ARLOOP: STRONG FULLY-OPEN LOOPED TRANSFORMERS WITH ATTENTION RESIDUALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped transformers increase effective depth by repeatedly applying a shared set of transformer layers, offering a parameter-efficient way to scale computation. However, standard residual connections aggregate information across recurrent depth into a single stream, providing limited control over which intermediate representations from preceding layers should be reused. In this paper, we introduce ARLOOP, a looped transformer architecture based on attention residuals (Kimi Team, 2026), where the input of each layer is selectively aggregated through attention over model depth. Distinct from standard transformers, we find that looped transformers do not require access to the full residual history: sliding-window depth attention matches or outperforms full-history attention at larger recurrence depths. Across controlled experiments with multiple parameter scales (140M and 370M) and recurrence depths ($R = 1$–5), ARLOOP consistently outperforms prior residual designs for looped transformers. More importantly, we further scale ARLOOP to 0.6B and 1.3B parameters under a realistic LLM pretraining recipe, demonstrating that the gains transfer to realistic pretraining, with particularly strong improvements on math and coding tasks. Our 1.3B model, pretrained on only 0.5T tokens with three recurrences, outperforms the fully-open looped LLM Huginn-3.5B across all evaluated tasks and is competitive with similar-sized non-looped models such as Qwen3.5-2B despite being trained on substantially less data.[1]

1 INTRODUCTION

Looped transformers repeatedly apply a small set of shared transformer layers, increasing effective depth without increasing the number of distinct parameters (Dehghani et al., 2019; Yang et al., 2024; Saunshi et al., 2025). This provides an appealing alternative to conventional depth scaling: the same parameters can be reused for additional computation, and the number of recurrences provides an explicit axis along which training and inference compute can be varied. Recent work has explored looped transformers for parameter-efficient pretraining of large language models (LLM) (Geiping et al., 2025; Zhu et al., 2025). A central question, however, is whether these models can effectively *exploit the additional computation provided by recurrent depth*.

Increasing recurrent depth repeatedly produces new intermediate representations, but standard transformer residual connections, such as PreNorm (Xiong et al., 2020), provide only a limited mechanism for controlling how information produced at earlier depths is reused. Under PreNorm, residual updates from previous depths are accumulated into a single residual stream with fixed unit coefficients. As recurrence increases, more updates are accumulated into the same state, while later computations have no explicit mechanism for selecting which representations produced at earlier depths are useful. Existing work has improved looped transformers through residual scaling or normalization (Geiping et al., 2025; Li et al., 2026; Prairie et al., 2026), but it remains unclear which residual design best enables models to make use of recurrent computation. Moreover, prior residual designs have mostly been evaluated under relatively small-scale pretraining setups, leaving their effectiveness on more complex generative tasks such as mathematical reasoning and code generation underexplored.

Motivated by recent advances in attention residuals for standard transformers (Kimi Team, 2026), we investigate how selective residual access should be adapted to recurrent depth. Compared with

[1] Our code and model checkpoints will be released publicly.

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.

PreNorm, attention residuals replace the fixed aggregation of earlier residual updates with attention-weighted aggregation, allowing each layer to selectively access representations from preceding depths. Unlike standard transformers, however, recurrent depth repeatedly applies the same parameters and can produce highly redundant representations across recurrences, raising the question of whether access to the full residual history is necessary.

Our contributions are twofold. First, we introduce ARLOOP, a looped transformer architecture that adapts attention residuals to recurrent depth. Across multiple parameter scales (140M and 370M) and recurrence depths ($R = 1$ to $R = 5$), ARLOOP consistently improves validation perplexity over PreNorm and outperforms prior residual designs (Geiping et al., 2025; Li et al., 2026; Prairie et al., 2026). Distinct from standard transformers, we find that access to the full residual history is unnecessary for looped transformers: sliding-window depth attention matches or outperforms full-history attention at larger recurrence depths. Our analysis further shows that depth attention concentrates on recent residual sources and that representations across recurrences are highly redundant, explaining the effectiveness of sliding-window attention.

More importantly, we demonstrate that ARLOOP scales to a realistic LLM pretraining recipe. Despite using a much smaller training budget (i.e. 0.5T tokens), our strongest ARLOOP-1.3B substantially outperforms previous large looped language models (i.e. Huginn-3.5B (Geiping et al., 2025)), and achieves performance competitive with strong dense LLMs such as Qwen3.5-2B. At scale, we find that ARLOOP generally outperforms PreNorm, with especially promising improvements on math and coding tasks (e.g. +5.2 points on GSM8K and +3.1 points on MBPP). ARLOOP will be fully open: we will release its code, checkpoints, and training recipe.

2 RELATED WORK

Looped transformers. Looped transformers, also known as recurrent-depth transformers or universal transformers (Dehghani et al., 2019; Lan et al., 2020; Yang et al., 2024), are widely studied as memory-efficient alternatives to transformers. As they are increasingly scaled to LLM pretraining (Csordás et al., 2024; Geiping et al., 2025; Bae et al., 2025), recent work has explored residual designs for better information propagation across recurrent depth, including Li et al. (2026); Zeitoun et al. (2026); Prairie et al. (2026); Knupp et al. (2026). Compared with these approaches, ARLOOP consistently matches or outperforms them across parameter and recurrence budgets in our controlled experiments. We further show that these gains extend beyond language modeling perplexity to mathematical reasoning and code generation, which remain underexplored in prior work.

