LOOPING BEYOND TWICE: A SCALABLE RECIPE FOR LOOPED MIXTURE-OF-EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers introduce recurrent depth as a new scaling axis for large language models (LLMs): by repeatedly applying shared Transformer blocks, they increase effective depth without increasing parameter count. However, the benefits of looping remain unclear for large MoE LLMs under FLOPs-matched comparisons. The main reason is that the gains from additional iterations diminish quickly and can even turn into degradation, so the extra FLOPs spent on looping yield little substantial improvement. Consequently, prior work typically settles on two loops. In this paper, we take a closer look at why looped MoE fails to scale and identify two main obstacles. First, looping inherits and amplifies the *curse of depth*: hidden-state variance grows with each iteration as residual updates accumulate, which destabilizes deep recurrence and causes representations to drift. Second, looped MoE suffers from *expert selection collapse*: routers repeatedly select the same experts across loops, so extra iterations add computation without adding computational diversity. Guided by this diagnosis, we propose LOOM, built on a single principle: *each loop should contribute new computation while keeping the recurrent state stable.* LOOM *stabilizes* recurrence by scaling residual updates to bound variance growth and re-injecting the input embedding at every loop, and *diversifies* it through per-loop routers that engage different experts and a Looping Residual that carries earlier outputs forward. Experiments across 100M–1.7B models show stable scaling to 9–12 loops, with the 1.7B model scaling to 9 loops. **Under matched FLOPs, the 700M model performs best at 5 loops**, reducing perplexity from 18.36 to 16.54 and improving average zero-shot accuracy from 38.84% to 39.53% over the non-looped baseline. **Without FLOP matching, the 1.7B model trained on 60B tokens peaks at 9 loops**, reducing perplexity from 9.62 to 7.77 and improving average zero-shot accuracy from 42.4% to 47.7%. Code is submitted.

1 INTRODUCTION

Looped Transformers scale the computational depth of LLMs by repeatedly applying shared Transformer blocks, increasing effective depth without increasing parameters (Dehghani et al., 2018; Giannou et al., 2023; Saunshi et al., 2025; Geiping et al., 2025). Unlike conventional depth scaling, looping introduces an additional scaling axis orthogonal to model size and training data (Kaplan et al., 2020; Hoffmann et al., 2022). Recent studies demonstrate that recurrent depth improves algorithmic computation (Giannou et al., 2023), length generalization (Fan et al., 2025), and latent reasoning (Saunshi et al., 2025; Geiping et al., 2025; Hao et al., 2024), establishing looped computation as a promising direction for LLM scaling.

MoE architectures offer a natural pathway for extending looped computation. By activating only a sparse subset of experts per token, MoE models provide additional computation capacity for recurrent refinement under a fixed budget (Shazeer et al., 2017; Fedus et al., 2022). However, current looped MoE systems struggle to benefit from deeper recurrence: increasing the number of loops often leads to loss degradation rather than further improvements. Beyond this regime, additional iterations yield diminishing returns. This has raised concerns about the practical benefits of looped MoE under matched FLOPs, since further increasing the number of loops mainly adds computation without a corresponding performance gain. Consequently, existing approaches often limit recurrence to two iterations (Gao et al., 2026; Wang et al., 2026b; Lab et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

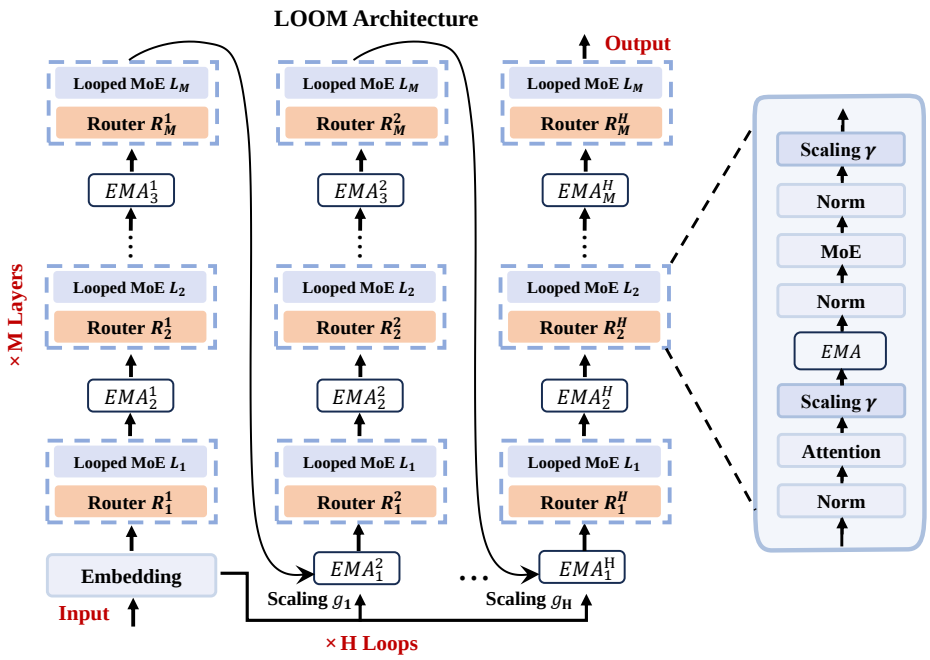


Figure 1: LOOM architecture. An M-layer MoE block is reused for H loops, with λ as a scalar hyperparameter. At layer a of loop t, $\gamma = \lambda/(H\sqrt{M})$ scales residual updates, $g_t = \lambda/(t\sqrt{M})$ controls embedding re-injection, EMA_a^t denotes the Looping Residual, and R_a^t denotes the router.

To understand why deeper recurrence fails to pay off, we investigate the root causes of this limitation and identify two main obstacles to scaling looped MoE. First, looping amplifies the *curse of depth* *Sun et al. (2025)*: repeatedly reusing shared Transformer blocks causes residual accumulation and activation variance growth, as shown in Figure 3, which makes hidden states drift and destabilizes optimization in deep recurrence. Second, looped MoE suffers from *expert selection collapse*: shared routers produce nearly invariant expert assignments across loops, so later iterations repeatedly activate similar experts instead of refining earlier representations (shown in Figure 4). As a result, increasing loop depth adds computation without adding useful computation. This reveals the central challenge of deep looped MoE:

> *Is there an effective way to improve cross-loop information flow in looped MoE, so that recurrence can scale well beyond two iterations?*

To answer this question, we propose LOOM, built on a single principle: *each loop should contribute new computation while keeping the recurrent state stable.* LOOM realizes this principle along two complementary axes, each targeting one of the obstacles identified above.

