LOOP THE LOOPIES!

Anonymous authors
Paper under double-blind review

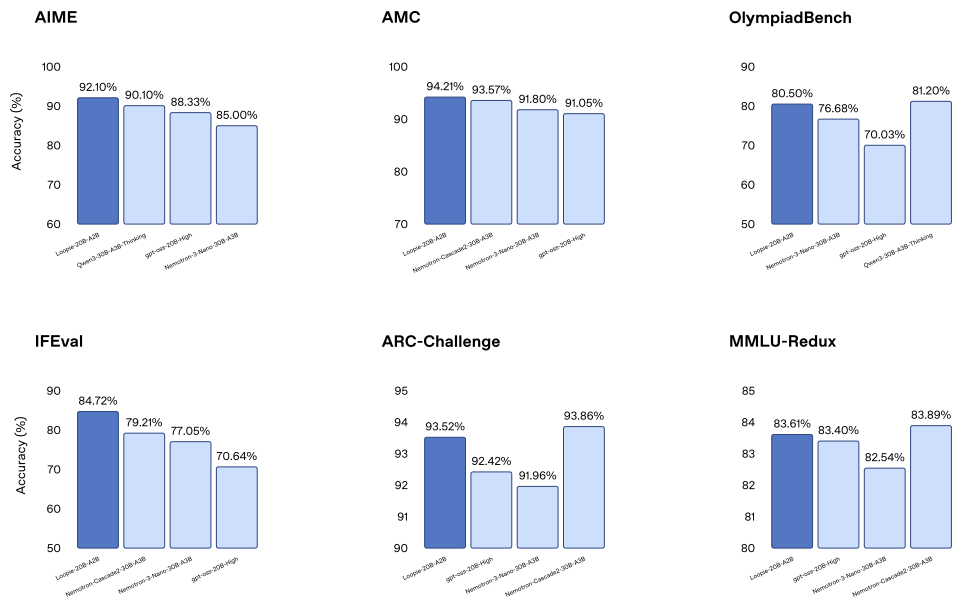
ABSTRACT

We present the Loopie series, consisting of two Mixture-of-Experts (MoE) models: a 20B-parameter model with 2B active parameters and a 6B-parameter model with 0.6B active parameters. Looped Transformers have long faced a challenge: given an N-fold increase in pre-training compute, increasing the parameter count by a factor of N usually outperforms looping a model N times. Loopie addresses this challenge. Extensive ablation studies, including comparisons with a vanilla 30B-A3B model, show that Loopie substantially outperforms vanilla Transformer baselines trained with the same compute budget. With a novel post-training method, Loopie develops strong reasoning abilities and achieves frontier-level reasoning performance.

1 INTRODUCTION

Looped Transformers, first introduced as Universal Transformers (Dehghani et al., 2019), have recently re-emerged as a compelling alternative to conventional depth scaling. Rather than stacking distinct layers, they repeatedly apply the same model across recurrent steps, closely connecting this approach to parameter sharing in Transformers (Lan et al., 2020; Dehghani et al., 2019). This recurrent approach has shown strong empirical performance across a wide range of domains, including language modeling, algorithmic learning, and abstract reasoning (Gao et al., 2025b; Yang et al., 2024; Jolicoeur-Martineau, 2025; Wang et al., 2025; Dehghani et al., 2019; Saunshi et al., 2025; Bae et al., 2025; Geiping et al., 2025; Zhu et al., 2025; Frey et al., 2026a; Huang et al., 2026b).

A growing body of work suggests that recurrent computation is especially well suited to complex problems. Looped Transformers can outperform vanilla Transformers on in-context learning and data-fitting tasks, and they can implement multi-step gradient descent in context with far fewer parameters (Yang et al., 2024; Fan et al., 2025; Giannou et al., 2023; Gatmiry et al., 2024a;b; Chen



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.

et al., 2025). Studies of looped Transformers have also shown that recurrent computation allows shallow parameterizations to approach the performance of deeper untied models on reasoning tasks (Saunshi et al., 2025; Jeddi et al., 2026; Geiping et al., 2025; Zhu et al., 2025). These advantages are particularly visible on abstract reasoning tasks such as ARC-AGI, Sudoku, and Maze, where results from recent recurrent models suggest that recurrence provides a powerful inductive bias for compositional generalization, rule discovery, and inductive reasoning (Wang et al., 2025; Jolicoeur-Martineau, 2025; Gao et al., 2025b).

Recent work has begun scaling looped Transformers to billion-parameter language models. Ouro models scale to 1.4B and 2.6B parameters with four recurrent steps, reporting strong parameter efficiency relative to larger dense baselines (Zhu et al., 2025). Huginn scales to 3.5B parameters and uses 32 recurrent steps to trade parameters for latent computation (Geiping et al., 2025). Other systems further study adaptive recurrence, efficient reasoning, memory–compute trade-offs, and scaling laws for stable looped language models (Bae et al., 2025; Jeddi et al., 2026; Frey et al., 2026a; Prairie et al., 2026). However, these gains expose a fundamental compute-accounting issue: **looping a model N times during pre-training also multiplies pre-training compute by N. Thus, looped Transformers should be compared not only against vanilla Transformers with the same parameter count but also against non-looped models trained under the same pre-training compute budget.** This view is consistent with language-model scaling and compute-optimal training analyses, which evaluate quality as a function of parameters, data, and total training FLOPs rather than parameter count alone (Kaplan et al., 2020; Hoffmann et al., 2022; Prairie et al., 2026). For example, Ouro-2.6B with 4 loops should be compared against a baseline model with a parameter count close to 2.6B × 4 = 10.4B; likewise, after accounting for pre-training compute, Huginn-3.5B with 32 loops should be compared against a much larger 112B-parameter baseline model. This motivates our central question:

Can looped Transformers match or exceed vanilla Transformers under the same pre-training compute budget?

We answer this question with the **Loopie Series**: two looped MoE LLMs, Loopie-20B-A2B and Loopie-6B-A0.6B, each trained with two loop steps. Our key idea is the **Loopie Recipe**, a compute-matched scaling recipe that addresses the main concern for recurrent-depth scaling: under a fixed pre-training compute budget, vanilla parameter scaling can otherwise dominate looping (Hoffmann et al., 2022; Prairie et al., 2026). With a novel post-training method, Loopie develops strong reasoning abilities and achieves frontier-level reasoning performance.

Our contributions are threefold:

- **Compute-matched scaling.** We introduce the Loopie Recipe to address the fixed-compute challenge for looped Transformers and validate it through extensive ablations.
- **Scalable looped MoE models.** We demonstrate that looped computation scales to large MoE language models by training Loopie-20B-A2B and Loopie-6B-A0.6B.
- **Large-scale post-training.** We scale Loopie through large-scale post-training, including a novel supervised pre-training stage, it yields strong reasoning abilities.

2 The Loopie Series

This section presents the Loopie Series, from its architecture to the evidence for its scalability. We begin by defining Loopie's layer-loop recurrence pattern and contrasting it with the classic model-loop pattern used in prior looped language models. We then motivate the design under a fixed pre-training compute budget, where models that use recurrent depth must be compared against vanilla Transformers that spend the same compute on ordinary non-recurrent capacity. Building on this motivation, we introduce the Loopie Recipe, a compute-matched scaling recipe for choosing stored width, stored depth, and recurrent depth. We next present the main compute-matched results, examine whether the advantage persists across model scales, and isolate the contribution of the layer-loop pattern through ablations. Finally, we discuss why Loopie uses only two recurrent steps, refer to Appendix J.1 for the detailed analysis and rationale behind this design..

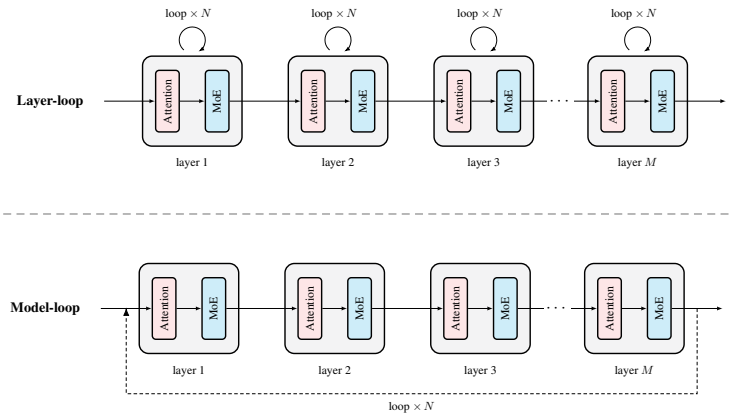


Figure 1: Illustration of the contrast between **layer-loop** and **model-loop**. The top panel shows layer-loop, the loop schedule adopted by Loopie, in which each Attention/MoE layer is applied recurrently before passing its output to the next layer. We demonstrate that this schedule achieves better performance on MoE backbones while also training more efficiently. The bottom panel shows the traditional whole-model recurrence pattern, which we call model-loop, where the entire layer stack is traversed and then repeated. This pattern is used by prior looped models such as Ouro and Huginn (Zhu et al., 2025; Geiping et al., 2025).

2.1 Architecture

Loopie largely follows the Qwen3-MoE architecture. In particular, its backbone is a decoder-only Mixture-of-Experts Transformer, and its attention mechanism, sparsity pattern, and other architectural details remain the same as those of the Qwen3-MoE family. Detailed architectural specifications are provided in Appendix E. The key architectural difference lies in how recurrent computation is applied. Instead of simply repeating the entire model multiple times, Loopie adopts a different recurrence pattern that we call **layer-loop**. Prior looped language models, such as Ouro and Huginn (Zhu et al., 2025; Geiping et al., 2025), mainly use what we call **model-loop**. In model-loop recurrence, the entire Transformer stack is unrolled recurrently: for a model with three layers and two loop steps, the computation order is

$$\text{Layer 1} \rightarrow \text{Layer 2} \rightarrow \text{Layer 3} \rightarrow \text{Layer 1} \rightarrow \text{Layer 2} \rightarrow \text{Layer 3}.$$

By contrast, Loopie uses **layer-loop**, in which each layer is applied recurrently before the computation moves to the next layer. For the same three-layer, two-step example, the computation order becomes

$$\text{Layer 1} \rightarrow \text{Layer 1} \rightarrow \text{Layer 2} \rightarrow \text{Layer 2} \rightarrow \text{Layer 3} \rightarrow \text{Layer 3}.$$

That is, each block performs local iteration on the hidden states and only then passes the resulting recurrent representation to the next layer. Figure 1 illustrates the difference.

2.2 Why Layer-Loop?

Layer-loop is not merely a different ordering of recurrent computation; it changes where iterative refinement happens inside the model and therefore affects scaling behavior, execution efficiency, and the nature of parameter sharing across effective depth. We adopt layer-loop because it offers three advantages that are especially important for large-scale pre-training.

Better empirical scaling. Layer-loop achieves better performance than model-loop in our pre-training experiments. As shown in Figure 2, layer-loop initially trails model-loop slightly on downstream benchmarks. However, it overtakes model-loop after approximately 1.2 trillion training tokens and improves more rapidly thereafter. The early advantage of model-loop therefore does not persist as the training budget increases.

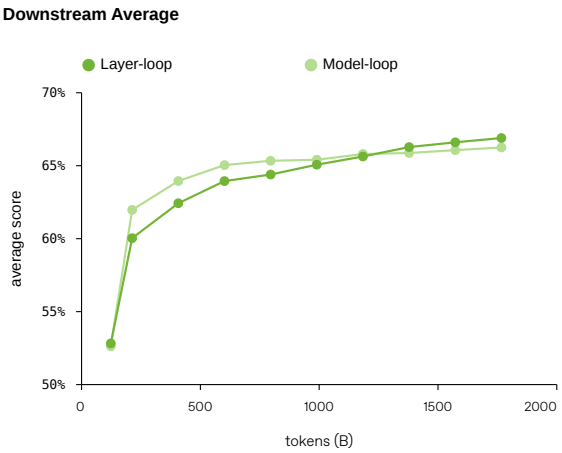


Figure 2: Comparison of the average downstream benchmark scores between the layer-loop and model-loop variants of Loopie-6B-A0.6B. Although layer-loop lags behind model-loop early in training, it surpasses model-loop later.

Infrastructure-friendly execution. Although layer-loop and model-loop have the same nominal layer applications and FLOPs, layer-loop has better execution locality. Repeated uses of a layer are adjacent in the forward and backward graphs, reducing reuse distance and simplifying checkpointing and gradient accumulation, especially with sharding or offloading. It also suits pipeline parallelism because recurrent applications remain within one stage before activations move onward. Model-loop instead repeatedly traverses the full pipeline and feeds the final-stage output back to the first stage, creating cyclic dependencies, extra communication, more pipeline bubbles, and lower utilization.

Natural parameter-sharing pattern. In a 48-layer model such as Qwen3-30B-A3B with two model-loop steps, physical layer 3 is used at effective depths 3 and 51. Prior work shows that Transformer representations vary across depth: lower and final layers differ from middle layers, and different linguistic abstractions emerge at different stages (Sun et al., 2025; Tenney et al., 2019; Jawahar et al., 2019). Although this does not establish gradient conflict, widely separated depths may impose different demands on shared parameters. Model-loop thus shares parameters across heterogeneous depth-dependent roles, whereas layer-loop reuses layers at adjacent depths, giving a more local and coherent sharing pattern.