Training looped transformers as LLMs. Recent work has begun scaling looped transformers for pretraining. Geiping et al. (2025) introduced Huginn, a 3.5B-parameter looped LLM trained with up to 64 recurrences, while Zhu et al. (2025) further scaled the training data with Ouro, a 1.4B model using up to four recurrences. Subsequent work has explored looped LLMs for agentic capabilities and mixture-of-experts architectures (Gao et al., 2026; Wang et al., 2026). Despite their promising results, these studies largely leave residual design unexplored. In particular, Huginn reports that PreNorm does not effectively exploit recurrent depth and instead relies on conservative optimization setups for stable training. Our work systematically studies residual design for looped transformers and further validates the resulting architecture under a realistic LLM pretraining recipe, substantially outperforming Huginn despite using smaller parameter, data, and recurrence budgets.

3 ARLOOP

We first describe the formulation of our looped transformer and then how we adapt attention residuals for it. We refer to the resulting model as ARLOOP.

3.1 PRELIMINARIES

Terminology. Throughout the paper, we use *sublayer* to refer to either a self-attention or a feed-forward network (FFN) module, *transformer layer* to refer to a pair of self-attention and FFN sublayers, and *residual block* to refer to a group of consecutive sublayers used as a source in attention residuals, as introduced below.

Standard Transformer with PreNorm residual. We start by defining a standard decoder-only transformer (Vaswani et al., 2017; Radford et al., 2019) with a PreNorm residual stream. Let x be a sequence of tokens, and let $\boldsymbol{h}_0 \in \mathbb{R}^{n\times d}$ denote their embeddings. A standard transformer consists of L layers, each of which contains a self-attention sublayer followed by an FFN. For the i-th sublayer, its input h_{i-1} and output h_i can be written as

$$\boldsymbol{h}_i = \boldsymbol{h}_{i-1} + f_i(\mathrm{Norm}(\boldsymbol{h}_{i-1})), \quad i = 1, \ldots, 2L \tag{1}$$

where f_i denotes either a self-attention or an FFN function. We can then define $\boldsymbol{h_i} = \mathcal{F}_i(h_{i-1})$, and the entire transformer can be written as $\boldsymbol{y} = \mathrm{LMHead} \circ \mathcal{F}_{2L} \circ \cdots \circ \mathcal{F}_1(\boldsymbol{h}_0)$, where LMHead denotes the language-modeling head that maps the final hidden states to output logits.

Looped transformer. Following the notation above, an L-layer looped transformer reuses the same transformer stack $\mathcal{S}$ (e.g. $\mathcal{F}_4 \circ \mathcal{F}_3 \circ \mathcal{F}_2 \circ \mathcal{F}_1$) for R recurrences. Although $\mathcal{S}$ can consist of any subset of the L distinct transformer blocks, in this paper we only consider looping the full stack of transformer blocks (i.e. $\mathcal{S} = \mathcal{F}_{2L} \circ \cdots \circ \mathcal{F}_1$). Hence, our looped transformer can be written as $\boldsymbol{y} = \mathrm{LMHead} \circ \mathcal{S}^{(R)} \circ \cdots \circ \mathcal{S}^{(1)}(\boldsymbol{h}_0)$, where $\mathcal{S}^{(1)} = \cdots = \mathcal{S}^{(R)} = \mathcal{S}$, since the same transformer stack is shared across all recurrences. The resulting unrolled network therefore has an effective depth of $2LR$ sublayers, while containing only $2L$ distinct transformer sublayers.

Attention Residuals. Unrolling Equation 1 gives us $\boldsymbol{h}_i = \boldsymbol{h}_0 + \sum_{j=1}^{i} \boldsymbol{f}_j(\mathrm{Norm}(\boldsymbol{h}_{j-1}))$, which implies that under PreNorm, every residual update produced at an earlier sublayer contributes to all subsequent residual states with a fixed unit coefficient. Attention Residuals (Kimi Team, 2026) replace this fixed aggregation with attention-based aggregation over residual sources across depth. Specifically, we define the input embedding as $\boldsymbol{z}_0 := \boldsymbol{h}_0$, and formulate attention residuals as

$$\boldsymbol{z}_j = f_j\left(\mathrm{Norm}(\boldsymbol{h}_{j-1})\right), \qquad \alpha_{i,j} = \mathrm{Softmax}_{j\in\{0,\ldots,i\}}\left(\boldsymbol{q}_i^\top \mathrm{Norm}(\boldsymbol{z}_j)\right), \tag{2}$$

$$\boldsymbol{h}_i = \alpha_{i,0}\boldsymbol{h}_0 + \sum_{j=1}^{i} \alpha_{i,j}\boldsymbol{z}_j. \tag{3}$$

where $\boldsymbol{q}_i$ is a learned query vector of the i-th sublayer. Compared with PreNorm (i.e. Equation 1), Attention Residuals (i.e. Equation 3) assign an attention-based coefficient $\alpha_{i,j}$ to each residual update $\boldsymbol{z}_j$, rather than aggregating all residual updates with unit coefficients.