Diversifying recurrence. To counter expert selection collapse, LOOM equips each loop with its own router, so that different iterations engage different subsets of experts rather than repeatedly activating the same ones. To make these diverse computations build on one another, LOOM further introduces a *Looping Residual*, which carries the outputs of earlier loops forward so that later iterations can refine previous representations rather than overwrite them.

Stabilizing recurrence. To counter the curse of depth, LOOM adopts residual scaling (Sun et al., 2025; Noci et al., 2022; Wang et al., 2026a), scaling the output of each attention and MoE block by $\gamma = \lambda/(H\sqrt{M})$. This bounds residual accumulation and keeps signal propagation stable as the number of loops grows. In addition, to prevent deep recurrence from diluting input information and causing latent drift, LOOM re-injects the input embedding at each iteration (Geiping et al., 2025). To adapt this technique to looped MoE, we recalibrate the injected embedding at step t with $g_t = \lambda/(t\sqrt{M})$, so that the anchoring signal remains well-scaled across iterations.

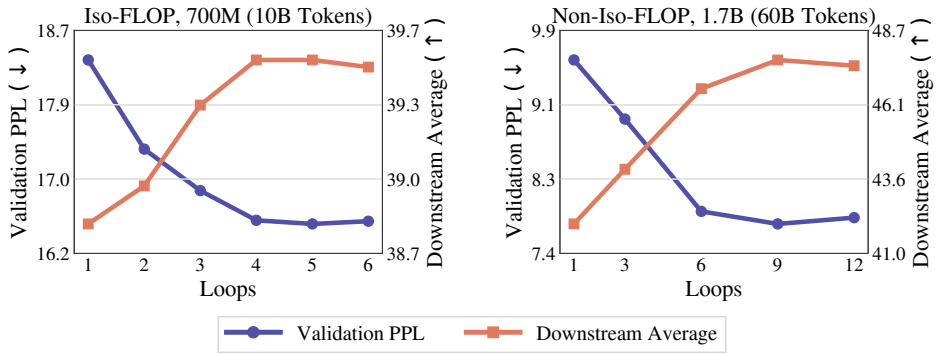


Figure 2: **Loop-depth scaling of LOOM. Left:** under a near-iso-FLOP budget, LOOM achieves near-optimal performance near 5 loops. **Right:** without FLOP matching, the 1.7B model scales to 9 loops, with substantial performance improvements.

Together, these components enable shared experts to progressively refine representations across loops, allowing LOOM to scale recurrence well beyond two iterations without routing collapse or recurrent degradation. Our main contributions are as follows:

❶ We identify two main obstacles that prevent looped MoE from scaling beyond two iterations: the *curse of depth*, in which residual accumulation causes variance growth and latent drift, and *expert selection collapse*, in which shared routers assign nearly identical experts across loops.

❷ We propose LOOM, a principled recipe for deep looped MoE built on a single design principle: each loop should contribute new computation while keeping the recurrent state stable. LOOM stabilizes recurrence through residual scaling and embedding re-injection, and diversifies recurrence through loop-specific routers and a Looping Residual.

❸ Experiments across model scales from 100M to 1.7B parameters and up to 60B training tokens show that LOOM scales to 9–12 loops, and that the 1.7B model scales to 9 loops. Notably, under matched FLOPs, LOOM scales looping to 5 iterations while consistently achieving lower loss, providing strong evidence that recurrent depth is a genuine and compute-efficient scaling axis for MoE LLMs.

2 Related Work

Looped models. Looped LLMs reuse the same Transformer blocks along the depth axis, increasing effective depth without increasing parameter count. This idea traces back to Universal Transformer (Dehghani et al., 2018) and ALBERT (Lan et al., 2019). Recent studies demonstrate the potential of recurrent computation in dense Transformers: Geiping et al. (2025) show that test-time looping improves language models; Saunshi et al. (2025) connect recurrence with latent-space reasoning; and Fu et al. (2026), Wang et al. (2026a), and Movahedi et al. (2026) enable deeper recurrence through stabilization, residual scaling, and fixed-point formulations. HRM and TRM further explore recursive architectures for iterative reasoning (Wang et al., 2025a; Jolicoeur-Martineau, 2025). Chain-of-Experts studies iterative expert computation within each layer rather than recurrently reusing an entire shared MoE block (Wang et al., 2025b). Extending recurrent computation to sparse MoEs introduces additional challenges due to routing dynamics and cross-loop information propagation.

Stability in deep residual stacks. Deep recurrence increases effective depth but can amplify residual dynamics, causing activation instability and loss degradation. Noci et al. (2022) show that rank collapse of token representations causes vanishing gradients in Transformers and can be prevented by scaling residual branches, while Sun et al. (2025) identify the curse of depth in LLMs, where Pre-LN variance growth renders deeper layers nearly identity mappings and propose $1/(\sqrt{l})$ to scale down the residual. To address this, Mix-LN (Li et al., 2024) applies Post-LN to earlier layers and Pre-LN to deeper layers. Another line of work redesigns the residual pathway itself: mHC (Xie et al., 2025) constrains the residual mixing matrices of hyper-connections to a doubly stochastic manifold to restore the identity-mapping property and stabilize large-scale training, and Attention Residuals (Team et al., 2026) replace fixed additive accumulation with attention over earlier layer outputs, allowing each layer to selectively aggregate previous representations. Residual scaling has

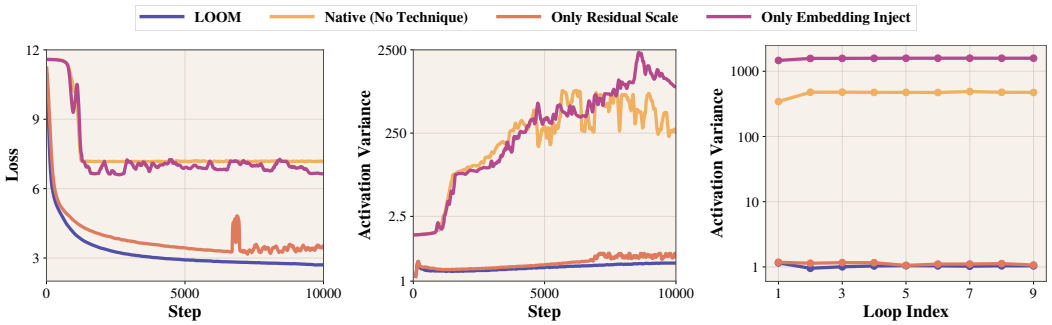


Figure 3: Effect of residual scaling and embedding re-injection on training stability in a 9-loop MoE model ($\sim$350M parameters, 10B training tokens). **Native (No Tech)** loops the unmodified backbone without additional techniques. **Only Res Scale** and **Only Embed Inject** add residual scaling and embedding re-injection, respectively, to Native, with no other modifications. (a) Training loss. (b) Activation variance during training. (c) Activation variance across recurrent iterations.

also been applied to recurrent models: Wang et al. (2026a) introduce a loop-aware factor $\lambda/(H\sqrt{M})$, and Movahedi et al. (2026) combine residual scaling with inter-loop mixing. Input re-injection further alleviates representation drift (Geiping et al., 2025). These techniques lay the foundation for stabilizing deep recurrence and motivate their adaptation to looped MoE.