2.3 Loopie Recipe

We now describe the empirical, hardware-aware scaling procedure used to instantiate Loopie-20B-A2B. Starting from a strong non-recurrent MoE reference, the **Loopie Recipe** comprises three steps:

(i) constructing a recurrent seed model by halving the number of stored layers;
(ii) executing each stored layer twice using *layer-loop*; and
(iii) using the resulting memory headroom to double the per-device microbatch size, then reinvesting the measured training efficiency gain into additional model capacity while keeping the optimizer-step time approximately matched to the reference model.

In this work, we match models by realized pre-training cost rather than by exact theoretical FLOPs. For every comparison, we fix the hardware allocation, sequence length, number of tokens per optimizer step, activation-checkpointing policy, optimizer, and training data. Architectures may use different per-device microbatch sizes according to their measured memory footprints. When the microbatch size is increased, the number of gradient-accumulation steps is reduced proportionally so that the number of tokens per optimizer step remains unchanged.

We select the Loopie configuration whose measured end-to-end optimizer-step time most closely matches the non-recurrent reference. Since both models use the same token budget and number of optimizer updates, this also approximately matches total wall-clock training cost.

The models are not matched by their theoretical FLOP counts. Loopie performs more nominal computation per token, but the additional work is offset by the higher realized efficiency. We refer to this operationally as a compute-matched comparison. Analytical FLOP and memory models are used only to construct and interpret candidate configurations. The final architecture is selected using measured end-to-end training time in Megatron-LM (Shoeybi et al., 2019).

The measured efficiency gain comes solely from the reduced stored depth, which lowers activation memory and enables us to double the per-device microbatch size while halving the number of gradient-accumulation steps.

Activation memory and microbatch efficiency. Under the checkpointing implementation used in our experiments, all recurrent applications of a stored layer are enclosed in the same checkpointed unit. Consequently, the dominant activation memory term scales with stored depth rather than executed depth. The activation memory M_{act} during training scales as (Korthikanti et al., 2023; Rajbhandari et al., 2020)

$$M_{\text{act}} \propto s\, bDL,$$

where D is the hidden dimension, L is the number of stored layers, b is the per-device microbatch size, and s is the sequence length. The recurrent applications increase the amount of executed computation through the loop count R but do not introduce R independently stored sets of layer-boundary activations under this checkpointing scheme.

For a fixed global batch size B, with per-device microbatch size b and g gradient-accumulation steps, we have

$$g = \frac{B}{b}.$$

Reducing DL creates memory headroom that can be used to increase b. Because the number of gradient-accumulation steps g is reduced proportionally, the global batch size remains fixed:

$$b_1 = 2b_0, \qquad g_1 = \frac{g_0}{2}.$$

The larger microbatch exposes more parallel work to each kernel and reduces the number of gradient-accumulation micro-steps required for each optimizer update.

For a fixed architecture, the corresponding microbatch efficiency gain is measured directly as

$$S_{\text{mb}} = \frac{t_{\text{step}}(D, L, R; b, g)}{t_{\text{step}}(D, L, R; 2b, g/2)}.$$

This measured quantity is the efficiency factor used by the Loopie Recipe.

The leading-order Transformer block compute proxy scales as

$$C(D, L, R) \propto LD^2R.$$

For the concise main-text comparison, the Qwen3-like 30B-A3B reference (Qwen Team, 2025) has $(D_0, L_0, R_0) = (2048, 48, 1)$, while applying the Loopie Recipe yields Loopie-20B-A2B with

$$D_1 = 2304, \qquad L_1 = 27, \qquad R_1 = 2.$$

The construction from the Qwen3-like reference, the candidate sweep, and the memory-feasibility derivation are provided in Appendix F.

The selected model has a leading-order Transformer block compute ratio of

$$\frac{C_1}{C_0} = \frac{27 \cdot 2 \cdot 2304^2}{48 \cdot 2048^2} \approx 1.424.$$

Loopie-20B-A2B matches the reference optimizer-step time only after changing the schedule from (b_0, g_0) to $(2b_0, g_0/2)$; thus, the measured efficiency gain is entirely attributable to the doubled microbatch size.

The compute-matching criterion is therefore the directly measured relation:

$$t_{\text{step}}\left(D_1, L_1, R_1; 2b_0, g_0/2\right) \approx t_{\text{step}}\left(D_0, L_0, R_0; b_0, g_0\right),$$

rather than an analytical prediction based on this proxy. The leading-order compute proxy omits lower-order operators, routing and communication costs, optimizer overhead, kernel-launch overhead, and the effect of the microbatch schedule on realized hardware utilization. It is therefore used to characterize nominal work, not to replace end-to-end timing. We evaluated the Loopie Recipe across multiple GPU platforms and observed consistent infrastructure-level gains.

The Loopie Recipe therefore does not claim equality in analytical FLOP estimates. It uses layer-loop recurrence to reduce the stored memory footprint, converts the resulting memory headroom into a doubled per-device microbatch size, spends the measured microbatch efficiency gain on additional model capacity, and selects the final architecture using measured end-to-end optimizer-step time.

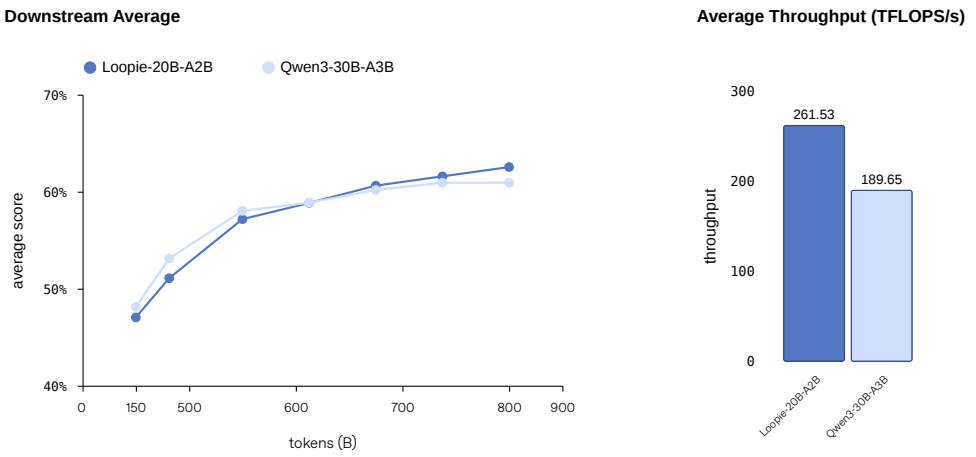


Figure 3: Comparison of Loopie-20B-A2B with our reproduction of Qwen3-30B-A3B under matched per-step pre-training wall-clock time in Megatron-LM. The left panel reports the average score across eight downstream benchmarks, while the right panel reports the highest average pre-training throughput achieved for each model during large-scale benchmarking. In the right figure, Loopie-20B-A2B uses EP = 8 and MBS = 2, while Qwen3-30B-A3B uses EP = 8 and MBS = 1. These configurations achieve the highest throughput in a large-scale TP/PP/EP/MBS grid search; the optimal settings may vary across different GPU.

2.4 Results

To test this principle, we compare Loopie against a strong non-recurrent baseline based on the Qwen3-MoE design. Specifically, we train a vanilla 30B-A3B MoE Transformer with a Qwen3-like architecture on 800 billion tokens under the same pre-training compute budget used for Loopie-20B-A2B. As shown in Figure 3, Loopie initially lags behind the larger vanilla baseline during the early phase of training. However, after roughly 600 billion tokens of pre-training, Loopie-20B-A2B overtakes the compute-matched baseline and maintains a consistent advantage thereafter. This result suggests that, with the right scaling recipe, looped MoE models can achieve stronger final performance by using recurrent computation than by allocating the same compute to a larger vanilla Transformer.

2.5 Scalability

A practical recurrent architecture should not only outperform a single compute-matched baseline but also remain effective as the model is scaled. Many architectural ideas show promising results at one small scale yet fail at larger scales for various reasons. We therefore evaluate whether the Loopie design preserves its advantage across a sequence of increasingly large pre-training runs.

We construct a scaling ladder with four non-recurrent MoE baselines and four compute-matched Loopie models. Each Loopie model is derived from its corresponding baseline using the Loopie

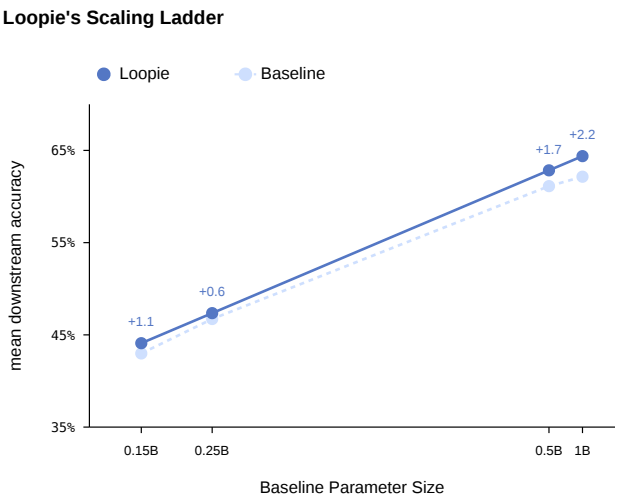


Figure 4: Scaling ladder for Loopie. The horizontal axis denotes the parameter count of the non-recurrent baseline, and the vertical axis reports the average score across eight downstream benchmarks. The Loopie models are not plotted at the parameter counts indicated on the horizontal axis; instead, each Loopie model is sized according to the Loopie Recipe so that its per-step pre-training wall-clock time exactly matches that of the corresponding baseline.

Recipe and applies two layer-loop steps. Its stored width and depth follow the compute-matching principle described in Section 2.3. The objective is not to match stored parameter count, but to match the effective pre-training compute of the corresponding vanilla model. As a result, Loopie uses fewer stored parameters while achieving greater effective depth through recurrence.

Each reference model is trained on a token budget equal to 1000× its active MoE parameter count, ensuring substantial overtraining and largely stabilized downstream performance. The corresponding Loopie model uses the same token budget despite having fewer active parameters. Specifically, the 0.15B baseline and 0.10B Loopie are trained on 150B tokens; the 0.25B and 0.50B rungs use 250B and 500B tokens, respectively; and the 1B baseline and 0.70B Loopie are also trained on 500B tokens due to compute limitations. Architectural details are provided in Appendix Table 7.

Figure 4 shows that Loopie outperforms its compute-matched vanilla counterpart at all four scales, and this advantage does not disappear as model size increases. This indicates that Loopie's gains are not limited to small models, but persist across a broader scaling range. These results show that Loopie delivers consistent scaling gains from 0.15B to 1B under matched pre-training compute, supporting the scalability of its layer-loop design and Loopie Recipe beyond small-model regimes.

2.6 Ablations

In addition to comparing Loopie against a compute-matched vanilla MoE baseline, we further isolate the contribution of the *layer-loop* recurrence schedule. To do so, we conduct a controlled experiment by training a 6B-A0.6B MoE model with the same backbone, optimizer, data mixture, and token budget as Loopie-6B-A0.6B but with the layer-loop pattern removed. This ablated model therefore preserves the overall looped computation budget while testing whether the ordering of recurrent computation is itself important.

Figure 5 shows the resulting average score across eight downstream benchmarks. The layer-loop pattern substantially improves downstream performance relative to the matched ablation. Importantly, because neither the number of active parameters nor the overall computation budget increases, the improvement cannot simply be attributed to additional computation.

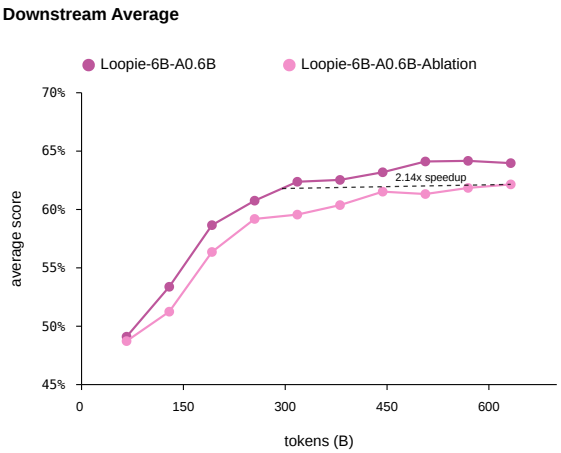


Figure 5: Effect of layer recurrence at fixed stored model size. We compare Loopie-6B-A0.6B, which applies each stored layer twice ($R = 2$), with a non-recurrent variant that applies each layer once ($R = 1$). Both models use the same stored backbone, parameter count, optimizer, data mixture, and training-token budget. Curves report the mean score across the eight downstream benchmarks listed in Appendix D.1. This comparison evaluates the benefit of recurrent computation at fixed stored capacity; it is not a FLOP-matched comparison.