Equation 3 introduces a new residual source after every sublayer, causing the number of sources to grow linearly with effective depth. Following Kimi Team (2026), we therefore use *block attention residuals*, which group consecutive sublayers and aggregate their residual updates into block-level representations. Specifically, for a block of S consecutive sublayers, we define its representation as

$$\boldsymbol{b}_k = \sum_{j=(k-1)S+1}^{kS} \boldsymbol{z}_j \tag{4}$$

At each sublayer, attention is computed over the input embedding, the representations of preceding completed blocks, and the partial sum of residual updates within the current block. Thus, residual updates are accumulated additively within a block as in PreNorm, while attention-based aggregation is performed across blocks. This reduces the number of stored residual sources from individual sublayer outputs to block-level representations, substantially reducing the overhead of full attention residuals. We use block attention residuals for all AttnRes models unless stated otherwise.

3.2 Applying Attention Residuals to Looped Transformer

Since attention residuals only change how each sublayer aggregates representations from preceding depths, they can be naturally applied to looped transformers. Figure 1a illustrates a simplified example in which each recurrence contains two sublayers. As the depth grows, residual updates are grouped into one or more residual blocks following Equation 4. Each effective sublayer has its own learned query vector (i.e. q_i in Figure 1b), which attends over the residual sources available at that depth. Compared with PreNorm-based looped transformers, this allows sublayers at deeper recurrences to selectively access representations produced at earlier recurrent depths.

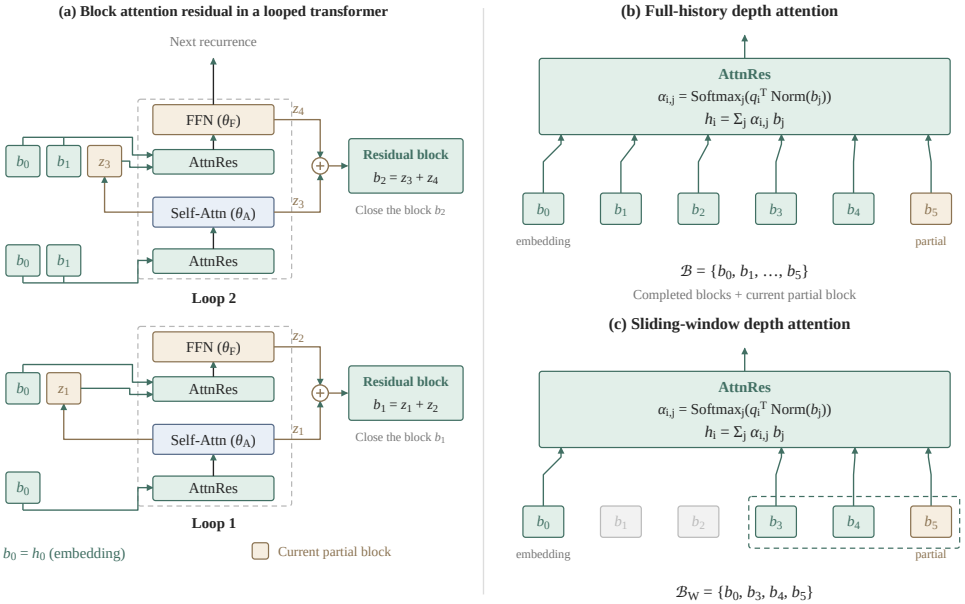


Figure 1: An example illustrating attention residuals in looped transformers. **(a)** Each loop contains one transformer layer (i.e. a self-attention and an FFN), with parameters shared across loops. We group every $S = 2$ sublayers into a residual block, yielding $b_1 = z_1 + z_2$ and $b_2 = z_3 + z_4$. **(b)** Full-history depth attention (Kimi Team, 2026) accesses all available residual sources. **(c)** Sliding-window depth attention retains the embedding and a window of recent residual sources.

Sliding window depth attention. Standard attention residuals allow each depth-attention operation to access all preceding residual blocks as candidate sources, which we refer to as *full-history depth attention*. In a looped transformer, however, the same set of parameters is repeatedly applied across recurrences, potentially introducing redundancy among representations produced at different recurrent depths. Indeed, prior work has observed high similarity between hidden states across recurrences in looped transformers (Zeitoun et al., 2026). This motivates us to ask whether access to the full residual history is necessary, or whether recent residual states already provide sufficient information.

Specifically, we study if sliding-window depth attention (Figure 1c) can match full-history depth attention. Let $\mathcal{B} = \{b_0, b_1, \ldots, b_m\}$ denote the residual sources available to a depth-attention operation, where b_0 denotes the input embedding and b_m is the most recent residual block. Given a window size W, sliding-window depth attention restricts the source set to $\mathcal{B}_W = \{b_0, b_{\max(1,m-W+1)}, \ldots, b_m\}$. Thus, the input embedding remains directly accessible at every depth, while depth attention is performed only over the W most recent residual blocks. Note that sliding-window depth attention has also been studied in standard transformers, where it reduces the gains of attention residuals over PreNorm (Kimi Team, 2026). In contrast, as we will show in Section 4, sliding-window depth attention can match and even outperform full-history attention in looped transformers, particularly at larger recurrence depths. We therefore make sliding-window depth attention the default in ARLOOP for $R > 2$.