Looped MoE. Sparse gating enables MoE models to scale parameter capacity under limited computation, but introduces additional challenges for recurrent computation, particularly in expert routing and cross-loop information propagation (Shazeer et al., 2017; Fedus et al., 2022). Existing recurrent MoE studies have explored limited forms of recurrence. MoEUT extends Universal Transformer with MoE layers rather than stabilizing deep full-block recurrence (Csordás et al., 2024); Chain-of-Experts iterates experts within layers instead of reusing a shared MoE block (Wang et al., 2025b); SMELT studies compute-matched MoE looping and identifies a two-loop optimum (Wang et al., 2026b). LoopMoE introduces iteration-adaptive normalization, while Lee et al. (2026) study recurrent MoE models with varying expert utilization but maintain effective depth by reducing layers as loop counts increase (Chen et al., 2026; Lee et al., 2026). These approaches demonstrate the potential of recurrent MoE, but scalable deep recurrence under a fixed architecture remains underexplored.

3 Methodology

Overview. LOOM repeatedly applies an M-layer, pre-norm MoE decoder. Each loop applies layers $\ell = 1, 2, \ldots, M$ in order and repeats this walk for H passes. The attention and expert weights are shared across loops, whereas each loop has its own routers. The resulting effective depth is MH; increasing H adds router parameters but does not replicate the attention or experts. Let $x = \mathrm{Embed}(\mathrm{tokens})$ and let h_ℓ^t denote the residual stream after layer ℓ of loop t, with $h_0^1 = x$. The final state h_M^H is used for next-token prediction.

3.1 Stabilizing the recurrent state

Residual scaling. Deep recurrence amplifies the curse of depth (Sun et al., 2025): correlated residual updates accumulate across loops, increasing hidden-state variance and destabilizing signal propagation. Figure 3(b,c) shows that unscaled residual accumulation leads to activation variance explosion during training and across loops, whereas residual scaling effectively controls this growth. Accordingly, LOOM incorporates residual scaling that accounts for both loop count and physical depth (Wang et al., 2026a):

$$\gamma = \frac{\lambda}{H\sqrt{M}}, \quad \widetilde{h}_\ell^t = h_{\ell-1}^t + \gamma\,\mathrm{Attn}_\ell(\mathrm{RMS}(h_{\ell-1}^t)), \quad h_\ell^t = \widetilde{h}_\ell^t + \gamma\,\mathrm{RMS}(\mathrm{MoE}_\ell^t(\mathrm{RMS}(\widetilde{h}_\ell^t))). \quad (1)$$

The $1/H$ factor limits residual accumulation across loops, while $1/\sqrt{M}$ controls variance growth across physical layers. LOOM fixes $\lambda = 0.5$ as a non-learnable hyperparameter and applies γ uniformly across layers and loops. Scaling the branch outputs preserves the identity path, while the additional MoE output normalization controls the magnitude of its residual contribution.

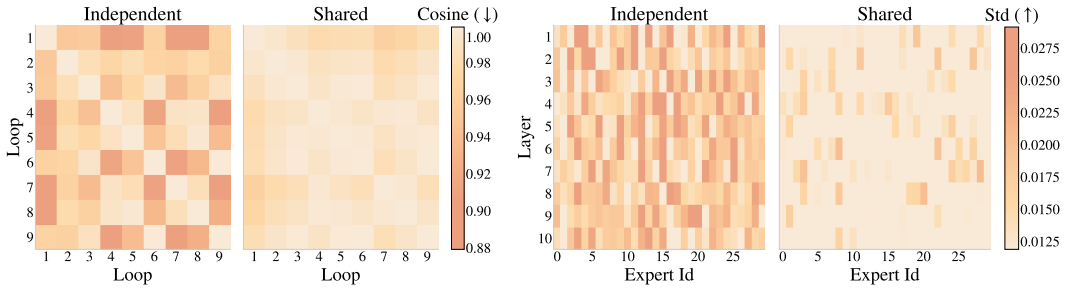


Figure 4: Expert routing in a 9-loop MoE model (∼350M parameters). (a) Layer-averaged cosine similarity of expert loads across loops. (b) Cross-loop standard deviation of expert loads. Loop-specific routers yield more diverse expert assignments than shared routers.

Embedding re-injection. Residual scaling alone substantially reduces activation variance but yields higher training loss than the complete recipe (Figure 3), indicating that variance control alone is insufficient. Deep recurrence can also dilute input information and induce representation drift (Wang et al., 2026a). To preserve access to the original token representation, LOOM re-injects the input embedding at the beginning of each loop $t \geq 2$:

$$g_t = \frac{\lambda}{t\sqrt{M}}. \qquad h_0^t = (1 - g_t)h_M^{t-1} + g_t x, \tag{2}$$

We use the same fixed $\lambda = 0.5$ as in residual scaling. Re-injection begins at $t = 2$, since $h_0^1 = x$. The decreasing coefficient g_t places progressively greater weight on the recurrent representation, while convex mixing regulates the embedding contribution. Removing embedding re-injection from LOOM increases validation perplexity from 18.62 to 26.05 at $H = 9$ (Table 5), supporting its complementary role in anchoring recurrent states alongside residual scaling.

3.2 Diversifying computation across loops

Loop-specific routers. Deep recurrence should enable successive loops to perform complementary computations. However, sharing routers across loops can produce nearly invariant expert selections, leading to *expert selection collapse* and limiting the benefits of additional iterations. LOOM addresses this limitation through **loop-specific routing with shared experts**: each layer uses a separate router for each loop, while its expert weights remain shared. This decouples routing decisions across iterations, enabling different stages of recurrence to access different expert combinations without replicating expert parameters. Figure 4 provides two complementary observations: **(i)** lower cross-loop cosine similarity of expert-load distributions and **(ii)** greater variation in expert loads across loops. Together, these observations support more diverse expert utilization, a prerequisite for complementary computation across recurrent iterations.

