the dashed line marks the unmodified model.

State readouts. We mask a state read by setting its projected output to zero. To account for interactions, we evaluate all 2^7 masking combinations. Each read's exact Shapley value (Shapley, 1953) averages the extra NLL increase from masking it across all removal orderings. Feedback reads account for 63.70% of the attributed NLL increase, and the terminal read accounts for 36.30% (Figure 4a). Contributions concentrate in R_5, R_6, R_7, indicating greater reliance on the state in the final backbone transitions and prediction.

State replacement and freezing. To test whether earlier states remain usable at later boundaries, we first account for differences in channel scale and offset. Let $\mu_{i,c}$ and $\sigma_{i,c}$ be the mean and standard deviation of channel c at boundary i, estimated from unmodified model evaluations on a separate calibration sample. To use S^i at boundary j, we calibrate it as

$$\widehat{S}^{i\to j}_{t,c} = \frac{\sigma_{j,c}}{\sigma_{i,c} + \epsilon}\left(S^i_{t,c} - \mu_{i,c}\right) + \mu_{j,c}. \tag{15}$$

Here $\epsilon = 10^{-6}$ ensures numerical stability. Calibration applies the same scale and offset to all token positions t within each channel.

Single replacement reuses an earlier state once by replacing S^{i+1} with $\widehat{S}^{i\to i+1}$, then resumes normal updates. Freezing reuses the same saved S^i at all remaining boundaries $j > i$, recalibrating it to each target boundary. Both interventions continue running the backbone.

Single replacement causes small PPL increases at early boundaries but larger increases at S^6 and S^7, indicating that later updates become more important for prediction. Replacing S^4 once with calibrated S^3 yields 9.08 PPL, whereas freezing at S^3 raises PPL to 18.41 (Figure 4b). This contrast suggests that earlier states remain partially usable at nearby boundaries, while continued state updates are important as the backbone computation proceeds.

6 Conclusion

StateLoop equips each token with one additional persistent state to retain information across iterations. It reduces perplexity and improves average downstream accuracy over Vanilla Looped, with optional state mixers based on causal attention or Mamba-2 providing further downstream gains. The gains obtained with both selective-SSM and GRU updates support the persistent-state design. Its growing perplexity advantage over Vanilla Looped at greater trained depths further highlights the value of persistent state for deeper recurrent computation. State interventions show that earlier states remain partially usable at later boundaries, while continued state updates are important for prediction. These findings support explicit persistent state as a promising design choice for looped Transformers.

Our experiments cover models up to the 1.4B scale, with the controlled loop-depth analysis conducted at 410M. Extending this analysis to larger models can further characterize how persistent state interacts with recurrent depth.

AI USE STATEMENT

Generative AI tools were used for literature search, language editing, and figure preparation. The authors conceived the research, designed and conducted the experiments, and verified the citations, claims, and reported results against primary sources and experiment records. The authors take full responsibility for the final manuscript.

REFERENCES

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. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, volume 35, pp. 11079–11091, 2022. doi: 10.52202/068431-0805. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/47e288629a6996a17ce50b90a056a0e1-Abstract-Conference.html.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

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

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://iclr.cc/virtual/2019/poster/1068.

Markus Frey, Behzad Shomali, Ali Hamza Bashir, David Berghaus, Joachim Koehler, and Mehdi Ali. Adaptive loops and memory in transformers: Think harder or know more? In *Latent & Implicit Thinking Workshop at ICLR 2026*, 2026. URL https://arxiv.org/abs/2603.08391.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. URL https://arxiv.org/abs/2101.00027.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1082. URL https://aclanthology.org/D17-1082/.

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

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4336. URL https://papers.nips.cc/paper_files/paper/2024/hash/f67449c7ab72f441d3a713b046c6818c-Abstract-Conference.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language-models.pdf.

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

Grigory Sapunov. Universal transformers need memory: Depth-state trade-offs in adaptive recursive reasoning. *arXiv preprint arXiv:2604.21999*, 2026. URL https://arxiv.org/abs/2604.21999.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=din0lGfZFd.

Lloyd S. Shapley. A value for n-person games. In Harold W. Kuhn and Albert W. Tucker (eds.), *Contributions to the Theory of Games II*, pp. 307–317. Princeton University Press, Princeton, NJ, 1953.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

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. Association for Computational Linguistics, 2017. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.

Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, Jiaang Li, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. MeSH: Memory-as-state-highways for recursive transformers. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IhTrFvY7p3.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026. URL https://arxiv.org/abs/2604.21254.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. PonderLM: Pretraining language models to ponder in continuous space. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=UrM4MNRYZm.

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/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. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.