4 CONTROLLED EVALUATION OF ARLOOP

We first use controlled experiments to evaluate how effectively ARLOOP reuses information across recurrent depth, examining both its benefits for language modeling and the contribution of its key design choices.

General setup. We evaluate ARLOOP under a controlled setup using 140M- and 370M-parameter decoder-only transformers with 6 and 12 transformer layers, respectively. Following Prairie et al. (2026), we pretrain these models on 11.2B and 29.6B tokens from FineWeb-Edu (Penedo et al.,

Model	140M			370M	
	$R=1$	$R=3$	$R=5$	$R=1$	$R=3$
PreNorm	18.37	16.91	16.34	13.55	12.48
Sandwich-Norm (Geiping et al., 2025)	18.98	17.19	16.59	13.92	12.97
DeepLoop (Li et al., 2026)	18.72	17.05	16.52	13.69	12.78
Parcae (Prairie et al., 2026)	18.37	16.60	16.01	13.55	12.31
Parcae + input injection (Prairie et al., 2026)	18.37	16.55	15.96	13.55	12.30
ARLOOP	**18.26**	**16.38**	**15.84**	**13.32**	**12.17**

Table 1: Validation perplexity (↓) for models at the 140M and 370M scales.

2024), respectively, with a context length of 2048. Following the general architecture of Karpathy (2025), each transformer layer uses multi-head attention and a standard MLP. For optimization, we use AdamW (Kingma & Ba, 2017) with a cosine learning rate schedule. Complete architecture and training details are provided in Appendix A.1.

ARLOOP training setup. For all ARLOOP models, we use fixed recurrence, such that the number of recurrences R is fixed throughout training. For the 140M and 370M models, we group every 6 and 12 consecutive sublayers, respectively, into a residual block. Since each recurrence contains 12 sublayers at 140M and 24 sublayers at 370M, this yields two residual blocks per recurrence at both scales. Following the design in Section 3.2, we use full-history depth attention for $R \leq 2$ and sliding-window depth attention for $R > 2$. For the sliding-window variant, we retain the input embedding and the three most recent residual blocks.

Baselines. Our compared baselines include PreNorm, which uses the residual in Equation 1, and previously proposed residual designs for looped transformers, including Sandwich-Norm (Geiping et al., 2025; Zhu et al., 2025), DeepLoop (Li et al., 2026), and Parcae (Prairie et al., 2026) with input injection (Bansal et al., 2022). For this comparison, we train all models with a fixed number of recurrences $R = 1, 3, 5$ at the 140M scale and $R = 1, 3$ at the 370M scale due to computational cost. PreNorm is the default baseline, and we reproduce other methods under our training setup while keeping all training hyperparameters fixed except for the learning rate; details of learning-rate tuning are provided in Appendix A.1.

Besides comparison with prior methods, we also train two variants of ARLOOP to better understand our method. Specifically, we train (1) $\text{ARLOOP}_{\text{full}}$, which applies full-history depth attention; and (2) $\text{ARLOOP}_{\text{sw}}$, which applies sliding-window depth attention, and compare them with the PreNorm baseline. We train each model with a fixed number of recurrences, using one model for each $R \in \{2, 3, 4\}$.

Evaluation. We use perplexity (PPL) on a held-out validation set of 10M tokens from Fineweb-Edu as the main evaluation metric. In addition, we also evaluate performance on a broad set of downstream tasks. Our *downstream* aggregate is the unweighted mean over ARC-Challenge, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA under both the OpenAI and standard formulations (Paperno et al., 2016), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2019), RACE (Lai et al., 2017), SciQ (Welbl et al., 2017), and WinoGrande (Sakaguchi et al., 2019). We use normalized accuracy for HellaSwag, OpenBookQA, and PIQA, and accuracy for the remaining tasks. Complete evaluation details are provided in Appendix A.2.

4.1 RESULTS

ARLOOP consistently outperforms existing residual designs in language modeling. We first compare the validation perplexity of ARLOOP in Table 1 against residual designs proposed in prior work. Across different parameter, data, and recurrence budgets, ARLOOP consistently achieves the lowest validation perplexity among PreNorm and the other residual designs, demonstrating the effectiveness of selective access to representations from earlier depths. Compared with PreNorm, ARLOOP yields larger gains at deeper recurrences (e.g. -0.11 PPL at $R = 1$ versus -0.53 at

$R = 3$), suggesting that ARLOOP makes more effective use of the additional computation provided by recurrent depth.

Sliding-window attention matches full-history attention at shallow recurrence, and even outperforms it in perplexity at deeper recurrences. We next ask whether looped transformers need direct access to the full residual history as recurrent depth increases. Table 2 compares PreNorm, full-history depth attention (ARLOOP$_{\text{full}}$), and sliding-window depth attention (ARLOOP$_{\text{sw}}$). In terms of perplexity, although ARLOOP$_{\text{sw}}$ performs worse than full-history depth attention (i.e. ARLOOP$_{\text{full}}$) at $R = 2$, it achieves the best performance at larger recurrence depths. Both variants also generally outperform PreNorm in downstream accuracy, suggesting that the effectiveness of our method extends to general knowledge and reasoning tasks. Given that the sliding-window variant also reduces computational overhead, we therefore adopt ARLOOP$_{\text{full}}$ for $R \leq 2$ and use the sliding-window variant ARLOOP$_{\text{sw}}$ as the default ARLOOP for $R > 2$.