Looping Residual. Loop-specific routers diversify computation across iterations, while progressive refinement additionally requires effective reuse of earlier computations. LOOM therefore introduces a **Looping Residual** based on a **fixed-decay exponential moving average (EMA)**, with two complementary memories: **(i)** a global memory spanning layers and loops, and **(ii)** a local memory aggregating attention outputs within the current loop. For layer ℓ in loop t, let $o_\ell^t = \gamma\,\mathrm{Attn}_\ell(\mathrm{RMS}(h_{\ell-1}^t))$ denote the scaled attention output. Each memory maintains an accumulator N_A and a normalization factor D_A:

$$N_A \leftarrow \beta N_A + o_\ell^t, \quad D_A \leftarrow \beta D_A + 1, \quad r_A = \frac{N_A}{D_A}, \qquad A \in \{H, L\}. \tag{3}$$

Both pairs are initialized to zero, with fixed $\beta = 0.5$. The global pair persists across the full recurrence, whereas the local pair is reset at the beginning of each loop. Normalization by D_A yields exponentially weighted summaries that emphasize recent computations while retaining earlier contributions. Crucially, **each memory stores only one accumulator tensor and one scalar, regardless of the number of loops**. This recursive formulation avoids storing the output history and requires only elementwise operations per update, introducing negligible additional memory and

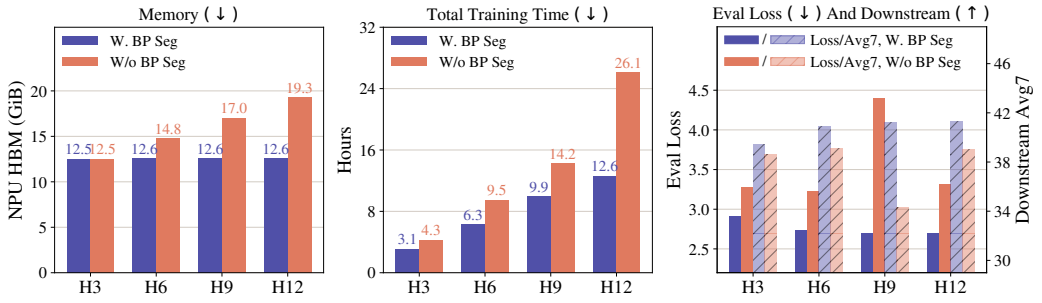


Figure 5: Effect of segmented backpropagation (K=3) on the ∼350M model, compared with backpropagation through all loops. (a) Peak NPU memory measured with a microbatch size of 1. (b) Total training times. (c) Held-out evaluation loss (solid bars) and average accuracy across seven downstream tasks (hatched bars).

Algorithm 1 Training procedure of LOOM. $\mathrm{sg}(\cdot)$ stops gradient flow.

Require: Loop count H, depth M, segment length $K = 3$, $\lambda = \beta = 0.5$

$x, h_0^1 \leftarrow \mathrm{Embed}(\text{tokens})$; $\gamma \leftarrow \lambda/(H\sqrt{M})$; init shared LR $(N_H, D_H) \leftarrow (0, 0)$

for $t = 1, \ldots, H$ **do**

  if $t \geq 2$ **then** $g_t \leftarrow \lambda/(t\sqrt{M})$; $h_0^t \leftarrow (1 - g_t)h_M^{t-1} + g_t x$ ▷ equation 2

  $(N_L, D_L) \leftarrow (0, 0)$ ▷ Reset local memory

  for $\ell = 1, \ldots, M$ **do**

    $o_\ell^t \leftarrow \gamma\,\mathrm{Attn}_\ell(\mathrm{RMS}(h_{\ell-1}^t))$; update both memories via equation 3

    $\widetilde{h}_\ell^t \leftarrow h_{\ell-1}^t + r_H + r_L$; $h_\ell^t \leftarrow \widetilde{h}_\ell^t + \gamma\,\mathrm{RMS}(\mathrm{MoE}_\ell^t(\mathrm{RMS}(\widetilde{h}_\ell^t)))$

  if $t \bmod K = 0$ **or** $t = H$ **then**

    $\hat{y}^t \leftarrow \mathrm{LMHead}(\mathrm{RMS}(h_M^t))$; backward; optimizer step

    $(h_M^t, N_H, D_H) \leftarrow \mathrm{sg}(h_M^t, N_H, D_H)$ ▷ Stop gradient

computational overhead relative to the backbone. The summaries replace the direct attention residual update in Equation 1:

$$\widetilde{h}_\ell^t = h_{\ell-1}^t + r_H + r_L. \tag{4}$$

The global summary r_H and local summary r_L aggregate attention outputs across loops and within the current loop, respectively, preserving intermediate computations for recurrent refinement. The resulting state $\widetilde{h}_\ell^t$ is subsequently processed by the MoE residual update defined in Eq. equation 1.

3.3 Training with Segmented Backpropagation

LOOM uses segmented backpropagation, dividing the H recurrent loops into consecutive segments of at most $K = 3$ loops. Each segment receives supervision from a language-modeling loss at its final loop, with gradients restricted to that segment. The residual state and global Looping Residual memory are then propagated to the next segment with gradients detached, preserving forward information across all H loops while limiting backpropagation depth.

Figure 5 compares this strategy with backpropagation through all H loops on the ∼350M model, highlighting two benefits. ❶ Lower memory consumption and faster training. From $H = 3$ to $H = 12$, segmentation maintains peak memory near 12.6 GiB, whereas unsegmented memory increases from 12.5 to 19.3 GiB. At $H = 12$, the reported training time decreases from 26.1 to 12.6 hours. ❷ Improved stability and performance. Segmented runs achieve lower evaluation loss and higher average accuracy across seven downstream tasks. At $H = 9$, the unsegmented run exhibits severe degradation, with evaluation loss reaching approximately 4.4 and average accuracy falling to approximately 34%. These benefits in speed and stability motivate using segmented backpropagation with K=3 by default in LOOM. Detailed comparisons in Table 8 further support using segmented backpropagation by default in LOOM.

Algorithm 1 summarizes the forward computation and segmented training procedure of LOOM.

Table 1: MoE layout and loop-specific knobs. Effective depth is MH.

Parameters (activated)	Eff. depth	Routed / shared	Top-k	$\lambda/\beta/K$	Peak LR	Tokens
$\sim$100M ($\sim$70M)	$6H$	24 / 2	6 (4+2)	0.5/0.5/3	4×10^{-4}	$\sim$5B
$\sim$350M ($\sim$141M)	$10H$	30 / 2	8 (6+2)	0.5/0.5/3	2×10^{-4}	$\sim$10B
$\sim$1.7B ($\sim$0.63B)	$15H$	30 / 2	8 (6+2)	0.5/0.5/3	6×10^{-5}	$\sim$60B

4 EMPIRICAL RESULTS

In this section, we present the experimental setup (Section 4.1), loop-depth results from $\sim$100M to $\sim$1.7B (Section 4.2), and ablations of the recipe (Section 4.3).