This result supports the architectural choice made in Loopie. While recurrence increases effective depth, the way recurrence is scheduled matters: naively looping computation is insufficient to achieve high compute efficiency. These ablations suggest that Loopie's gains arise not only from using recurrence but from using recurrence in a form that is aligned with the hierarchical structure of Transformer representations.

3 Pre-Training

We pre-train Loopie in two stages: 2.28T tokens from scratch, followed by 1.26T tokens of high-quality annealing focused on synthetic, STEM, code, mathematics, reasoning, and web data. Both models use the Qwen3 tokenizer. The recipe combines SmallInit initialization, AdamW with a warmup-then-stable schedule, and a global batch size of 1024 at sequence length 8192. We track the mean score over eight standard language-model benchmarks using the LM Evaluation Harness (Gao et al., 2024). The complete initialization, optimization schedule, data composition, and annealing breakdown are provided in Appendix D.

4 Post-Training

Since the main focus of this paper is the pre-training recipe, we keep the post-training discussion concise in the main text. Detailed descriptions of the supervised pre-training (SPT) and reinforcement-learning (RL) stages are provided in the Appendix C.

Loopie's post-training recipe consists of two main stages following Stage 2 high-quality annealing. First, we introduce a supervised pre-training stage in which we continue training the model on 2T tokens of instruction-following, reasoning, coding, mathematics, and tool-use data, allowing the base model to acquire broad task-following and problem-solving capabilities while preserving the knowledge and general capabilities learned during pre-training. Second, we apply reinforcement learning to further enhance Loopie's reasoning capabilities, improve its long-horizon problem solving, and align the model so that it produces reliable thinking traces. Together, these stages transform Loopie Base into the final Loopie Thinking model.

4.1 RESULTS

We conduct two complementary comparisons of the final Loopie Thinking models: Loopie-20B-A2B is compared with MoE reasoning models at a similar active-parameter scale, while Loopie-6B-A0.6B is compared with a broader set of compact reasoning models. All evaluations use the EvalScope framework. For AIME 2024 and 2025, we report avg@8; for all other benchmarks, we report pass@1. The AIME result in the teaser figure corresponds to AIME 2024.

Table 1: Comparison of Loopie-20B-A2B with similarly sized MoE models across knowledge, general, code, and math benchmarks. For Loopie-6B-A0.6B results, please refer to the Appendix K.

	Qwen3 30B-A3B Thinking	**Nemotron 3 Nano 30B-A3B**	**Nemotron Cascade 2 30B-A3B**	**GPT-OSS 20B-A2B High**	**Loopie 20B-A2B Thinking**
Pre-training tokens	36T	25T	25T	Unknown	**3.54T**
Knowledge					
MMLU	85.83	80.52	81.22	81.64	81.28
MMLU-Redux	88.25	82.54	83.89	83.40	83.61
General					
ARC-Challenge	96.67	91.96	93.86	92.42	93.52
DROP	87.70	85.92	79.06	62.82	82.08
BBH	86.59	68.76	75.86	84.76	82.28
SciQ	95.40	93.40	92.40	93.80	92.50
IFEval	85.58	77.05	79.21	70.64	84.72
Code					
MBPP	96.50	90.66	80.54	78.99	89.49
MBPP+	98.68	91.53	83.60	97.62	83.07
HumanEval	96.95	86.59	79.88	92.07	89.02
HumanEval+	92.07	87.80	78.66	89.02	87.20
Math					
AIME 24	90.10	85.00	93.33	88.33	92.10
AIME 25	83.75	74.17	91.25	87.50	83.75
AMC	96.73	91.80	93.57	91.05	94.21
OlympiadBench	81.20	76.68	88.64	70.03	80.50

Table 1 demonstrates the pre-training efficiency of Loopie-20B-A2B. Loopie is pre-trained on only 3.54T tokens, whereas Nemotron 3 Nano and Nemotron Cascade 2 are each trained on 25T tokens from the same pre-training data source. Despite using less than one seventh of their pre-training tokens, Loopie matches or outperforms both models on most knowledge and general-capability benchmarks. For example, Loopie achieves 81.28 on MMLU, compared with 80.52 for Nemotron 3 Nano and 81.22 for Nemotron Cascade 2. It scores 93.52 on ARC-Challenge, and reaches 82.28 on BBH and 84.72 on IFEval, substantially outperforming both models on the latter two benchmarks.

Overall, Loopie exceeds at least one of the two Nemotron models on every reported knowledge or general-capability benchmark, and outperforms each of them on five of the seven benchmarks, indicating substantially higher token efficiency under the same pre-training data. Meanwhile, Loopie retains strong post-training reasoning performance, achieving 92.09 on AIME 2024 and 94.21 on AMC, ranking second among the listed models on both benchmarks. On AIME 2025, it scores 83.75, tying Qwen3-30B-A3B Thinking.

5 CONCLUSION

We introduced Loopie, a family of looped MoE language models that makes recurrent depth competitive under a matched pre-training compute budget. By combining layer-loop recurrence with a hardware-aware scaling recipe, Loopie consistently outperforms compute-matched vanilla Transformer baselines across model scales. A large-scale post-training pipeline based on Supervised Pre-training and reinforcement learning further equips Loopie with strong mathematical reasoning and coding abilities. These results suggest that recurrent computation, when jointly optimized with architecture and training efficiency, can serve as a practical scaling axis for large language models.

6 AI USE STATEMENT

We used ChatGPT solely for language polishing of the manuscript, including improvements to grammar, wording, and readability. ChatGPT was not used to develop research ideas or methods, generate code or data, conduct experiments, analyze data, or interpret results. The authors take full responsibility for the final content of this work, including its accuracy, originality, and integrity.

REFERENCES

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin Mohamed Elnouby Ali, Josh Susskind, and Vimal Thilak. Parameters vs FLOPs: Scaling laws for optimal sparsity for mixture-of-experts language models. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2501.12370.

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. In *Advances in Neural Information Processing Systems*, 2023.

Ekin Akyurek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023.

Artur Back de Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.01107.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025. URL https://arxiv.org/abs/2507.10524.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning, 2026. URL https://arxiv.org/abs/2605.19376.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2020.

Akhiad Bercovich, Itay Levy, Izik Golan, et al. Llama-Nemotron: Efficient reasoning models, 2025. URL https://arxiv.org/abs/2505.00949.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14113–14130, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.781. URL https://aclanthology.org/2024.emnlp-main.781/.

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

Aydar Bulatov, Yuri Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, 2022.

Chris Cameron, Wangzheng Wang, Nikita Ivanov, Ashmita Bhattacharyya, Didier Chételat, and Yingxue Zhang. One step forward and K steps back: Better reasoning with denoising recursion models, 2026. URL https://arxiv.org/abs/2604.18839.

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

Ayushi Chadha. When to re-plan: Subgoal persistence in hierarchical latent reasoning, 2026. URL https://arxiv.org/abs/2606.03741.

Bo Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Bypassing the exponential dependency: Looped transformers efficiently learn in-context by multi-step gradient descent. In *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 4447–4455. PMLR, 2025. URL https://proceedings.mlr.press/v258/chen25i.html.

Hsuan-Hao Chen. Thinking deeper, not longer: Depth-recurrent transformers for compositional generalization, 2026.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers, 2026a.

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*, 2026b. URL https://arxiv.org/abs/2606.04438.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceReason-Nemotron: Advancing math and code reasoning through reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026c. URL https://openreview.net/forum?id=EgArbnS0BA.

Zhoujun Cheng, Shibo Hao, Tianyang Liu, Fan Zhou, Yutao Xie, Feng Yao, Yuexin Bian, Yonghao Zhuang, Nilabjo Dey, Yuheng Zha, Yi Gu, Kun Zhou, Yuqi Wang, Yuan Li, Richard Fan, Jianshu She, Chengqian Gao, Abulhair Saparov, Haonan Li, Taylor W. Killian, Mikhail Yurochkin, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Revisiting reinforcement learning for llm reasoning from a cross-domain perspective. *arXiv preprint arXiv:2506.14965*, 2025. doi: 10.48550/arXiv.2506.14965.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC prize 2025: Technical report, 2026. URL https://arxiv.org/abs/2601.10904.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019. URL https://arxiv.org/abs/1911.01547.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A new challenge for frontier ai reasoning systems, 2025. URL https://arxiv.org/abs/2505.11831.

Jishnu Ray Chowdhury and Cornelia Caragea. Investigating recurrent transformers with dynamic halt, 2024. URL https://arxiv.org/abs/2402.00976.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

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.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://openreview.net/forum?id=ZxVrkm7Bjl.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro.

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

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

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569, 2022. URL https://arxiv.org/abs/2112.06905.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Li Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, et al. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, 2023.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory, 2020.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=2edigk8yoU.

Ying Fan, Anej Svete, and Kangwook Lee. Bridging the gap between latent and explicit reasoning with looped transformers, 2026. URL https://arxiv.org/abs/2606.31779.

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. URL https://arxiv.org/abs/2101.03961.

Jacob Fein-Ashley and Paria Rashidinejad. Solve the loop: Attractor models for language and reasoning, 2026. URL https://arxiv.org/abs/2605.12466.

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 *Workshop on Latent and Implicit Thinking, International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2603.08391.

Markus Frey, Behzad Shomali, Joachim Koehler, and Mehdi Ali. A dual-path architecture for scaling compute and capacity in LLMs. *arXiv preprint arXiv:2605.30202*, 2026b. URL https://arxiv.org/abs/2605.30202.

Hengyu Fu, Tianyu Guo, Zixuan Wang, Hanlin Zhu, Jason D. Lee, Jiantao Jiao, Stuart Russell, and Song Mei. Discoloop: Looping discrete embeddings and continuous hidden states for multi-hop reasoning, 2026a. URL https://arxiv.org/abs/2607.00341.

Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. Simply stabilizing the loop via fully looped transformer, 2026b. URL https://arxiv.org/abs/2605.18797.

Tianyu Fu, Yang You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Selective latent iterations to improve reasoning language models, 2025.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024. URL https://github.com/EleutherAI/lm-evaluation-harness.

Yihang Gao, Chuanyang Zheng, Enze Xie, Han Shi, Tianyang Hu, Yu Li, Michael K. Ng, Zhenguo Li, and Zhaoqiang Liu. Algoformer: An efficient transformer framework with algorithmic structures, 2025a. URL https://arxiv.org/abs/2402.13572.

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

Zitian Gao, Haoming Luo, Lynx Chen, Jason Klein Liu, Ran Tao, Joey Zhou, and Bryan Dai. What makes diffusion language models super data learners?, 2025c. URL https://arxiv.org/abs/2510.04071.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15130–15152. PMLR, 2024a. URL https://proceedings.mlr.press/v235/gatmiry24b.html.

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

Renee Ge, Qianli Liao, and Tomaso Poggio. Hierarchical reasoning models: Perspectives and misconceptions, 2025. URL https://arxiv.org/abs/2510.00355.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025. doi: 10.48550/arXiv.2502.05171. URL https://arxiv.org/abs/2502.05171.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, et al. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256. PMLR, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

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

Tianyu Guo, Hanlin Zhu, Ruiqi Zhang, Jiantao Jiao, Song Mei, Michael I. Jordan, and Stuart Russell. How do LLMs perform two-hop reasoning in context?, 2025. URL https://arxiv.org/abs/2502.13913.

Sang-Il Han. Hierarchical vs. flat iteration in shared-weight transformers, 2026. URL https://arxiv.org/abs/2604.14442.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022.

Ailin Huang, Ang Li, Aobo Kong, et al. Step 3.5 Flash: Open frontier-level intelligence with 11B active parameters, 2026a. URL https://arxiv.org/abs/2602.10604.

Benhao Huang, Zhengyang Geng, and J Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=lh95PnOlpM.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, and Behnam Neyshabur. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*, 2022.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3651–3657, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1356.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Hongpeng Jin, Wenqi Wei, Xuyu Wang, Wenbin Zhang, and Yanzhao Wu. Rethinking learning rate tuning in the era of large language models, 2023. URL https://arxiv.org/abs/2309.08859.

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

Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. In *International Conference on Learning Representations*, 2016.

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. URL https://arxiv.org/abs/2001.08361.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, et al. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers, 2026. URL https://arxiv.org/abs/2604.07822.

Mieszko Komisarczyk, Saurabh Mathur, Maurice Kraus, Sriraam Natarajan, and Kristian Kersting. Recursive inference machines for neural reasoning, 2026. URL https://arxiv.org/abs/2603.05234.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 341–353. Curran, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/80083951326cf5b35e5100260d64ed81-Paper-mlsys2023.pdf.

Hsun-Yu Kuo, El Mahdi Chayti, Patrik Reizinger, Wieland Brendel, and Martin Jaggi. Stabilizing extrapolation in looped transformers via learned stochastic stopping, 2026. URL https://arxiv.org/abs/2606.29983.

Dohyeon Kwon and Youngjin Park. Cherry: Compressed hierarchical experts with recurrent representational yield, 2026. URL https://arxiv.org/abs/2606.31796.

Alexander Labovich. Stability and generalization in looped transformers, 2026.

Venkat Akhil Lakkapragada. Cosmicfish-hrm: Adaptive reasoning via hierarchical recurrent mechanisms in compact language models, 2026. URL https://arxiv.org/abs/2605.28919.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

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. URL https://arxiv.org/abs/2605.09165.