Scale	R	PPL ↓			Downstream ↑		
		PreNorm	ARLOOP$_{\text{full}}$	ARLOOP$_{\text{sw}}$	PreNorm	ARLOOP$_{\text{full}}$	ARLOOP$_{\text{sw}}$
140M	2	17.48	**17.27**	17.30	42.22	41.96	**42.27**
	3	16.91	16.46	**16.38**	42.72	**43.69**	43.09
	4	16.59	16.07	**16.03**	43.27	**44.09**	43.73
370M	2	12.83	**12.68**	12.85	48.98	48.70	**49.05**
	3	12.48	12.27	**12.17**	49.29	50.23	**50.67**
	4	12.27	12.01	**11.99**	50.19	**50.93**	50.76

Table 2: Validation perplexity and average zero-shot downstream accuracy (%). Bold marks the best result for each metric within each row.

Why sliding-window depth attention works for looped transformers. To understand why recent residual blocks are sufficient, we visualize the attention weights assigned to each residual source by ARLOOP$_{\text{full}}$ with $R = 4$ in Figure 2 (Left). Depth attention places most of its mass on recent residual sources, suggesting that direct access to distant residual history contributes relatively little to the representations used at deeper recurrences. We hypothesize that this locality arises from parameter sharing across recurrences: repeatedly applying the same transformer layers produces redundant representations, while more recent sources have already undergone refinement through additional recurrent computation. Consequently, retaining direct access to distant residual sources may provide little additional information and could even introduce unnecessary interference.

To examine this hypothesis, we further visualize the pairwise cosine similarity between residual blocks in Figure 2 (Right). Residual sources across recurrences exhibit high similarity, supporting the presence of substantial redundancy across recurrent depth. We observe the same pattern across other parameter scales and recurrence settings, with complete results provided in Appendix A.5. Importantly, this behavior differs from a standard transformer with matched effective depth, where representations exhibit substantially lower cross-depth similarity, and applying the same sliding-window restriction degrades performance (Appendix A.5). Together, these results suggest that local depth attention is particularly well suited to the recurrent-depth structure of looped transformers.

5 SCALING TO PRACTICAL PRETRAINING OF LOOPED TRANSFORMERS

Having established the effectiveness of ARLOOP in controlled experiments, we now scale it to billion-parameter models, bringing together carefully curated pretraining data mixtures and a multi-stage training strategy to build strong, fully-open ARLOOP-based LLMs.

Setup. We adopt a pretraining recipe similar to SmolLM (Allal et al., 2025; HuggingFaceTB, 2025). Compared with the controlled experiments in Section 4, we replace multi-head attention with grouped-query attention (Ainslie et al., 2023) and use SwiGLU (Shazeer, 2020) in place of the standard FFN sublayers to better align with modern LLM architectures (Qwen Team, 2025). For pretraining data, we use FineWeb-Edu (Penedo et al., 2024), FineMath, and Stack-Edu (Allal et al., 2025) as the primary sources. We use the same cosine learning rate schedule and adopt a multi-stage training strategy, introducing higher-quality data in later stages following HuggingFaceTB (2025); Allen Institute for AI (2025). We also extend the context length to 4096 tokens. Such a realistic pretraining recipe enables us to evaluate our models not only on general knowledge and reasoning

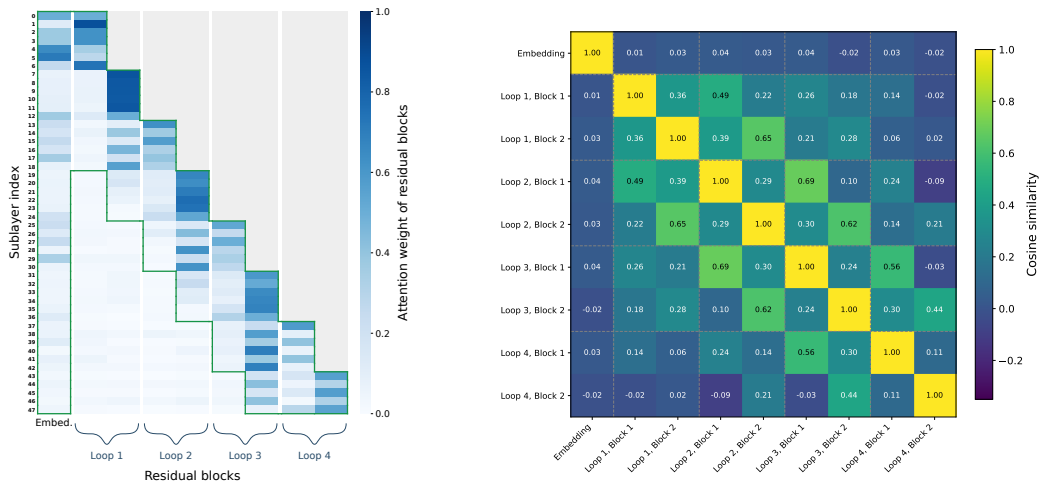


Figure 2: **Left**: Attention weights over residual sources (i.e. blocks) at each sublayer for the 140M ARLOOP$_{\text{full}}$ model with $R = 4$, averaged over the validation set. Green lines indicate the sliding windows used in the corresponding ARLOOP$_{\text{sw}}$ variant. **Right**: Pairwise cosine similarity between residual blocks for the same model.

tasks, but also on mathematical reasoning and code generation. We provide complete architecture, training, and data-mixture details in Appendix B.1.