4.1 EXPERIMENTAL SETUP

Models and datasets. LOOM uses a Llama-style pre-norm decoder (Xiong et al., 2020; Touvron et al., 2023) with grouped-query attention (Ainslie et al., 2023), RoPE (Su et al., 2024) ($\theta = 10^4$), and sparse SwiGLU MoE layers (Shazeer, 2020; Dai et al., 2024; Jiang et al., 2024). **Each loop applies layers $\ell = 1, 2, \ldots, M$ in order and repeats this walk for H passes.** Models are pretrained on FineWeb-Edu (Penedo et al., 2024) with sequence length 1,024 and a 65,536-token vocabulary. Table 1 summarizes the training settings; Appendix A provides architectural details.

Hyperparameters. We train all models with AdamW (Loshchilov & Hutter, 2017) ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), gradient clipping at 1.0, and a 10/80/10 warmup–stable–decay schedule (Hu et al., 2024) with a $0.1\times$ peak learning-rate floor. The global batch is 1,024.

Baselines. We compare LOOM with a non-looped $1\times$ baseline and three recurrent baselines matched in backbone, training data, and loop count. ❶ "No tech" loops the unmodified backbone without additional techniques. ❷ "Res. scale" builds on this native loop by applying only residual scaling. ❸ "Embed inject" builds on this native loop by applying only embedding re-injection. As both No tech and Embed inject become unstable at relatively low loop counts, we provide their full results in Section B.1 (Table 7).

Zero-shot commonsense reasoning. We evaluate pretrained checkpoints with standard zero-shot prompts on seven tasks: PIQA (Bisk et al., 2019), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-c and ARC-e (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). Unless noted otherwise, we report length-normalized accuracy (acc_norm) for OBQA, ARC-c, ARC-e, HellaSwag, and PIQA, and raw accuracy (acc) for WinoGrande and SIQA.

4.2 LOOP-DEPTH EXPERIMENTS

We evaluate loop depth under two protocols: near-iso-FLOP comparisons with a non-looped baseline (Section 4.2.1) and non-iso-FLOP sweeps with a fixed MoE layout, where effective depth and compute grow with H (Section 4.2.2).

4.2.1 ISO-FLOP COMPARISON

We design a near-iso-FLOP experiment to test whether a looped LOOM outperforms a non-looped one at matched compute. The backbone has M=10 layers, hidden size d=512, expert width I=d, and a pool of E=80 routed experts (no shared experts); we vary H and shrink top-k, while equivalent depth MH still rises from 10 to 60. **These runs use full backpropagation through all loops, without segmented backpropagation**; Consequently, the comparison incurs no additional LM-head evaluations or backward passes associated with segmentation. We count multiply–adds in units of d^2 per token per stacked layer. With I=d, grouped-query qkv costs $3d^2$, causal SDPA costs $2d^2$, and o_{proj} costs $1d^2$, so attention is $6d^2$. Each routed SwiGLU expert is three $d\times d$ maps and costs $3d^2$; k experts therefore cost $3kd^2$. Over H loops the unit is

$$f(H, k) = H(6 + 3k) = 3H(2 + k). \tag{5}$$

Table 2: Near-iso-FLOP comparison on the M=10, E=80 backbone. All models have 700M parameters and are trained on 10B tokens. Per-token layer cost is $f = H(6 + 3k)$ in units of d^2, anchored at $f_1 = 84$. Best results are bold.

Method	Loops	Eff. depth	top-k	$f(H,k)$	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
Baseline	1×	10	26	84	18.36	28.40	51.38	22.95	43.10	29.43	35.72	60.88	38.84
LOOM	2×	20	12	84	17.37	30.60	49.33	24.32	44.57	29.98	34.65	59.58	39.00
LOOM	3×	30	8	90	16.91	30.20	50.38	**25.00**	44.02	30.05	35.18	60.55	39.34
LOOM	4×	40	5	84	16.58	30.20	50.28	24.74	44.11	30.14	**36.01**	**61.26**	**39.53**
LOOM	5×	50	4	90	**16.54**	**30.80**	50.20	24.91	**44.70**	**30.46**	34.65	60.99	**39.53**
LOOM	6×	60	3	90	16.57	30.40	**52.17**	24.66	44.02	29.91	35.16	60.17	39.50

Table 3: Non-iso-FLOP loop-depth sweeps at ∼100M and ∼350M scales. We report validation perplexity (↓) and zero-shot accuracy (%, ↑). **Results for Embed inject and No tech, which become unstable even at relatively low loop counts, are reported in Section B.1 (Table 7).** Best results within each scale are bold; † denotes an unrecoverable loss spike.

Model size	Method	Loops	Eff. depth	top-k	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
	Baseline	1×	6	6	26.26	27.20	48.46	22.95	40.07	26.85	34.54	57.02	36.73
	Res. scale	3×	18	6	28.99	26.80	50.12	23.29	37.46	26.64	34.29	56.37	36.42
	Res. scale	6×	36	6	34.67†	25.20	48.62	25.17	25.88	25.84	34.14	48.59	33.35†
	Res. scale	9×	54	6	33.78†	27.80	47.75	**26.11**	26.85	25.42	32.09	49.89	33.70†
∼100M	Res. scale	12×	72	6	35.64†	26.40	48.54	25.26	26.64	25.88	33.52	49.80	33.72†
	LOOM	3×	18	6	23.21	28.00	49.49	24.15	39.81	27.11	33.21	57.73	37.07
	LOOM	6×	36	6	19.81	29.20	**54.06**	23.29	42.30	28.26	**35.57**	59.90	38.94
	LOOM	9×	54	6	19.29	**31.00**	50.91	24.32	43.35	**28.75**	35.26	**60.17**	**39.11**
	LOOM	12×	72	6	**19.26**	30.80	50.49	25.17	**44.23**	28.66	35.21	59.19	**39.11**
	Baseline	1×	10	8	20.07	28.40	48.54	23.04	42.42	28.43	34.80	60.07	37.96
	Res. scale	3×	30	8	23.80	28.40	50.99	22.44	40.07	27.26	34.44	57.56	37.31
	Res. scale	6×	60	8	23.40	27.60	51.07	22.35	39.27	27.67	34.60	56.47	37.00
	Res. scale	9×	90	8	33.89†	24.80	47.59	24.66	26.09	25.53	32.55	48.59	32.83†
∼350M	Res. scale	12×	120	8	42.15†	26.20	50.83	22.18	34.93	25.76	34.90	54.90	35.67†
	LOOM	3×	30	8	18.27	31.20	**51.70**	24.15	44.19	29.04	35.11	60.28	39.38
	LOOM	6×	60	8	15.35	32.20	50.43	26.11	47.98	31.59	36.13	61.70	40.88
	LOOM	9×	90	8	**14.80**	32.00	50.12	**27.22**	47.18	32.41	**36.90**	**62.24**	41.15
	LOOM	12×	120	8	14.86	**32.40**	51.14	26.43	**48.23**	**32.78**	36.80	62.13	**41.42**