Kaixin Li. Taco-verified. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/likaixin/TACO-verified.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023a.

Yingcong Li, M. E. Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In *International Conference on Machine Learning*, 2023b.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? learning program-of-layers in llms, 2026. URL https://arxiv.org/abs/2606.06574.

Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceReason-Nemotron 1.1: Advancing math and code reasoning through SFT and RL synergy. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IaEqjWXd1d.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Wenquan Lu, Yuechuan Yang, Kyle Lee, Yanshu Li, and Enqi Liu. Latent chain-of-thought? decoding the depth-recurrent transformer, 2025.

Kairong Luo, Zhenbo Sun, Haodong Wen, Xinyu Shi, Jiarui Cui, Chenyi Dang, Kaifeng Lyu, and Wenguang Chen. How learning rate decay wastes your best data in curriculum-based llm pretraining. *arXiv preprint arXiv:2511.18903*, 2025. URL https://arxiv.org/abs/2511.18903.

Justus Mattern, Sami Jaghouar, Manveer Basra, Jannik Straube, Matthew Di Ferrante, Felix Gabriel, Jack Min Ong, Vincent Weisser, and Johannes Hagemann. Synthetic-1: Two million collaboratively generated reasoning traces from deepseek-r1. Prime Intellect blog, 2025. URL https://www.primeintellect.ai/blog/synthetic-1-release.

Ronan Killian McGovern. Test-time adaptation of tiny recursive models, 2025. URL https://arxiv.org/abs/2511.02886.

Sean McLeish, Andrew Li, John Kirchenbauer, Deepak S. Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence, 2025.

Andrea Miele, Yiming Qin, Alba Carballo-Castro, Justin Deschenaux, and Pascal Frossard. Fixed-point masked generative modeling, 2026. URL https://arxiv.org/abs/2605.31215.

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, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

MiniMax, Aili Chen, Aonian Li, Baichuan Zhou, et al. The minimax-m2 series: Mini activations unleashing max real-world intelligence, 2026. URL https://arxiv.org/abs/2605.26494.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference, 2025. URL https://arxiv.org/abs/2310.10845.

Ibraheem Muhammad Moosa, Suhas Lohit, Ye Wang, Moitreya Chatterjee, and Wenpeng Yin. Understanding dynamic compute allocation in recurrent transformers, 2026. URL https://arxiv.org/abs/2602.08864.

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, 2026. URL https://arxiv.org/abs/2606.18206.

Toan Q. Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong, November 2019. Association for Computational Linguistics. URL https://aclanthology.org/2019.iwslt-1.17/.

Eshaan Nichani, Alex Damian, and Jason D. Lee. How transformers learn causal structure with gradient descent, 2024.

NVIDIA. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2024. URL https://huggingface.co/datasets/nvidia/Nemotron-CC.

NVIDIA. Nemotron 3 Nano: Open, efficient Mixture-of-Experts hybrid Mamba-Transformer model for agentic reasoning, 2025. URL https://arxiv.org/abs/2512.20848.

NVIDIA. Nemotron 3 Super: Open, efficient Mixture-of-Experts hybrid Mamba-Transformer model for agentic reasoning, 2026a. URL https://arxiv.org/abs/2604.12374.

NVIDIA. Nemotron 3 Ultra: Open, efficient Mixture-of-Experts hybrid Mamba-Transformer model for agentic reasoning, 2026b. URL https://arxiv.org/abs/2606.15007.

Francesco Pappone, Donato Crisostomi, and Emanuele Rodolà. Two-scale latent dynamics for recurrent-depth transformers, 2025. URL https://arxiv.org/abs/2509.23314.

Taekhyun Park, Yongjae Lee, Dohee Kim, and Hyerim Bae. Loopus: Recasting pretrained llms into looped latent refinement models, 2026. URL https://arxiv.org/abs/2605.11011.

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.

Qwen Team. Qwen3 technical report, 2025. URL https://qwenlm.github.io/blog/qwen3/.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Paulius Rauba, Claudio Fanconi, and Mihaela van der Schaar. Tiny autoregressive recursive models, 2026. URL https://arxiv.org/abs/2603.08082. ICLR 2026 Workshop RSI Spotlight.

Zirui Ren and Ziming Liu. Are your reasoning models reasoning or guessing? a mechanistic analysis of hierarchical reasoning models, 2026. URL https://arxiv.org/abs/2601.10679.

Antonio Roye-Azar, Santiago Vargas-Naranjo, Dhruv Ghai, Nithin Balamurugan, and Rayan Amir. Tiny recursive models on ARC-AGI-1: Inductive biases, identity conditioning, and test-time compute, 2025. URL https://arxiv.org/abs/2512.11847.

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://doi.org/10.1609/aaai.v34i05.6399.

Grigory Sapunov. Universal transformers need memory: Depth-state trade-offs in adaptive recursive reasoning, 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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=din0lGfZFd.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026. URL https://arxiv.org/abs/2604.21106.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model, 2026. URL https://arxiv.org/abs/2605.19943.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4596–4604. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/shazeer18a.html.

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. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(24):25219–25227, 2025. doi: 10.1609/aaai.v39i24.34708.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In *Proceedings of the Fourth Workshop on Simple and Efficient Natural Language Processing*, pp. 78–90, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.sustainlp-1.5. URL https://aclanthology.org/2023.sustainlp-1.5/.

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. Spike no more: Stabilizing the pre-training of large language models. In *Proceedings of the Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=52YBEzcI0l. COLM 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

Sijun Tan, Michael Luo, Justin Wong, Colin Cai, Xiaoxiang Shi, William Yuan Tang, Manan Roongta, Tianjun Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Effective rl scaling of reasoning models via iterative context lengthening. OpenReview, 2025. URL https://openreview.net/forum?id=I6GzDCne7U.

Guanghao Tang, Shizhe Jiang, Heng Chang, Nuo Chen, Yifei Li, Hao Fan, Jun Li, Ming Liu, and Bing Qin. Looprpt: Reinforcement pre-training for looped language models, 2026.

Team OLMo. OLMo 3, 2026. URL https://arxiv.org/abs/2512.13961.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452.

Sahar Vahdati, Andrei Aioanei, Haridhra Suresh, and Jens Lehmann. The ARC of progress towards AGI: A living survey of abstraction and reasoning, 2026. URL https://arxiv.org/abs/2603.13372.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, 2023.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15071.

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

Guan Wang, Changling Liu, Chenyu Wang, Cai Zhou, Yuhao Sun, Yifei Wu, Shuai Zhen, Luca Scimeca, and Yasin Abbasi Yadkori. Hrm-text: Efficient pretraining beyond scaling, 2026a. URL https://arxiv.org/abs/2605.20613.

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability, 2026b. URL https://arxiv.org/abs/2606.18524.

Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024. URL https://arxiv.org/abs/2410.05192.

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

Lyumin Wu, Chenyang Zhang, and Yuan Cao. Looped transformers with layer normalization provably learn the power method, 2026. URL https://arxiv.org/abs/2606.00605.

Yichen Wu, Hong Wang, Peilin Zhao, Yefeng Zheng, Ying Wei, and Long-Kai Huang. Mitigating catastrophic forgetting in online continual learning by modeling previous task interrelations via pareto optimization. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=olbTrkWo1D.

Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. Tied transformers: Neural machine translation with shared encoder and decoder. In *AAAI Conference on Artificial Intelligence*, 2019.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms. *arXiv preprint arXiv:2504.14655*, 2025.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022.

Kevin Xu and Issei Sato. To CoT or to loop? a formal comparison between chain-of-thought and looped transformers, 2025a. URL https://arxiv.org/abs/2505.19245.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69613–69646. PMLR, 2025b. URL https://proceedings.mlr.press/v267/xu25x.html.

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.

Kazuki Yano, Shun Kiyono, Sosuke Kobayashi, Sho Takase, and Jun Suzuki. Pre-training llm without learning rate decay enhances supervised fine-tuning. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2603.16127.

Yuekun Yao, Yupei Du, Dawei Zhu, Michael Hahn, and Alexander Koller. Language models can learn implicit multi-hop reasoning, but only if they have lots of training data. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9684–9702, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.490. URL https://aclanthology.org/2025.emnlp-main.490/.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers, 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, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Ruiqi Zhang, Spencer Frei, and Peter L. Bartlett. Trained transformers learn linear models in-context, 2023.

Xiaojing Zhang, Haifeng Wu, Gang He, Jiyang Shen, Bochen Lyu, and Zhanxing Zhu. Modr: Mixture-of-depth-recurrent transformers for test-time reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=9Pba4rcQbE.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Yunao Zheng, Xiaojie Wang, Lei Ren, and Chen Wei. Chaingpt: Dual-reasoning model with recurrent depth and multi-rank state updates. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=kdZbxizwGK.

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. doi: 10.48550/arXiv.2510.25741. URL https://arxiv.org/abs/2510.25741.

Ege Çakar, Ketan Ali Raghu, and Lia Zheng. Laser: Low-rank activation svd for efficient recursion, 2026. URL https://arxiv.org/abs/2604.17224.

A Related Work

A.1 Looped Transformers

We use the term *looped model* to denote an architecture that reuses a learned internal operator within a single forward computation, rather than repeatedly calling an otherwise complete model. This design lineage predates contemporary language models: Neural GPUs repeatedly apply a shared convolutional transition to learn algorithms, Adaptive Computation Time (ACT) learns how many recurrent updates to execute, Universal Transformers (UTs) tie self-attention and transition blocks across depth, and Deep Equilibrium Models (DEQs) avoid explicit unrolling by solving for a fixed point of a weight-tied transformation (Kaiser & Sutskever, 2016; Graves, 2016; Dehghani et al., 2019; Bai et al., 2019; 2020). Work on training implicit models and on generalist neural algorithmic learners further established practical optimization methods and reusable processors for effectively unbounded or task-shared computation. These are direct architectural antecedents of looped Transformers, not merely generic recurrent or test-time-compute methods.

Across Transformer families, layer tying, recurrent state, adaptive depth, and memory reuse also appear in ALBERT, tied Transformers, depth-adaptive Transformers, feedback memory, recurrent-memory Transformers, and block-recurrent Transformers (Lan et al., 2020; Xia et al., 2019; Elbayad et al., 2020; Fan et al., 2020; Bulatov et al., 2022; Hutchins et al., 2022). Modern variants make the shared-depth computation itself the central scaling axis. MoEUT recurrently reuses fine-grained expert groups, Relaxed Recursive Transformers supplement a repeated block with depth-specific low-rank adapters, and implicit state-space language models iterate a shared transition toward a fixed point while preserving substantial training parallelism (Csordás et al., 2024). Thus, rather than relying on ordinary depth scaling, looped Transformers learn a reusable transition $h_{t+1} = F_\theta(h_t, x, e_t)$, where e_t may encode the loop index, a halting state, or an injected copy of the input. Effective inference depth can then vary without introducing an independently parameterized layer at every step.

A.2 Inductive Biases

Looped depth imposes several biases that are weak or absent in a stack of independent Transformer layers. *First, recurrent operator sharing* encourages a reusable update rule rather than a sequence of layer-specific feature maps. This is the defining recursive bias of UTs and tied-depth models, and theoretical analyses connect it to fixed-point iteration, gradient-based in-context learning, normalized gradient descent, and power iteration (Dehghani et al., 2019; Takase & Kiyono, 2023; Yang et al., 2024; Gatmiry et al., 2024a; Chen et al., 2025; Wu et al., 2026). *Second, shared depth decouples effective computation from parameter count*: a compact recurrent core can be unrolled for more steps, favoring iterative refinement over memorizing a separate computation in each layer (Lan et al., 2020; Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025; Schwethelm et al., 2026).

Third, looping induces an iterative-algorithm bias. When a target computation is naturally expressed by repeated application of a small rule set, looped models can emulate programs, optimization procedures, graph algorithms, and structured algorithmic processors, often improving length or depth extrapolation when the loop count adapts to problem size (Giannou et al., 2023; Back de Luca & Fountoulakis, 2024; Gao et al., 2025a; Fan et al., 2025; Xu & Sato, 2025a). *Fourth, weight sharing creates a knowledge re-access and compositionality bias*: after each new bridge entity is inferred, the same retrieval, binding, and update machinery can be applied again. This directly targets failures of implicit multi-hop composition in ordinary Transformers and motivates architectures that carry discrete and continuous states or align latent and explicit reasoning (Wang et al., 2024; Biran et al., 2024; Guo et al., 2025; Yao et al., 2025; Kohli et al., 2026; Fu et al., 2026a; Fan et al., 2026). *Fifth, recurrence provides a latent scratchpad and an adaptive-computation bias*. Additional computation occurs in the hidden state rather than through the generation of extra verbal tokens, while timestep conditioning, token-wise routing, dynamic halting, fixed-point convergence, and shortcut consistency allocate more computation to harder inputs (Saunshi et al., 2025; Geiping et al., 2025; Bae et al., 2025; Fu et al., 2025; Jeddi et al., 2026; Movahedi et al., 2026). These benefits are not automatic: looped models can drift, overthink, or collapse to shallow computation unless their state injection, normalization, supervision, and stopping rules are designed for stable iteration (Chowdhury & Caragea, 2024; Labovich, 2026; Rauba et al., 2026; Moosa et al., 2026).