Using this recipe, we train three models: (1) PRENORM-0.6B, a 0.6B PreNorm model with 18 transformer layers and $R = 3$, trained on 180B tokens; (2) ARLOOP-0.6B, a 0.6B ARLOOP model with the same architecture, recurrence, and training budget; and (3) ARLOOP-1.3B, a 1.3B ARLOOP model with 24 transformer layers and $R = 3$, trained on 500B tokens. For both ARLOOP models, we adopt the sliding-window configuration from Section 4.

Evaluation. We use ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), and OpenBookQA (Mihaylov et al., 2018) to evaluate general knowledge and reasoning; GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021; Lewkowycz et al., 2022) for mathematical reasoning; and MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021) for code generation. Complete evaluation details are provided in Appendix B.2.

Baselines. We additionally compare against both existing looped language models and strong non-looped LLMs of similar or larger sizes. The recurrent-depth baselines include Huginn (Geiping et al., 2025) and Ouro (Zhu et al., 2025), while the non-looped open models include Qwen3.5-2B (Qwen Team, 2026), Gemma 3-4B (Gemma Team, 2025), OLMo 2-7B, and OLMo 3-7B (Team OLMo et al., 2024; Allen Institute for AI, 2025). The non-looped open models are included mainly as references rather than for head-to-head comparison. For all of these models, we re-evaluate the corresponding base checkpoints using the same evaluation harness to ensure consistent evaluation. Among these models, we particularly focus on Huginn because it is a fully-open looped LLM trained with a comparable data scale, but with substantially larger parameter and recurrence budgets (e.g. 3.5B parameters and up to 64 recurrences).

Note that we do not compare with other residual methods such as Parcae or DeepLoop (Prairie et al., 2026; Li et al., 2026; Zeitoun et al., 2026) in this realistic pretraining comparison. Such previous work generally validates their methods only at small data scales (e.g. below 50B tokens) or with a single pretraining source (e.g. FineWeb-Edu), leaving the capabilities of the resulting models on more complex tasks such as math and coding open. Our contribution specifically targets this question, aiming to show that ARLOOP remains effective on such generative tasks under realistic pretraining settings.

Model	FLOPs	Data (T)	Recur.	ARC-C	Hella.	Wino.	OBQA	GSM8K	MATH	MBPP	HumanEval
Open-weight LLMs											
Qwen3.5-2B	432	36	No	54.3	67.5	66.4	37.8	60.6	35.8	49.5	32.9
Gemma 3-4B	96	4.0	No	58.4	77.7	72.3	43.0	38.1	25.4	60.1	34.2
Ouro-1.4B	259	7.7	4	59.9	74.4	70.5	41.4	78.2	70.5	68.3	52.4
Fully-open LLMs											
OLMo 2-7B	180	4.1	No	64.6	82.0	76.8	45.6	68.1	20.9	32.0	12.2
OLMo 3-7B	252	5.9	No	61.0	76.4	73.8	42.4	73.8	43.6	66.1	14.0
Huginn-3.5B	28	0.18	32	33.6	50.6	52.7	33.4	5.1	3.7	14.8	10.3
Huginn-3.5B	127	0.8	32	44.2	68.0	62.4	40.0	26.1	14.1	33.3	20.7
PRENORM-0.6B (Ours)	3	0.18	3	46.3	60.6	61.2	39.8	35.7	20.9	33.1	17.6
ARLOOP-0.6B (Ours)	3	0.18	3	45.5	62.5	61.8	40.8	40.9	26.3	36.2	20.1
ARLOOP-1.3B (Ours)	15	0.5	3	52.8	69.7	65.2	43.0	61.5	33.6	50.0	28.1

Table 3: Comparison of open-weight and fully-open language models. Training FLOPs are reported in units of 10^{21}; Data (T) denotes pretraining tokens in trillions. Recur. denotes the number of recurrences at evaluation. Rows in light blue refer to looped language models, including our fully-open models.

5.1 RESULTS

ARLOOP scales to realistic LLM pretraining, with substantial improvements on math and coding over PreNorm. Table 3 reports our large-scale training results. Under the same 0.6B parameter scale, 180B-token training budget, and $R = 3$ recurrence, ARLOOP-0.6B outperforms PRENORM-0.6B on 7 of 8 evaluated tasks. The gains are particularly large on mathematical reasoning and code generation, including +5.2 points on GSM8K and +3.1 points on MBPP. These results suggest that the benefits of ARLOOP indeed transfer to a realistic LLM pretraining recipe. Interestingly, while the gains in validation perplexity are relatively modest in the controlled experiments, they become substantially larger on complex generative tasks. This highlights the importance of evaluating residual designs beyond perplexity and multiple-choice benchmarks, which is missing in existing work (Prairie et al., 2026; Li et al., 2026; Zeitoun et al., 2026).