We choose top-k so that $f(H,k) \approx f_1 = 84$ (the non-looped anchor). Table 2 reports the compute budget of each loop count together with the experimental results. Val.ppl falls from the non-looped Baseline through Loop4, so four loops already improve on Loop2, Loop3, and Baseline (no-loop). Loop5 and Loop6 then stay with Loop4 rather than slipping. Under this near-iso-FLOP budget, LOOM performs best at five loops, which we therefore adopt as its compute-matched operating point. This already extends well beyond the two-loop setting adopted by prior looped models (Gao et al., 2026; Wang et al., 2026b; Lab et al., 2026).

4.2.2 NON-ISO-FLOP LOOP EXPERIMENTS

To quantify the unconstrained optimum of LOOM, we keep the MoE layout fixed and vary only the number of loops. We sweep $H \in \{1, 3, 6, 9, 12\}$ at ∼100M and ∼350M scales, trained on ∼5B and ∼10B tokens, respectively, allowing effective depth and per-token FLOPs to grow with H.

Table 3 supports two conclusions. ❶ After the equivalent-FLOP comparison of Section 4.2.1, extra loops remain beneficial: both validation perplexity and zero-shot accuracy continue to improve as H increases. ❷ Without a FLOP budget, the best LOOM operating point in this sweep is 12 loops, which attains the highest seven-task average (41.42). The matched-compute peak and the unconstrained peak therefore do not coincide.

4.2.3 SCALING UP TRAINING

To evaluate LOOM at a larger scale, we train a ∼1.7B-parameter model on ∼60B tokens. Table 4 highlights two findings. ❶ Stability in large-scale training. The ∼1.7B-parameter LOOM remains

Table 4: Scaling LOOM to ∼1.7B parameters and ∼60B tokens. We report validation perplexity (↓) and zero-shot accuracy (%, ↑) after annealing.

Method	Loops	Eff. depth	top-k	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
Baseline	1×	15	8	9.62	29.6	50.3	26.7	50.2	37.1	37.2	66.0	42.4
LOOM	3×	45	8	8.94	31.6	51.3	27.2	51.0	39.1	39.1	68.0	43.9
LOOM	6×	90	8	7.91	32.2	51.5	31.1	57.0	46.6	39.5	69.2	46.7
LOOM	9×	135	8	**7.77**	33.0	**53.6**	**31.7**	**57.7**	**48.8**	39.2	70.2	**47.7**
LOOM	12×	180	8	7.84	**34.8**	50.7	30.5	56.9	48.6	**40.0**	**70.7**	47.5

stable over ∼60B training tokens. Nine loops are optimal, at an unrolled scale of roughly 15B parameters (1.7B × 9), achieving 7.77 validation perplexity and 47.7% average zero-shot accuracy. ❷ Scalability with model size. The same framework extends from ∼100M and ∼350M to ∼1.7B parameters without changing its loop-specific hyperparameters, while retaining the benefits of recurrent computation. At the ∼1.7B scale, nine loops reduce perplexity from 9.62 to 7.77 and improve average accuracy from 42.4% to 47.7% over the non-looped baseline. These results demonstrate the scalability of LOOM as MoE parameter counts increase.

4.3 Ablation

Component ablation Table 5 ablates components of the ∼350M, 9-loop LOOM model at step 5,000, supporting two design principles. ❶ Stable recurrent states are essential for deep looping. Removing residual scaling, embedding re-injection, or MoE RMSNorm substantially increases perplexity and reduces average downstream accuracy, underscoring the need to control residual accumulation, preserve input information, and regulate activation magnitudes under repeated block reuse. ❷ Effective recurrence benefits from computational diversity and information preservation. Removing per-loop routing or the Looping Residual worsens both metrics, supporting their roles in diversifying expert computation and retaining information across iterations. Together, these results support our central claim: scaling recurrent depth requires stable hidden states and useful computation from successive loops.

Table 5: Component ablation of the ∼350M LOOM model with 9 loops, evaluated at training step 5,000. Each ablated variant removes one component from the full recipe; "w/o routing refresh" uses a shared router across loops, and "w/o LR" removes the Looping Residual. The best result in each column is shown in bold.

Variant	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
LOOM	**18.62**	**31.4**	51.1	**26.1**	43.3	**29.5**	35.8	61.4	**39.8**
w/o Res. scale	24.63	29.2	50.1	23.8	42.6	29.4	35.9	59.4	38.6
w/o Embed inject	26.05	29.4	50.2	24.3	43.9	29.2	36.2	**61.5**	39.2
w/o MoE RMSNorm	26.50	29.6	**52.1**	24.1	43.6	28.7	35.5	59.4	39.0
w/o Looping Residual	19.25	30.8	49.2	24.2	**44.8**	**29.5**	**36.6**	60.2	39.3
w/o routing refresh	19.83	30.0	51.7	25.4	43.1	29.4	35.7	59.2	39.2

5 Conclusion

This paper presents LOOM, a scalable recipe for extending recurrent depth in MoE LLMs beyond the commonly used two-loop regime. Our analysis identifies hidden-state variance growth and expert selection collapse as key obstacles to deeper recurrence. LOOM addresses these obstacles through residual scaling and embedding re-injection to stabilize recurrent states, together with per-loop routers and a Looping Residual to diversify expert computation and preserve information across iterations. Experiments spanning approximately 100M to 1.7B parameters and up to 60B training tokens show stable scaling to 9–12 loops, with the 1.7B model scaling to 9 loops. Under matched training FLOPs, LOOM scales to five loops while consistently achieving lower loss, providing evidence that recurrent depth offers a compute-efficient scaling axis for MoE language models.

AI USE STATEMENT

Generative AI tools were used solely to assist with writing: polishing the language of author-written text, improving readability, checking the manuscript for consistency and formatting errors, and formatting references. All suggested edits were reviewed and verified by the authors. Generative AI tools were not used for research ideation, formulating hypotheses, designing methods or experiments, implementing code, generating or processing data, analyzing or interpreting results, or creating figures. The authors take full responsibility for the content of this paper.