A.3 Theory, Mechanisms, and Scaling

Formal results characterize both the capabilities and the limits of shared-depth computation. Looped Transformers can execute instruction-level programs, simulate latent chains of thought, and achieve better approximation rates through timestep modulation (Giannou et al., 2023; Saunshi et al., 2025; Xu & Sato, 2025b). More specialized analyses show that a recurrently reused attention layer can learn normalized-gradient updates for in-context logistic regression or implement the power method under layer normalization (Wu et al., 2026). At the same time, compressed recurrent states create a memory-budget separation relative to full sequence-state chain-of-thought methods, and set-valued or graph computations require careful distinctions among fixed points, convergence, and halting. A complementary reinforcement-learning analysis formalizes how additional internal computation changes the class of compute-bounded policies.

A second line of work studies optimization, stability, and scaling with loop count. Fixed-point analyses identify conditions under which recurrent states remain reachable, input-dependent, and trainable; residual-scaling theory argues that correlated tied-block updates require loop-aware scaling; and iso-depth studies quantify the exchange rate among recurrent passes, unique depth, and training compute (Labovich, 2026; Wang et al., 2026b; Schwethelm et al., 2026). Two-scale latent-dynamics analyses derive convergence-sensitive early-exit criteria, while controlled studies of adaptive computation show that learned loop allocation can correlate with token difficulty without necessarily extrapolating to longer inputs (Pappone et al., 2025; Moosa et al., 2026). Memory tokens and ACT initialization can determine whether a UT enters a non-trivial reasoning regime, and hierarchical and flat recurrence can exhibit materially different optimization behavior even under shared-weight controls (Sapunov, 2026; Han, 2026). Compression studies likewise show that preserving per-cell accuracy or local states is insufficient if quantization or pruning damages the recursive trajectory needed for exact solutions.

Mechanistic work examines what is represented across recurrent steps. Probing studies find that recurrent language models often approach loop-specific fixed points or replay feed-forward-like inference stages, while evidence for a literal latent chain-of-thought is mixed and depth-dependent (Blayney et al., 2026; Lu et al., 2025). Shared recurrent modules can nevertheless specialize to perform distinct functional roles through asymmetric state identities, and interaction-locality measurements reveal how repeated local updates accumulate into global puzzle structure. Other probes recover relational preference information from differences between loop states, and analyses of tabular foundation models use the observed layer redundancy to motivate a single repeatedly applied layer. Finally, recurrent-depth models can unlock systematic multi-hop generalization, but autoregressive controls show that the benefit depends on where compute is placed rather than on recursion alone (Kohli et al., 2026; Rauba et al., 2026).

A.4 Architectures, Training Objectives, and Inference

Recent looped language models treat recurrence as a third scaling axis alongside parameter count and token count. Huginn and Ouro pretrain recurrent-depth language models whose shared cores can be unrolled for variable test-time depth, while retrofitted recurrence and LoopUS convert pretrained feed-forward models into latent-refinement systems without training a recurrent model entirely from scratch (Geiping et al., 2025; Zhu et al., 2025; McLeish et al., 2025; Park et al., 2026). HRM-Text and CHERRY explore recurrent compression of deep language models, Hyperloop repeats only a middle block with cross-loop connections, and LoopMoE, sparse looped layers, and universal expert pools combine iterative depth with conditional capacity (Wang et al., 2026a; Kwon & Park, 2026; Zeitoun et al., 2026; Chen et al., 2026b; Lee et al., 2026). MoEUT and Relaxed Recursive Transformers provide earlier expert-routing and low-rank mechanisms for recovering expressivity under parameter sharing (Csordás et al., 2024).

Dynamic-depth architectures decide not only *how* to update a state but also *where* and *for how long* to recur. Mixture-of-Recursions routes tokens to different depths; CoTFormer exposes earlier recurrent representations and learns a compute-budgeted router; AdaPonderLM and Think-at-Hard allocate extra latent iterations selectively; and Chain-of-Layers methods skip or repeat pretrained layers at test time (Bae et al., 2025; Mohtashami et al., 2025; Fu et al., 2025; Li et al., 2026). LoopFormer regularizes trajectories across sampled depths, adaptive-loop models couple halting with external memory, and ChainGPT, MoDr, and depth-recurrent attention mixtures enrich the recurrent state transition through multi-rank updates, branch routing, or mixtures of sequence and depth attention

(Jeddi et al., 2026; Frey et al., 2026a; Zheng et al., 2026; Zhang et al., 2026). SpiralFormer instead changes the resolution schedule across repeated applications, and subgoal-persistence models study when a hierarchical reasoner should re-plan rather than update the same latent plan at every step (Yu et al., 2026; Chadha, 2026).

A parallel line of work targets stable and efficient unrolling. Parallel Loop Transformers share representations across loops, and LT2 replaces quadratic attention with a linear-time looped design (Wu et al., 2025; Deng et al., 2026). MELT keeps cache memory independent of loop depth; LASER compresses recursive activations during training; and Hyperloop and CHERRY reduce the number of distinct parameter blocks (Çakar et al., 2026; Zeitoun et al., 2026; Kwon & Park, 2026). Parcae constrains recurrent dynamics and derives scaling laws, CART anchors each iteration to context, fully looped signal routing stabilizes high loop counts, stochastic stopping improves extrapolation across unseen depths, and stability-aware recurrent training reduces drift under test-time scaling (Prairie et al., 2026; Capps, 2026; Fu et al., 2026b; Kuo et al., 2026). Training-free looping explores whether a pretrained model can be recurrently reused without architectural retraining (Chen et al., 2026a).

Several methods supervise the *trajectory* rather than the final answer alone. LoopRPT applies reinforcement pre-training across latent iterations, RLTT distributes reward over the latent trajectory, denoising recursion trains models to repeatedly correct corrupted targets, and Generative Recursive Reasoning expands recurrence into stochastic multi-trajectory generation (Tang et al., 2026; Cameron et al., 2026; Baek et al., 2026). Probabilistic TRM injects noise at inference and selects among recursive trajectories, whereas LoopFormer uses shortcut consistency to align different computation budgets (Sghaier et al., 2026; Jeddi et al., 2026). Looped diffusion language models, fixed-point masked generative models, equilibrium reasoners, and attractor models connect explicit recurrence to denoising or convergence-based generation (Miele et al., 2026; Fein-Ashley & Rashidinejad, 2026). These approaches make intermediate-state quality, convergence geometry, and trajectory diversity first-class training targets rather than incidental by-products of depth.

A.5 In-Context, Algorithmic, and Compositional Generalization

In-context learning can be interpreted as executing an implicit learning algorithm over demonstrations. Bayesian, gradient-descent, preconditioned-gradient, linear-model, and causal-structure accounts motivate studying whether each recurrent step implements one additional internal update (Xie et al., 2022; Akyurek et al., 2023; von Oswald et al., 2023; Ahn et al., 2023; Li et al., 2023b; Zhang et al., 2023; Nichani et al., 2024). Looped Transformers make this correspondence explicit: they learn data-fitting algorithms with substantially fewer unique parameters, implement multi-step gradient procedures, and can use distinct preprocessing, looping, and postprocessing stages (Yang et al., 2024; Gatmiry et al., 2024a; Chen et al., 2025; Gao et al., 2025a). Program-simulation results and normalized-gradient analyses further connect shared depth to reusable algorithmic primitives (Giannou et al., 2023).

The same prior is relevant to length and compositional extrapolation. Neural GPUs and generalist recurrent processors learn repeated algorithmic updates, while looped Transformers simulate graph algorithms and obtain strong length generalization when the number of recurrent steps grows with the instance (Kaiser & Sutskever, 2016; Back de Luca & Fountoulakis, 2024; Fan et al., 2025). Timestep modulation, multi-resolution recursion, and depth-recurrent compositional models address the failure of a single stationary update to remain useful far beyond the trained depth (Xu & Sato, 2025b; Yu et al., 2026; Chen, 2026). DiscoLoop carries both discrete embeddings and continuous hidden states across hops, and LOTUS aligns recurrent latent blocks with explicit chain-of-thought computation (Fu et al., 2026a; Fan et al., 2026). These results complement evidence that ordinary Transformers often memorize local facts or short computations yet fail to compose them systematically out of distribution (Biran et al., 2024; Yao et al., 2025; Dziri et al., 2023).

A.6 Abstract Reasoning

ARC-style tasks are a natural stress test because they require inducing a new transformation from a few demonstrations rather than recalling a fixed skill (Chollet, 2019; Chollet et al., 2025). HRM obtains substantial effective depth through coupled high- and low-level recurrent modules, TRM reduces the design to a tiny repeatedly applied network, and the URM study attributes much of the gain to Universal-Transformer-style recurrence and nonlinear depth computation (Wang et al.,

2025; Jolicoeur-Martineau, 2025; Gao et al., 2025b). Controlled analyses caution that hierarchy, augmentation, identity conditioning, majority voting, and competition-time adaptation can materially affect reported scores (Ge et al., 2025; Ren & Liu, 2026; Roye-Azar et al., 2025; McGovern, 2025).

Follow-up designs explore complementary ways to improve recursive abstract reasoning. CosmicFish-HRM adapts hierarchical recurrence to compact language models; Recursive Inference Machines generalize generator–solver recursion; and Fixed-Point Reasoners halt upon convergence rather than at a preset depth (Lakkapragada, 2026; Komisarczyk et al., 2026; Movahedi et al., 2026). Generative, probabilistic, and denoising recursive models introduce trajectory diversity or iterative corruption-and-repair curricula (Baek et al., 2026; Sghaier et al., 2026; Cameron et al., 2026). Equilibrium and attractor reasoners learn convergent solution dynamics, whereas Tiny Autoregressive Recursive Models test whether comparable compute is better spent on recurrence or ordinary autoregressive depth (Fein-Ashley & Rashidinejad, 2026; Rauba et al., 2026). Memory-augmented UTs, LoopViT, and interaction-locality analyses expose complementary depth–state, visual-recurrence, and mechanism-level views (Sapunov, 2026; Shu et al., 2026). Overall, this literature supports recurrence as an architectural prior for iterative abstraction, while showing that successful test-time scaling depends on stable state transitions and informative intermediate supervision rather than loop count alone (Chollet et al., 2026; Vahdati et al., 2026).

B Future Work

Due to computational constraints, our post-training experiments focus primarily on mathematical and code reasoning tasks. We do not extensively explore other important capabilities studied in prior work, such as scientific question answering, instruction following, alignment with human conversational preferences, or agentic task-solving. A natural direction for future work is to continue to post-train Loopie on a broader set of tasks and preference signals, which may further improve its practical utility and general-purpose capabilities. Also, due to limited computational resources, we were unable to conduct a sufficiently comprehensive ablation study of supervised pre-training, which we leave for future work.

In addition, our study focuses primarily on matching compute budgets during pre-training, and we have not yet conducted systematic studies of inference-time computation. Matching and optimizing inference-time compute remain important directions for future work; prior work such as the Parallel Loop Transformer (Wu et al., 2025) provides a promising example of this approach.

Finally, our study intentionally focuses on applying looped Transformers to a relatively clean and well-controlled base architecture, Qwen3-30B-A3B, in order to avoid confounding factors introduced by architectural variations. As a result, we do not investigate several recent architectural advances that may be complementary to our method. Exploring how Loopie interacts with these newer designs remains an important direction for future work.

C Post-Training Details

C.1 Supervised Pre-Training

We introduce *supervised pre-training* (SPT), a training regime that combines the supervision pattern of supervised fine-tuning (SFT) with the optimization scale of language-model pre-training (PT). SPT uses exactly the same training data and token-level objective as SFT: prompt and context tokens are excluded from the loss, and only supervised target tokens contribute to optimization. Unlike conventional SFT, however, SPT uses global batch sizes, sequence lengths, and token budgets typical of language-model pre-training.

Table 2: Comparison of supervised pre-training (SPT), conventional supervised fine-tuning (SFT), and pre-training (PT). The SPT column reports the representative configuration used for LOOPIE. $\uparrow$, $\downarrow$, and $\approx$ indicate improvement, degradation, and no material change, respectively, relative to the corresponding starting checkpoint. Stability entries summarize our observations over the evaluated training horizon rather than universal guarantees.

	SPT	**SFT**	**PT**
Loss function	Cross-entropy	Cross-entropy	Cross-entropy
Loss-bearing positions	Supervised target tokens only	Supervised target tokens only	All non-padding tokens
Pre-training metrics	$\uparrow$	$\downarrow$	$\uparrow$
Reasoning metrics	$\uparrow$	$\uparrow$	$\approx$
Overfitting after multiple epochs	Not observed	Observed	Not observed
Global batch size	$\geq$ 1,024	32–128	$\geq$ 1,024
Sequence length	$\geq$ 128 K	8–32 K	4–8 K
Nominal token positions per batch	$\geq$ 128 M	128 K–1 M	8–32 M
Total training token budget	$\geq$ 2 T	10–100 B	$\geq$ 4 T

SPT separates two design choices that are commonly coupled: (i) which tokens contribute to the training loss, and (ii) the scale at which optimization is performed. SFT and SPT share the former choice, whereas PT and SPT share the latter. Consequently, SPT does not interpolate between an SFT loss and a PT loss. Instead, it applies an SFT-style supervised objective in a PT-scale optimization regime.