Further scaling makes ARLOOP competitive with strong LLMs at a similar scale. Scaling ARLOOP to 1.3B parameters and 500B training tokens yields substantial improvements across all evaluated benchmarks. In particular, ARLOOP-1.3B outperforms Huginn on every task in Table 3, despite using fewer parameters, fewer training tokens, and substantially fewer recurrent steps. Although ARLOOP-1.3B underperforms Ouro (Zhu et al., 2025) on most tasks, we attribute this gap primarily to their substantially larger pretraining data budget and its use of higher-quality pretraining data sources. Instead, our training recipe relies on open and commonly used data sources. Our model is also competitive with Qwen3.5-2B despite its substantially smaller pretraining budget, matching or exceeding it on HellaSwag, OpenBookQA, GSM8K, and MBPP. These results show that attention residuals can substantially strengthen looped LLMs, making them competitive with strong dense LLMs at similar parameter scales.

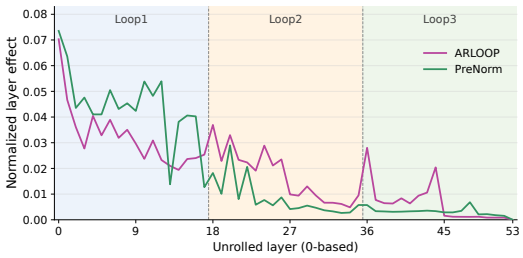


Figure 3: Layer effects on 50 MATH examples for PRENORM-0.6B and ARLOOP-0.6B. Shaded regions mark Loop1 (layers 0–17), Loop2 (18–35), and Loop3 (36–53).

Deeper recurrence in ARLOOP contributes more strongly to subsequent computation on math and coding tasks. To better understand the gains of ARLOOP on math and coding tasks, we adopt the layer effect metric from Csordás et al. (2025) to quantify how much each layer contributes to representations at subsequent layers. We randomly sample 50 examples from MATH and, for each example, intervene on the representation of a source layer and measure the resulting change in the outputs of deeper layers. Figure 3 compares the layer effects between PRENORM-0.6B

and ARLOOP-0.6B. We find that layers in the second and third recurrences of ARLOOP make substantially larger contributions to subsequent layers than the corresponding deeper layers in PRENORM-0.6B. This suggests that ARLOOP makes greater use of representations computed at later recurrent depths, consistent with its stronger performance on tasks that benefit from deeper computation. We report the detailed score computation and results on MBPP in Appendix C.

6 FURTHER DISCUSSION

Dynamic versus fixed recurrence. Existing looped language models often vary the number of recurrences during training to support multiple effective depths (Geiping et al., 2025; Prairie et al., 2026; Bae et al., 2025). We compare such dynamic-recurrence training with fixed $R = 4$ training and find that fixed recurrence consistently achieves better validation perplexity and downstream performance at both the 140M and 370M scales. We therefore use fixed recurrence in our main experiments. Complete training setups and results are provided in Appendix D.1.

Inference-time scaling. We also investigate whether the improved residual access of ARLOOP enables inference-time recurrence extrapolation, i.e. benefiting from more recurrences than observed during training (Fan et al., 2025; Kohli et al., 2026). Consistent with prior observations for looped language models (Prairie et al., 2026; Zhu et al., 2025), we find that performance improves up to the maximum training recurrence but does not further improve beyond it. This suggests that selective access to earlier representations alone does not unlock recurrence extrapolation: effectively composing information within the trained depth range and generalizing recurrent computation beyond that range appear to be distinct problems. Complete experimental details are provided in Appendix D.2.

Training efficiency. ARLOOP introduces additional computation due to the attention residual calculation for each sublayer. When activation checkpointing (e.g. recomputation of intermediate hidden states) is not enabled, the residual sources used by attention residual are already retained for backpropagation, resulting in negligible additional memory overhead. We report training memory and throughput for our experimental setups in Table 4. Across all model scales, ARLOOP uses essentially the same amount of memory as PreNorm. In terms of throughput, ARLOOP$_{\text{full}}$ is approximately 15% slower than PreNorm on average, while sliding window attention residual reduces this gap to approximately 10%. We expect that further kernel optimization could reduce this overhead, as depth attention introduces relatively little additional arithmetic computation but incurs additional high-bandwidth memory traffic.

Model	140M	370M	0.6B	1.3B
		Memory (GiB) ↓		
PreNorm	32.8	80.0	66.5	65.4
ARLOOP$_{\text{full}}$	33.0	80.1	66.6	65.4
ARLOOP$_{\text{sw}}$ SW	32.9	80.1	66.6	65.4
		Throughput (tokens/s) ↑		
PreNorm	131.1k	45.8k	28.9k	15.0k
ARLOOP$_{\text{full}}$	107.5k	39.1k	24.8k	13.0k
ARLOOP$_{\text{sw}}$	114.5k	40.9k	25.9k	13.6k

Table 4: Training memory and throughput at $R = 4$ on a single NVIDIA H100, with batch sizes of 8, 8, 4, and 2 for the 140M, 370M, 0.6B, and 1.3B models, respectively.