ETHICS STATEMENT

This work studies architectural and training techniques for looped Mixture-of-Experts language models. All models are trained from scratch on the publicly available FineWeb-Edu corpus and evaluated on standard, publicly available benchmarks. The research does not involve human subjects, crowdsourcing, or the collection of personal data, and we do not release new datasets. As with any web-derived corpus, the pretraining data may contain social biases or inaccurate content that models trained on it can reflect; our models are small-scale research artifacts and are not intended for deployment. Because `LOOM` aims to improve the compute efficiency of language models, we expect its main societal effect to be lowering the cost of training capable models, which may broaden access but, like other efficiency improvements, could also facilitate misuse. We have read and adhere to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We have taken several steps to support reproducibility. The complete source code for training and evaluation is included in the supplementary material. Section 3 specifies all components of `LOOM` (Equations 1–4), and Algorithm 1 gives the full training procedure, including segmented backpropagation. Architecture and optimization settings for every model scale are listed in Table 1 and Appendix A (Table 6), and all loop-specific hyperparameters (λ, β, K) are fixed across scales. The FLOP accounting behind our near-iso-FLOP comparison is derived in Section 4.2.1 (Equation 5). The pretraining corpus (FineWeb-Edu) and all evaluation benchmarks are publicly available, and the evaluation protocol, including the accuracy metric used for each task, is described in Section 4.1.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 4895–4901, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL `https://arxiv.org/abs/1911.11641`.

Wenkai Chen, Tianshu Li, Wenyong Huang, Yichun Yin, Lifeng Shang, and Chengwei Qin. Loopmoe: Unifying iterative computation with mixture-of-experts for language modeling. *arXiv preprint arXiv:2606.04438*, 2026.

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.

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

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

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

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, volume 2025, pp. 14502–14520, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. Simply stabilizing the loop via fully looped transformer. *arXiv preprint arXiv:2605.18797*, 2026.

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

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025.

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

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Nanbeige Lab, Chen Yang, Chengrui Huang, Fufeng Lan, Hanhui Chen, Hao Zhou, Huatong Song, Jiaqi Cao, Jiaying Zhu, Jinlin Niu, et al. Nanbeige4. 2-3b: Unlocking agentic capabilities in a compact model. *arXiv preprint arXiv:2607.22083*, 2026.

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

Ryan Lee, Jacob Biloki, Edward J Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026.

Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-ln: Unleashing the power of deeper layers by combining pre-ln and post-ln. *arXiv preprint arXiv:2412.13795*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 35:27198–27211, 2022.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. The curse of depth in large language models. *Advances in Neural Information Processing Systems*, 38: 163104–163136, 2025.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability. *arXiv preprint arXiv:2606.18524*, 2026a.

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

Zihan Wang, Rui Pan, Jiarui Yao, Robert Csordas, Linjie Li, Lu Yin, Jiajun Wu, Tong Zhang, Manling Li, and Shiwei Liu. Chain-of-experts: unlocking the communication power of mixture-of-experts models. *arXiv preprint arXiv:2506.18945*, 2025b.

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

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

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.

A Details of experiments

Table 6 summarizes the architecture and optimization settings of the three LOOM scales used in Section 4.1. Effective depth is defined as MH, where an M-layer block is recurrently applied for H iterations. The $\sim$100M model uses M=6, hidden size 384, 24 routed experts, and top-k=6. The $\sim$350M and $\sim$1.7B models use 30 routed experts and 2 shared experts, with top-k=8 comprising 6 routed and 2 shared experts. All scales employ loop-specific sigmoid routers in float32, with the same z-loss coefficient, capacity factor, and load-balancing bias learning rate.

Loop-specific coefficients are fixed across model scales, with λ=0.5, β=0.5, and backward segment length K=3. The peak learning rates are 4×10^{-4}, 2×10^{-4}, and 6×10^{-5} for the $\sim$100M, $\sim$350M, and $\sim$1.7B models, respectively, trained on approximately 5B, 10B, and 60B tokens. Model width and expert dimensions scale with model size, whereas the loop-specific hyperparameters remain unchanged.

Table 6: Architecture and optimization settings for the three LOOM model scales. Effective depth is MH, with an M-layer block reused for H recurrent iterations.

Parameters (activated)	$\sim$100M ($\sim$70M)	$\sim$350M ($\sim$141M)	$\sim$1.7B ($\sim$0.63B)
Physical layers M	6	10	15
Effective depth	$6H$	$10H$	$15H$
Hidden size d	384	512	1280
Query / KV heads	6 / 3	8 / 4	20 / 10
Head dimension	64	64	64
Expert width I	384	512	768
Routed / shared experts	24 / 2	30 / 2	30 / 2
Top-k	6(4+2)	8(6+2)	8(6+2)
Router	per-loop sigmoid, float32	per-loop sigmoid, float32	per-loop sigmoid, float32
z-loss / capacity	10^{-3} / 1.5	10^{-3} / 1.5	10^{-3} / 1.5
Load-balance bias LR	2×10^{-3}	2×10^{-3}	2×10^{-3}
$\lambda/\beta/K$	0.5/0.5/3	0.5/0.5/3	0.5/0.5/3
Peak LR	4×10^{-4}	2×10^{-4}	6×10^{-5}
Tokens	$\sim$5B	$\sim$10B	$\sim$60B

Within each model scale, all architectural and optimization settings are held fixed when varying the loop count H. Thus, the loop-depth comparisons in the main text and Appendix B isolate the effect of recurrence depth without changing model width, expert configuration, or learning rate.

B Additional experiments

B.1 Additional baselines for Table 3

Table 7 extends the results in Table 3 by introducing two additional baselines at both $\sim$100M and $\sim$350M scales. The 1$\times$ rows correspond to width-matched non-looped models, while the LOOM rows use the complete recurrent recipe. Following Figure 3, "No tech." directly loops the unmodified backbone without additional techniques, providing a native recurrence baseline. "Embed inject" builds on the same native loop and adds only embedding re-injection, without introducing any other stabilization mechanism. These baselines isolate whether the improvements observed in Table 3 arise simply from increasing recurrence depth or require the full design of LOOM.