Training objective. Consider a supervised example $z_i = [c_i; y_i]$, where c_i denotes the input context and y_i denotes the supervised target response. Let $w_{i,t} \in \{0, 1\}$ be a binary loss mask for the token at position t. All three training regimes use token-level cross-entropy:

$$\mathcal{L}_{\mathrm{CE}}(\theta; w) = -\frac{\sum_{i=1}^{B}\sum_{t=1}^{T_i} w_{i,t} \log p_\theta\left(z_{i,t} \mid z_{i,<t}\right)}{\sum_{i=1}^{B}\sum_{t=1}^{T_i} w_{i,t}}.$$

For both SPT and SFT, $w_{i,t} = 1$ only when $z_{i,t}$ belongs to the supervised target y_i; prompt, context, and padding positions are masked out. In conventional PT, by contrast, $w_{i,t} = 1$ for every non-padding token. Thus, all three regimes employ the same cross-entropy loss, but differ in the positions to which the loss is applied and in the scale of optimization.

Table 2 summarizes these distinctions. For representative SFT settings, including global batch sizes, training token counts, and sequence lengths, we primarily follow the configurations reported for OLMo 3, the Nemotron 3 series, and Step-3.5-Flash (Team OLMo, 2026; NVIDIA, 2025; Bercovich et al., 2025; NVIDIA, 2026a;b; Huang et al., 2026a). We use *pre-training metrics* to refer to metrics that evaluate general base-model capabilities and *reasoning metrics* to refer to metrics that primarily evaluate performance on challenging reasoning tasks, such as competition-level

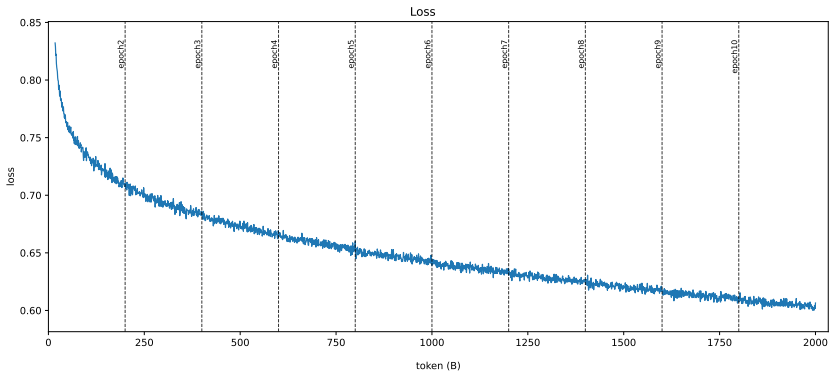


Figure 6: Training loss for Loopie-6B-A0.6B during supervised pre-training. Because the evaluation and training loss curves nearly overlap, only the training loss curve is shown. Over 10 epochs of supervised pre-training on a total of 2 trillion tokens, the loss decreases smoothly, with no loss cliff observed at epoch boundaries.

mathematics and coding problems. In our experiments, SPT improves performance on both types of metrics simultaneously. In contrast, the conventional SFT baseline only improves performance on reasoning metrics but degrades performance on pre-training metrics, whereas PT improves pre-training metrics performance while leaving performance on reasoning metrics approximately unchanged.

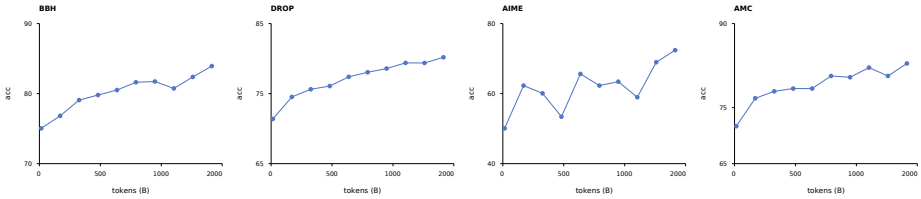


Figure 7: Trends in reasoning metrics during supervised pre-training. Throughout training on 2 trillion tokens, performance on the reasoning metrics continues to improve, with no sign of slowing.

Sustained downstream gains over repeated data exposure. The central evidence for the effectiveness of multi-epoch SPT is its downstream learning trajectory, rather than the shape of the training-loss curve. Across approximately ten epochs and 2T training tokens, the reasoning benchmarks exhibit an overall upward trend, with further gains in the later epochs despite checkpoint-level fluctuations (Figure 7). The evaluated general-capability metrics show a compatible pattern: MMLU continues to improve, while ARC-Challenge retains its early gains without a sustained late-stage decline (Figure 8). These results show that repeated exposure to the supervised data continues to yield useful downstream improvements. Within the evaluated training horizon, we observe no sustained downstream degradation indicative of overfitting. The smoothly decreasing loss in Figure 6 provides complementary evidence of stable optimization.

Pre-training-scale optimization over supervised data. SPT processes 128 million tokens per global batch, approximately 1,000 times the number processed per batch in conventional SFT. This makes overfitting much less likely in SPT than in standard SFT. This scale gives SPT distinct training dynamics, consistent with observations that optimization hyperparameters and scaling behavior can change in large-language-model training regimes (Jin et al., 2023). Under SPT, the model can be trained smoothly for substantially more epochs without showing signs of overfitting, such as abrupt drops in loss at epoch boundaries. At the same time, performance on downstream metrics continues to improve steadily. In addition, SPT mitigates catastrophic forgetting (Wu et al., 2024), which is a major drawback of conventional SFT and typically causes substantial degradation of the general

knowledge acquired during pre-training. Notably, SPT not only avoids this degradation but also further improves general-knowledge performance.

As shown in Figure 8, SPT consistently improves performance on downstream pre-training metrics, including ARC-Challenge and MMLU, throughout approximately 10 epochs of training on 2T tokens. This finding challenges the conventional view that SFT leads to catastrophic forgetting.

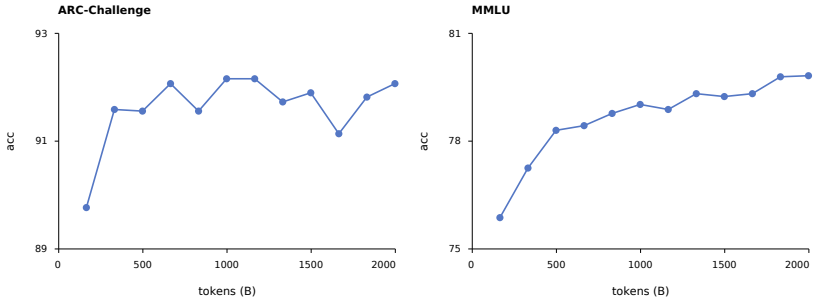


Figure 8: Trends in pre-training metrics during supervised pre-training. ARC-Challenge represents general reasoning performance, while MMLU represents general knowledge performance. Throughout training on 2 trillion tokens, performance on the pre-training metrics does not degrade; instead, it continues to improve.

Simply running a conventional SFT configuration for more epochs does not reproduce the optimization regime of SPT. Conventional-scale SFT often begins to overfit between the second and fourth epochs. In small-batch SFT, each epoch involves many parameter updates computed from small batches, and repeated exposure to the same examples can lead to rapid specialization and memorization. In contrast, every SPT update aggregates supervision from more than one thousand sequences and over one hundred million nominal token positions. We hypothesize that this broad gradient aggregation, together with long contexts and a large token budget, mitigates the overly narrow specialization commonly observed during repeated SFT.

C.2 Reinforcement Learning

After SPT, we apply a reinforcement-learning stage to obtain **Loopie Thinking**. Following prior work, we do not mix data from multiple domains during training because domain-specific length biases may interfere with one another (Chen et al., 2026c; Liu et al., 2026). We first conduct reinforcement learning on mathematical tasks; once performance saturates, we continue with reinforcement learning on coding tasks.

Algorithm. Our optimizer builds on Group Sequence Policy Optimization (GSPO) (Zheng et al., 2025), with the asymmetric clipping and dynamic sampling techniques introduced by DAPO (Yu et al., 2025). For each prompt q, we sample a group of G completions $\{o_i\}_{i=1}^{G}$ from the old policy $\pi_{\theta_{\text{old}}}$ and score each completion using a verifier, yielding outcome rewards $\{R_i\}_{i=1}^{G}$. We estimate sequence-level advantages by normalizing rewards within the sampled group:

$$\hat{A}_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^{G})}{\text{std}(\{R_j\}_{j=1}^{G}) + \epsilon},$$

where ϵ is a small numerical constant.

Unlike token-level policy optimization, GSPO defines a length-normalized sequence-level importance ratio:

$$s_i(\theta) = \left(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)} \right)^{\frac{1}{|o_i|}} = \exp\left(\frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \log \frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})} \right).$$

That is, $s_i(\theta)$ is the geometric mean of the token-level importance ratios over the response. The length normalization reduces the variance of the sequence-level ratio and keeps responses of different lengths within a comparable numerical range.

The policy is optimized using a sequence-level clipped objective:

$$\mathcal{J}_{\mathrm{RL}}^{\mathrm{GSPO}}(\theta) = \mathbb{E}_{\substack{q\sim\mathcal{D}, \\ \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}(\cdot|q)}} \left[\frac{1}{G}\sum_{i=1}^{G} \min\left(s_i(\theta)\hat{A}_i,\ \mathrm{clip}\left(s_i(\theta), 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}}\right)\hat{A}_i \right) \right].$$

In contrast to token-level clipping, GSPO applies the clipping operation to the entire response. Consequently, all tokens in the same response share the same sequence-level importance weight, aligning the unit of off-policy correction and optimization with the unit of reward.

Following the Clip-Higher strategy, we use $\epsilon_{\mathrm{high}} > \epsilon_{\mathrm{low}}$. Under GSPO, this asymmetric clipping is applied to the sequence-level importance ratio rather than individual token-level ratios. The larger upper clipping range allows positively advantaged exploratory responses to receive stronger probability-increasing updates, while the more conservative lower clipping range limits excessive probability decreases for negatively advantaged responses.

We further apply prompt-level dynamic filtering. A prompt is retained only when its sampled group contains both successful and unsuccessful completions:

$$0 < \sum_{i=1}^{G} \mathbf{1}\{R_i = R_{\mathrm{pass}}\} < G.$$

Groups that are entirely correct or entirely incorrect provide no useful relative preference signal for GRPO. We therefore oversample candidate prompts and filter out such groups until the effective batch is full. This keeps the optimization focused on prompts that yield non-degenerate policy gradients.

Training data. We use the mathematics and code subsets of the Guru-RL corpus (Cheng et al., 2025). The mathematics pool is drawn from the OR1, DAPO, and DeepScaleR sources (He et al., 2025; Yu et al., 2025; Tan et al., 2025). The code pool combines LeetCodeDataset, TACO-Verified, PrimeIntellect/SYNTHETIC-1, and historical LiveCodeBench training problems (Xia et al., 2025; Li et al., 2023a; Li, 2025; Mattern et al., 2025; Jain et al., 2024). We apply an additional curation pass that removes malformed prompts, unverifiable answers, flaky tests, duplicate examples, and examples that substantially overlap with held-out evaluation sets. Rewards for mathematical problems are computed using rule-based answer equivalence, whereas rewards for coding problems are computed using sandboxed unit-test execution.

Training schedule. We train in two context-length stages. The first stage uses a maximum response length of 32K tokens. This stage is computationally efficient because most rollouts terminate naturally within this limit. When the rollout truncation rate exceeds 10%, we switch to a 64K-token stage, giving the policy additional room for longer derivations and code-reasoning trajectories. We continue RL until the aggregate validation score stops improving and begins to decline; the checkpoint immediately before sustained degradation is selected as **Loopie Thinking**. Throughout training, we monitor validation accuracy, mean response length, generation entropy, and truncation rate to detect late-stage over-optimization.

Table 1 demonstrates the pre-training efficiency of Loopie-20B-A2B. Loopie is pre-trained on only 3.54T tokens, whereas Nemotron 3 Nano and Nemotron Cascade 2 are each trained on 25T tokens from the same pre-training data source. Despite using less than one seventh of their pre-training tokens, Loopie matches or outperforms both models on most knowledge and general-capability benchmarks. For example, Loopie achieves 81.28 on MMLU, compared with 80.52 for Nemotron 3 Nano and 81.22 for Nemotron Cascade 2. It scores 93.52 on ARC-Challenge, and reaches 82.28 on BBH and 84.72 on IFEval, substantially outperforming both models on the latter two benchmarks.

Overall, Loopie exceeds at least one of the two Nemotron models on every reported knowledge or general-capability benchmark, and outperforms each of them on five of the seven benchmarks, indicating substantially higher token efficiency under the same pre-training data. Meanwhile, Loopie retains strong post-training reasoning performance, achieving 92.09 on AIME 2024 and 94.21 on AMC, ranking second among the listed models on both benchmarks. On AIME 2025, it scores 83.75, tying Qwen3-30B-A3B Thinking.

D Pre-Training Details

D.1 Evaluation

We evaluate the pre-trained checkpoints using the LM Evaluation Harness framework (Gao et al., 2024). For all experiments in Section 2, we report the mean score across the following eight benchmarks: ARC-Challenge (Clark et al., 2018), ARC-Easy (Clark et al., 2018), BoolQ (Clark et al., 2019), CommonsenseQA (Talmor et al., 2019), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021), OpenBookQA (Mihaylov et al., 2018), and WinoGrande (Sakaguchi et al., 2020).

D.2 Initialization

Let d denote the model width. We initialize the token embedding matrix E and the language-modeling head W_{lm}—or the shared embedding/unembedding matrix when weight tying is used—as $E_{ij}, (W_{\mathrm{lm}})_{ij} \sim \mathcal{N}(0, d^{-1})$. On the input side, the looked-up embeddings are multiplied by $\sqrt{d}$ before entering the residual stream, so that the embedding shortcut has $O(1)$ activation scale, while the same d^{-1} variance ensures that the initial logits $h^\top w$ have $O(1)$ variance for normalized hidden states (Shazeer & Stern, 2018; Takase et al., 2025). For hidden-to-hidden parameters inside Transformer blocks, including attention projections and feed-forward layers, we use SmallInit:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{5d}\right), \qquad \mathrm{std}(W) = \frac{1}{\sqrt{2.5d}}.$$

This scale corresponds to Xavier fan-in/fan-out initialization for a Transformer FFN with an expansion ratio of 4, since $d_{\mathrm{in}} + d_{\mathrm{out}} = d + 4d = 5d$. We also apply this scale to attention projections to reduce the scale of residual-branch sub-layers (Glorot & Bengio, 2010; Nguyen & Salazar, 2019). Thus, we use $d^{-1/2}$ for embeddings/unembeddings, which determine the shortcut and logit scales, but $(2.5d)^{-1/2}$ for block sub-layers, whose Jacobians should remain small for stable pre-training (Takase et al., 2025).

D.3 Learning Schedule

We optimize all models with AdamW (Kingma & Ba, 2015; Loshchilov & Hutter, 2019). We set the peak learning rate to 5×10^{-4} for Loopie-6B-A0.6B and 3×10^{-4} for Loopie-20B-A2B. We set the AdamW hyperparameters to $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-15}$, use a weight decay of 0.1, and apply global gradient-norm clipping at 1.0. We use a warmup-stable-only learning-rate schedule. After a specified number of warmup steps—6,000 steps for Loopie-20B-A2B and 2,000 steps for Loopie-6B-A0.6B—the learning rate reaches its peak value and then remains constant without decay. This decay-free stable phase keeps the effective update scale high after warmup, avoiding the diminished influence of late-stage high-quality data that can occur under annealed schedules (Wen et al., 2024; Luo et al., 2025; Yano et al., 2026). We also use a constant learning rate during the high-quality annealing phase in Stage 2 to maximize learning from the highest-quality annealing data (Luo et al., 2025). The global batch size is 1024, and the sequence length is 8192.

D.4 Stage 1: Multi-Epoch High-Quality Pre-Training

In Stage 1, we train both Loopie-20B-A2B and Loopie-6B-A0.6B on Nemotron-CC-v2-HQ (NVIDIA, 2024), a high-quality subset of Nemotron-CC. The corpus contains approximately 570B unique tokens, and we train for four epochs, totaling approximately 2.28T training tokens. This design is motivated by repeated training on high-quality data can be more effective than single-pass training on a larger corpus of lower average quality (Gao et al., 2025c).

D.5 Stage 2: High-Quality Annealing

In Stage 2, we construct a high-quality annealing mixture from Nemotron pre-training datasets. The resulting pool contains approximately 1.26T tokens. The mixture combines high-quality SFT-style data, synthetic reasoning data, code data, synthetic web data, and math data. In total, the pool contains approximately 1263B tokens: 351B from Nemotron-pre-training-SFT-v1 (27.8%), 277B

from Nemotron-pre-training-Specialized-v1 (21.9%), 262B from a 60% sample of Nemotron-pre-training-Code-v2 (20.7%), 197B from a 16% sample of Nemotron-CC-v2-HQ-Synthetic (15.6%), 126B from Nemotron-CC-Math-v1 with quality scores ≥ 4 (10.0%), 25B from Nemotron-CC-v2.1-HQ (2.0%), and 25B from Nemotron-CC-v2.1-HQ-Synthetic (2.0%). Figure 9 visualizes these relative proportions.

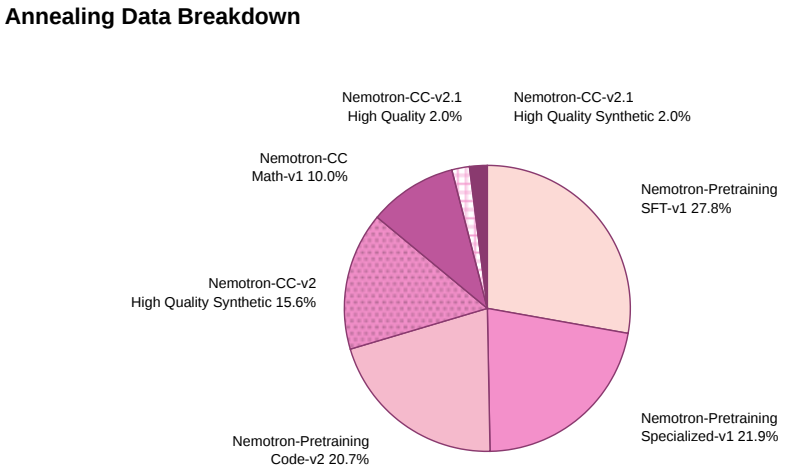


Figure 9: Composition of the Stage-2 high-quality annealing data pool: multiple data sources make up the 1.26T-token annealing recipe.

Nemotron-pre-training-SFT-v1. We include approximately 351B tokens from Nemotron-pre-training-SFT-v1, a diverse SFT-style dataset of synthetic and curated examples spanning STEM, academic, code, mathematics, and reasoning domains, with multilingual coverage. Its STEM component is expanded from high-quality math and science seeds through iterative generation with Qwen3 and DeepSeek models, producing harder and more varied questions with solutions. The dataset also contains academic question-answer pairs synthesized from undergraduate- and graduate-level texts, as well as MMLU-style general QA and fundamental reasoning data.

Nemotron-pre-training-Specialized-v1. We include approximately 277B tokens from Nemotron-pre-training-Specialized-v1, which comprises synthetic data for STEM reasoning, scientific coding, and cross-domain coding, as well as synthetic Wikipedia data and synthetic mathematics textbook data. The STEM reasoning component includes reasoning question-answer demonstrations generated from advanced scientific seed documents, while the scientific coding and cross-domain coding subsets introduce graduate- or research-level programming tasks with structured solutions. This data is intended to strengthen scientific reasoning, mathematical abstraction, and code generation during annealing.

Nemotron-pre-training-Code-v2. We randomly sample 60% of Nemotron-pre-training-Code-v2, yielding approximately 262B tokens. This dataset combines recent GitHub source code, synthetic code-grounded question-answer data, student-teacher dialogues, code-review dialogues, and LLM-rewritten or transpiled source code. The rewriting and transpilation components are designed to improve downstream code generation by increasing stylistic diversity and exposing the model to semantically equivalent code variants.

High-quality synthetic web data. We randomly sample 16% of Nemotron-CC-v2-High-Quality-Synthetic, yielding approximately 197B tokens. This dataset is derived from Nemotron-CC-v2 and contains English web-crawl documents augmented through synthetic rephrasing with Qwen3-30B-A3B. We further include 25B tokens from Nemotron-CC-v2.1-High-Quality-Synthetic, which extends

the synthetic high-quality web corpus with newer Common Crawl snapshots and additional rephrased medium- to high-quality documents.

High-quality web data. We include approximately 25B tokens from Nemotron-CC-v2.1-High-Quality. This subset incorporates recent Common Crawl snapshots and high-quality data translated into English from multiple languages. Additional LLM-based filtering removes uninformative translated documents. The inclusion of this data preserves exposure to natural web text during the annealing phase.

Mathematics data. Finally, we include approximately 126B tokens from Nemotron-CC-Math-v1, using only documents with quality scores of 4 and above. Nemotron-CC-Math-v1 is a high-quality math pre-training corpus built from Common Crawl using a pipeline designed to preserve equations and code, convert mathematical notation to standardized LaTeX, and remove noise. This component is included to strengthen mathematical reasoning and symbolic problem-solving capabilities.

E ARCHITECTURE DETAILS

Table 3: Architecture Details of Loopie-20B-A2B and Loopie-6B-A0.6B

	Loopie-20B-A2B	Loopie-6B-A0.6B
Layer-loop times	2	2
QK layer norm	Enabled	Enabled
Num layers	27	18
Hidden size	2304	1536
MoE hidden size	832	576
Attention type	GQA	GQA
Attention heads	72	48
Attention groups	36	24
Position embedding type	RoPE	RoPE
Rotary base	10000	10000
RMSNorm eps	1e-6	1e-6
MoE router topk	8	8
Num experts	128	128
MLP type	SwiGLU	SwiGLU
Tokenizer	Qwen3	Qwen3
Vocabulary size	151936	151936

Table 4: Candidate configurations generated from the Qwen3-like 30B-A3B reference. $\widehat{C}$ is the normalized leading-order Transformer block work proxy, and $\widehat{M}_{\text{act}}$ is the normalized activation memory proxy at a fixed per-device microbatch size. Final compute matching is based on measured optimizer-step wall-clock time.

Configuration	D	L	R	$\widehat{C}$	$\widehat{M}_{\text{act}}$
Qwen3 30B-A3B	2048	48	1	1.00×	1.00×
Seed Loopie	2048	24	2	1.00×	0.50×
Loopie candidate 1	2176	25	2	1.18×	0.55×
Loopie candidate 2	2304	27	2	1.42×	0.63×
Loopie candidate 3	2432	28	2	1.65×	0.69×

F LOOPIE RECIPE DERIVATION

This appendix provides the concrete architecture derivation and candidate-selection procedure for the Loopie Recipe summarized in Section 2.3.

We use a Qwen3-like 30B-A3B MoE Transformer (Qwen Team, 2025) as the non-recurrent reference, with

$$D_0 = 2048, \qquad L_0 = 48, \qquad R_0 = 1.$$

The first step is to construct a recurrent seed model by halving the number of stored layers and setting $R = 2$:

$$D = 2048, \qquad L = 24, \qquad R = 2.$$

The leading-order pre-training compute (Hoffmann et al., 2022) scales as

$$C(D, L, R) \propto LD^2R.$$

At a fixed width, this transformation preserves the number of Transformer block executions:

$$LR = 24 \cdot 2 = 48 = L_0R_0,$$

and therefore preserves the leading-order Transformer block compute proxy:

$$LRD^2 = 24 \cdot 2 \cdot 2048^2 = 48 \cdot 2048^2.$$

At the same per-device microbatch size, the dominant activation memory term is reduced by half:

$$\frac{DL}{D_0L_0} = \frac{2048 \cdot 24}{2048 \cdot 48} = 0.5.$$

The recurrent seed model therefore performs approximately the same leading-order Transformer block work as the reference while requiring substantially less memory for stored activations (Shoeybi et al., 2019; Rajbhandari et al., 2020).

The second step is to spend the resulting microbatch efficiency on additional model capacity. To satisfy architectural and hardware alignment constraints, we restrict the candidate hidden sizes to multiples of 128. We sweep feasible (D, L) configurations around the recurrent seed model and retain candidates that support a microbatch size twice that of the reference.

Table 4 lists the candidate configurations. We report the normalized leading-order compute proxy:

$$\widehat{C} = \frac{LRD^2}{48 \cdot 2048^2},$$

and the activation memory proxy at the reference microbatch size,

$$\widehat{M}_{\text{act}} = \frac{DL}{48 \cdot 2048}.$$

These quantities are used to describe the candidates, not to predict their final optimizer-step times.

We conduct large-scale benchmarking of each candidate configuration in Megatron-LM, following the matching protocol described above. For the reference model and each candidate model, we jointly

search over tensor parallelism, expert parallelism, and microbatch size. We then select the candidate whose measured optimizer-step time is closest to that of the non-recurrent reference model:

$$D_1 = 2304, \qquad L_1 = 27, \qquad R_1 = 2,$$

which defines Loopie-20B-A2B.

At the reference per-device microbatch size, the dominant activation memory proxy of Loopie-20B-A2B relative to the reference is

$$\frac{D_1 L_1}{D_0 L_0} = \frac{2304 \cdot 27}{2048 \cdot 48} \approx 0.633.$$

This expression is a scaling proxy rather than an exact peak-memory equation. Candidate feasibility is therefore determined using the measured full memory footprint, which also includes parameters, optimizer states, temporary buffers, and communication workspaces. The measured memory profile of Loopie-20B-A2B permits the per-device microbatch size to be doubled:

$$b_1 = 2b_0, \qquad g_1 = \frac{g_0}{2}.$$

This leaves the number of tokens per optimizer step unchanged.