Table 7 yields three observations. ❶ **Native looping does not benefit from increasing recurrence depth.** At 3$\times$, "No tech." already underperforms the corresponding 1$\times$ baseline, and by 6$\times$ validation perplexity exceeds 10^3 at both scales, indicating that additional recurrent computation alone does not translate into effective depth. ❷ **Embedding re-injection alone is insufficient to stabilize deep recurrence.** Although it partially mitigates degradation at shallow recurrence, validation perplexity increases from 27.48 to 71.28 between 3$\times$ and 6$\times$ at $\sim$100M, and from 42.76 to 803.6 at $\sim$350M. ❸ **The full LOOM recipe consistently benefits from deeper recurrence.** At $\sim$100M, validation perplexity decreases from 23.21 at 3$\times$ to 19.26 at 12$\times$, while at $\sim$350M it reaches 14.80 at 9$\times$ and the best seven-task average of 41.42 at 12$\times$. Together with Table 3, these results show that the gains from deeper recurrence arise

from the stabilization mechanisms of LOOM, rather than from additional loop iterations alone.

Table 7: Loop-depth ablation of partial stabilization recipes at ∼100M and ∼350M scales. Val.ppl (↓) and zero-shot accuracy (%, ↑) are reported; † denotes an unrecoverable loss spike.

Model size	Method	Loops	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
∼100M	Baseline	1×	26.26	27.20	48.46	22.95	40.07	26.85	34.54	57.02	36.73
	No tech.	3×	27.61	27.80	48.30	21.93	38.89	26.24	34.03	56.96	36.31
	No tech.	6×	1311†	27.40	51.54	24.49	27.53	25.51	34.90	48.91	34.33†
	No tech.	9×	1312†	26.80	47.75	23.63	27.27	25.61	34.90	48.97	33.56†
	No tech.	12×	1587†	27.20	49.33	24.06	28.07	25.34	34.18	49.51	33.96†
	Embed inject	3×	27.48	24.80	52.01	22.27	37.75	26.21	33.73	54.68	35.92
	Embed inject	6×	71.28†	24.60	49.49	21.84	32.03	24.79	34.24	52.72	34.24†
	Embed inject	9×	141.5†	26.40	50.99	21.25	31.31	25.58	34.44	50.92	34.41†
	Embed inject	12×	186.4†	25.80	48.86	22.61	29.84	25.03	33.67	51.37	33.88†
	LOOM	3×	23.21	28.00	49.49	24.15	39.81	27.11	33.21	57.73	37.07
	LOOM	6×	19.81	29.20	**54.06**	23.29	42.30	28.26	**35.57**	59.90	38.94
	LOOM	9×	19.29	**31.00**	50.91	24.32	43.35	**28.75**	35.26	**60.17**	**39.11**
	LOOM	12×	**19.26**	30.80	50.49	**25.17**	**44.23**	28.66	35.21	59.19	**39.11**
∼350M	Baseline	1×	20.07	28.40	48.54	23.04	42.42	28.43	34.80	60.07	37.96
	No tech.	3×	23.75	27.60	51.54	22.10	40.07	26.95	34.75	56.96	37.14
	No tech.	6×	1312†	26.80	49.96	23.89	27.53	25.62	35.31	49.24	34.05†
	No tech.	9×	1313†	26.60	48.07	24.32	27.53	25.69	34.95	49.46	33.80†
	No tech.	12×	1528†	27.40	49.41	23.46	26.92	25.47	34.29	48.75	33.67†
	Embed inject	3×	42.76	26.60	50.99	20.99	35.27	26.30	33.52	54.57	35.46
	Embed inject	6×	803.6†	29.00	50.67	24.83	27.82	25.42	34.60	49.62	34.57†
	Embed inject	9×	780.6†	28.00	47.20	25.34	28.66	25.55	34.65	49.95	34.19†
	Embed inject	12×	864.2†	27.60	49.09	23.98	29.41	25.21	33.98	50.28	34.22†
	LOOM	3×	18.27	31.20	**51.70**	24.15	44.19	29.04	35.11	60.28	39.38
	LOOM	6×	15.35	32.20	50.43	26.11	47.98	31.59	36.13	61.70	40.88
	LOOM	9×	**14.80**	32.00	50.12	**27.22**	47.18	32.41	**36.90**	**62.24**	41.15
	LOOM	12×	14.86	**32.40**	51.14	26.43	**48.23**	**32.78**	36.80	62.13	**41.42**

Table 8: Comparison of segmented backpropagation (K=3) and full backpropagation across loop depths on the ∼350M model.

Loops	Backprop	Val.ppl	OBQA	Wino	ARCC	ARCE	HSwg	SIQA	PIQA	Avg.
3×	K=3	**18.27**	**31.2**	**51.7**	**24.2**	**44.2**	**29.0**	35.1	60.3	**39.4**
	none	26.63	29.2	50.2	23.1	43.7	28.4	**35.2**	**60.8**	38.7
6×	K=3	**15.35**	**32.2**	50.4	**26.1**	**48.0**	**31.6**	**36.1**	**61.7**	**40.9**
	none	25.33	29.2	**51.5**	24.7	43.2	29.0	35.7	60.3	39.1
9×	K=3	**14.80**	**32.0**	**50.1**	**27.2**	**47.2**	**32.4**	**36.9**	**62.2**	**41.1**
	none	81.04†	25.2	47.5	20.6	32.9	25.1	35.1	53.4	34.3†
12×	K=3	**14.86**	**32.4**	51.1	**26.4**	**48.2**	**32.8**	**36.8**	**62.1**	**41.4**
	none	27.52	30.8	**51.8**	23.0	43.2	28.3	35.9	59.8	39.0

B.2 SEGMENTED BACKPROPAGATION

Algorithm 1 adopts segmented backpropagation with segment length K=3, detaching the hidden state between consecutive segments to prevent gradients from propagating through earlier loops. Table 8 compares this strategy with full backpropagation through all H loops on the ∼350M model. The segmented results correspond to the LOOM runs in Table 3, while the full-backpropagation variants use the same architecture, data, and optimization settings. For H=9 and H=12, full backpropagation exceeds the default memory budget, so those training runs use micro-batch size 8 while preserving the global batch size. Segmentation keeps the backward graph inside K loops, so the stored activations do not grow with H. The H=9 and H=12 full-backpropagation entries are therefore not trained

under the same activation memory as the $K{=}3$ runs. This training micro-batch is separate from the peak-memory measurement in Figure 5(a), which uses a microbatch size of 1. The comparison still holds the optimizer, the width, and the global batch fixed, and changes only where the backward graph is cut.

Segmented backpropagation remains stable as the loop count increases, reducing validation perplexity from 18.27 at $3\times$ to 14.80 at $9\times$. In contrast, full backpropagation becomes increasingly unstable at larger loop counts, with the $9\times$ run reaching a perplexity of 81.04 and an average accuracy of 34.3. These results indicate that segmented backpropagation substantially improves the stability and scalability of training at larger recurrence depths; accordingly, all main experiments use $K{=}3$.