G PRE-TRAINING DETAILS

Table 5: Pre-training Stage-1 Details of Loopie-20B-A2B and Loopie-6B-A0.6B

	Loopie-20B-A2B	Loopie-6B-A0.6B
Initialization std	$1/\sqrt{2.5 * 2304}$	$1/\sqrt{2.5 * 1536}$
LM head Initialization std	$1/\sqrt{2304}$	$1/\sqrt{1536}$
Embedding Initialization std	$1/\sqrt{2304}$	$1/\sqrt{1536}$
MoE Auxiliary loss coeffient	0.01	0.01
Learning rate	3×10^{-4}	5×10^{-4}
Learning rate schedule	Warmup-then-stable	Warmup-then-stable
Warmup steps	6000	2000
Global batch size	1024	1024
Sequence length	8192	8192
Optimizer	AdamW	AdamW
β_1	0.9	0.9
β_2	0.95	0.95
Weight decay	0.1	0.1
Clip gradient	1.0	1.0
Adam epsilon	1×10^{-15}	1×10^{-15}

Table 6: Pre-training Stage-2 Details of Loopie-20B-A2B and Loopie-6B-A0.6B

	Loopie-20B-A2B	Loopie-6B-A0.6B
Learning rate	3×10^{-4}	5×10^{-4}
Learning rate schedule	Constant	Constant
MoE Auxiliary loss coeffient	0.01	0.01
Global batch size	1024	1024
Sequence length	8192	8192
Optimizer	AdamW	AdamW
β_1	0.9	0.9
β_2	0.95	0.95
Weight decay	0.1	0.1
Clip gradient	1.0	1.0
Adam epsilon	1×10^{-15}	1×10^{-15}

H SCALING LADDER DETAILS

Table 7: Architecture details for the Loopie scaling ladder. Each Loopie model is paired with a non-recurrent vanilla MoE baseline under matched pre-training wall time. Here D denotes hidden dimension, D_{MoE} denotes each expert's hidden size, L denotes the number of stored Transformer/MoE layers, Heads denotes the number of attention heads, Head Dim denotes the dimension of each attention head, N denotes the number of recurrent layer-loop steps, and Width/Depth denotes $D/(LN)$.

Rung	Model	Total Params	Active Params	L	D	D_{MoE}	Heads	Head Dim	N	Width/ Depth	Tokens
1	Vanilla	1.34B	0.15B	27	640	192	20	32	1	23.70	150B
	Loopie	1.08B	0.11B	15	704	256	22	32	2	23.47	150B
2	Vanilla	2.37B	0.25B	30	768	256	24	32	1	25.60	250B
	Loopie	1.81B	0.18B	17	832	320	26	32	2	24.47	250B
3	Vanilla	4.76B	0.51B	36	1280	320	32	40	1	35.56	500B
	Loopie	3.78B	0.41B	19	1280	384	32	40	2	33.68	500B
4	Vanilla	9.14B	1.00B	46	1280	384	40	32	1	27.83	500B
	Loopie	6.36B	0.68B	25	1408	448	44	32	2	28.16	500B

I SUPERVISED PRE-TRAINING DETAILS

Table 8: Training hyperparameters for Loopie-20B-A2B and Loopie-6B-A0.6B in Supervised Pre-training.

	Loopie-20B-A2B	Loopie-6B-A0.6B
Global batch size	1024	1024
Sequence length	131072	131072
Learning rate	1×10^{-5}	1×10^{-5}
Learning rate schedule	Warmup-then-stable	Warmup-then-stable
Warmup steps	500	500
MoE Auxiliary loss coeffient	0.01	0.01
Optimizer	AdamW	AdamW
β_1	0.9	0.9
β_2	0.95	0.95
Weight decay	0.1	0.1
Clip gradient	1.0	1.0
Adam epsilon	1×10^{-15}	1×10^{-15}

J MOTIVATION

Despite their conceptual appeal, looped Transformers have historically been studied in settings that do not fully reflect the constraints of modern large-scale language model pre-training. In particular, much of the prior work has focused on dense, weight-shared, or recurrent Transformer variants and has evaluated recurrence primarily as a mechanism for improving parameter efficiency (Zhu et al., 2025; Geiping et al., 2025). This leaves two important issues underexplored.

First, modern frontier language models increasingly rely on Mixture-of-Experts architectures, as exemplified by the recent Qwen3, Kimi K2.5, GLM-5, DeepSeek-V4, and MiniMax-M2 model families (Qwen Team, 2025; Kimi Team et al., 2026; GLM-5 Team et al., 2026; DeepSeek-AI, 2026; MiniMax et al., 2026). MoE models expand total model capacity while keeping the number of active parameters per token relatively small, making them substantially more favorable than dense models under both training and inference compute constraints (Shazeer et al., 2017; Fedus et al., 2022; Du et al., 2022; Abnar et al., 2025). A practical looped architecture should therefore be compatible with MoE scaling rather than be restricted to dense backbones, because such a restriction would limit its ability to scale to larger model sizes.

Second, pre-training compute is often the dominant constraint in large-model training. It determines how large a model can be, how many tokens it can see, and how many experimental variants can be trained (Kaplan et al., 2020; Hoffmann et al., 2022). This creates a key difficulty for looped Transformers: recurrent computation is not free. If a model is looped N times during pre-training, its training compute is also multiplied by approximately N. Therefore, a looped model should not merely outperform a vanilla Transformer with the same stored parameter count. To justify recurrence as a scaling strategy, it must compete with vanilla Transformers trained under the same pre-training compute budget.

This requirement is challenging. Prior work suggests that, under fixed-FLOP comparisons, a looped model can have less non-recurrent capacity than a standard Transformer baseline (Csordás et al., 2024; Frey et al., 2026b). This is consistent with the conventional view that recurrence mainly improves parameter efficiency but not necessarily compute efficiency (Lan et al., 2020; Takase & Kiyono, 2023). As a result, prior looped Transformers have not yet provided a clear path toward flagship models at the trillion-parameter scale with frontier-level performance.

Loopie is designed to address this gap. Our central observation is that recurrence becomes competitive under a fixed compute budget only when it is paired with an appropriate recurrence-width-depth trade-off. We formalize this principle as the **Loopie Recipe**: rather than treating loops as a direct substitute for additional parameters, one should jointly choose stored width, stored depth, and recurrent depth so that the resulting model maximizes performance under a fixed pre-training compute budget. This changes the role of recurrence from a parameter-saving device into a compute-matched scaling mechanism.

J.1 WHY ONLY TWO LOOP STEPS?

Many recurrent language model designs use substantially deeper unrolling. For example, prior looped or latent-recurrent models often use 4, 16, or even more than 30 recurrent steps (Zhu et al., 2025; Geiping et al., 2025). These settings are useful for studying recurrence as a general mechanism for iterative computation. Loopie targets a different regime: large-scale language model pre-training under a fixed compute budget. In this regime, the number of loop steps is not merely an architectural hyperparameter; it is a direct allocation of pre-training FLOPs.

At a fixed stored parameter count and a fixed number of optimizer steps, increasing the number of loop steps usually improves the training curve. However, this comparison is not compute-matched. For a model with R loop steps, the per-token training cost scales approximately linearly with R, up to MoE routing and implementation constants:

$$\mathcal{C}(D, L, R) \propto LD^2R.$$

Under the same pre-training budget, the model must either be trained on fewer tokens, use a smaller stored architecture, or be compared against a stronger non-recurrent model that spends the same compute on ordinary Transformer capacity. The relevant question is thus not whether a model with $R = 4$ outperforms one with $R = 2$ at the same stored size but whether the marginal gain from

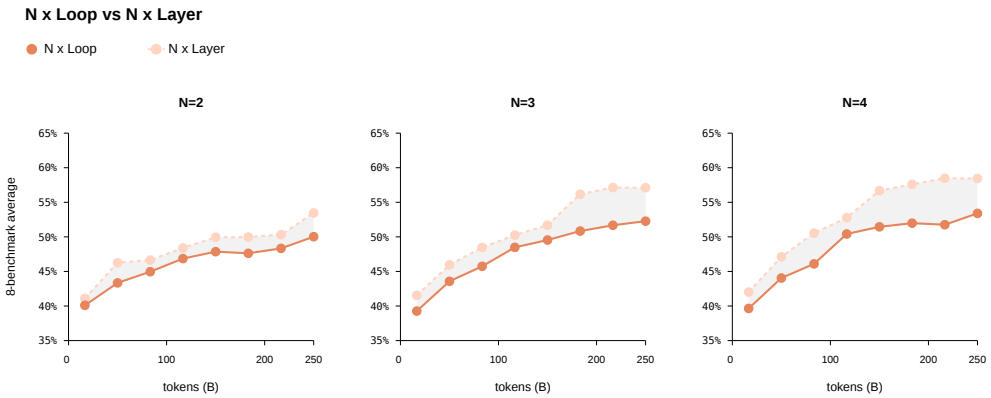


Figure 10: Comparison between $N\times$ layer-loop and $N\times$ stored-layer scaling. We report the average score across eight downstream benchmarks after training each model on 250B tokens while keeping all other architectural details fixed. The shaded gray region denotes the gap between the $N\times$ layer baseline and the $N\times$ loop model.

additional recurrence exceeds the gain that could be obtained by reallocating the same FLOPs to width, depth, or data.

Our loop-count sweep suggests that this marginal return decays rapidly in the large-scale pre-training regime. To isolate whether the gain comes from recurrence rather than simply adding more non-shared layers, we compare a model that uses $N\times$ layer-loop steps with one that has $N\times$ as many stored layers, while holding all other architectural details fixed. Figure 10 reports the average score across eight downstream benchmarks after the models are trained on 250B tokens. The shaded gray region highlights the performance gap between the $N\times$ layer baseline and the $N\times$ loop model. For the $2\times$ comparison, we use a $2\times$ stored-layer model only as a conservative proxy for the previous compute-matched setting. Because this model is very small, with only 0.25B active parameters, precise compute matching is difficult under multiple practical constraints, so the actual training compute of the $2\times$ layer baseline is substantially higher than that of the $2\times$ loop model. Thus, the figure should not be read as evidence that $2\times$ looping is dominated by $2\times$ layer scaling; rather, it shows that, when comparing $N\times$ looping against $N\times$ stored-layer scaling, the marginal benefit of recurrence is largest at $R = 2$.

We therefore choose $R = 2$ for the Loopie Series. This is the smallest nontrivial recurrent setting: each layer performs one ordinary transformation followed by one local refinement step before passing its representation to the next layer. This permits recurrence to change the computation qualitatively while keeping the compute multiplier small enough for scalable pre-training. It also preserves training throughput and makes the comparison against Qwen3-like vanilla MoE baselines stringent: Loopie must outperform these baselines not by using a large amount of extra recurrent computation but by using a small amount of recurrence more effectively.

This choice should not be interpreted as claiming that larger loop counts are ineffective in isolation. Larger R may be useful in settings where inference-time computation is cheap, where adaptive computation is available, or where the goal is to study recurrent reasoning rather than pre-training efficiency. For frontier-scale pre-training, however, the dominant constraint is total compute. Under this constraint, $R = 2$ provides the best trade-off we observe between iterative refinement, throughput, and compute-matched scaling.

K RESULTS OF LOOPIE-6B-A0.6B

Table 9: Comparison of Loopie-6B-A0.6B with similarly sized MoE reasoning models across knowledge, general, code, and math benchmarks.

	DeepSeek-R1 Distill-Qwen 1.5B	Gemma-4 E2B-it	Gemma-4 E4B-it	Qwen3 1.7B	MiniCPM5 1B	Ouro 1.4B Thinking	Ouro 2.6B Thinking	Loopie 6B-A0.6B
Knowledge								
MMLU	44.85	63.33	72.97	69.08	61.54	72.40	82.70	78.36
MMLU-Redux	51.28	72.28	79.61	74.28	70.63	73.75	86.28	81.35
General								
ARC-Easy	69.19	87.08	92.97	91.20	85.27	90.87	94.28	93.77
ARC-Challenge	61.09	83.28	90.02	86.09	75.34	88.48	93.94	91.13
SciQ	44.10	86.80	94.40	87.70	82.48	91.60	94.90	90.10
Code								
MBPP	35.80	77.04	82.49	80.54	50.97	88.63	95.69	76.65
MBPP+	63.76	86.77	90.21	82.80	81.75	89.89	96.01	84.66
HumanEval	65.85	79.27	91.46	85.98	86.59	95.12	95.12	84.15
HumanEval+	65.24	81.71	85.98	82.93	86.59	87.80	89.02	79.88
Math								
AIME 24	35.83	38.33	49.17	49.58	47.50	50.83	62.50	80.42
AIME 25	24.17	26.25	35.42	35.00	38.75	44.17	51.67	70.83
AMC	67.91	75.37	85.08	73.88	28.36	78.36	85.82	84.33
MATH-500	83.60	89.00	91.60	90.60	60.00	92.20	92.20	93.80
GSM8K	72.18	90.60	93.40	90.14	76.35	94.09	96.13	93.63