7 CONCLUSION

We introduce ARLOOP, a looped transformer architecture that adapts attention residuals to recurrent depth. Through controlled experiments across multiple parameter scales, ARLOOP consistently outperforms PreNorm and prior residual designs for looped transformers. Distinct from standard transformers, we find that looped transformers do not require access to the full residual history: sliding-window depth attention can match or outperform full-history attention at larger recurrence depths. More importantly, we demonstrate that these gains transfer to realistic LLM pretraining. At the 0.6B scale, ARLOOP substantially improves over a matched PreNorm model, with particularly large gains on math and coding tasks. Our strongest model ARLOOP-1.3B substantially outperforms the fully-open looped LLM Huginn and is competitive with Qwen3.5-2B despite being trained on substantially less data. Together, these results demonstrate that selective residual access provides an effective way for looped transformers to better exploit recurrent computation at scale.

AI USE STATEMENT

In this work, we used generative AI tools to help refine research hypotheses, provide feedback on experimental design, and assist with the implementation of the training pipeline and model architectures. We did not use generative AI tools to formulate mathematical claims, provide critical ingredients for mathematical proofs, assist in writing proofs, or interpret experimental results. Other required-disclosure tasks, including synthetic dataset generation, theoretical model or conceptual framework development, translation, dataset cleaning or reformatting, and qualitative or thematic data analysis, were not applicable to this work.

Additionally, we used generative AI tools to create or modify scientific figures, polish the writing, and suggest improvements to the structure of the paper. We reviewed all AI-assisted work. In particular, we manually reviewed AI-assisted code used in the core implementation and verified its correctness through testing and empirical checks.

We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to facilitate reproduction of our results. The architecture and training configurations for the controlled experiments are described in Appendix A.1, with evaluation protocols in Appendix A.2 and additional ablations and per-task results in Appendix A.3-A.6. For the large-scale pretraining experiments, Appendix B.1 specifies the model architectures, optimization settings, pretraining stages, and data mixtures, while Appendix B.2 provides the complete benchmark evaluation protocols. Appendices C , D.1, D.2 describe the setups for the layer-effect analysis, dynamic-recurrence experiments, and recurrence extrapolation study. We will publicly release our code, model checkpoints, and training recipe to further support reproducibility.

REFERENCES

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. 2025. URL https://arxiv.org/abs/2504.01943.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big—data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Allen Institute for AI. Olmo-3-1025-7B model card, 2025. URL https://huggingface.co/allenai/Olmo-3-1025-7B.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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, 2025. URL https://arxiv.org/abs/2507.10524.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D Manning. MoEUT: Mixture-of-experts universal transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=ZxVrkm7Bjl.

Róbert Csordás, Christopher D. Manning, and Christopher Potts. Do language models use their depth efficiently?, 2025. URL https://arxiv.org/abs/2505.13898.

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.

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

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

Jonas Geiping, Sean Michael McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. URL https://openreview.net/forum?id=D6o6Bwtq7h.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

HuggingFaceTB. SmolLM3-3B-Base model card, 2025. URL https://huggingface.co/HuggingFaceTB/SmolLM3-3B-Base.

Andrej Karpathy. nanochat: The best chatgpt that $100 can buy, 2025. URL https://github.com/karpathy/nanochat.

Kimi Team. Attention residuals, 2026. URL https://arxiv.org/abs/2603.15031.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. Depth-recurrent attention mixtures: Giving latent reasoning the attention it deserves, 2026. URL https://arxiv.org/abs/2601.21582.

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.

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

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.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Shuzhen Li, Yifan Zhang, Jiacheng Guo, Quanquan Gu, and Mengdi Wang. Deeploop: Depth scaling for looped transformers, 2026. URL https://arxiv.org/abs/2607.13491.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023. URL https://arxiv.org/abs/2305.01210.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

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

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

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

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qwen Team. Qwen3.5-2B-Base model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-2B-Base.

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

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

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.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 OLMo 2 Furious. *arXiv preprint arXiv:2501.00656*, 2024.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture, 2020. URL https://arxiv.org/abs/2002.04745.

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

Weizhe Yuan, Jane Yu, Song Jiang, Karthik Padthe, Yang Li, Dong Wang, Ilia Kulikov, Kyunghyun Cho, Yuandong Tian, Jason E Weston, and Xian Li. Naturalreasoning: Reasoning in the wild with 2.8m challenging questions, 2025. URL https://arxiv.org/abs/2502.13124.

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?, 2019. URL https://arxiv.org/abs/1905.07830.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning, 2025. URL https://arxiv.org/abs/2410.17897.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Controlled experiment details

A.1 Architecture and Training.

Table 5 specifies the 140M and 370M model families used in Section 4. The table describes the shared backbone of PreNorm, $\text{ARLOOP}_{\text{full}}$, and $\text{ARLOOP}_{\text{sw}}$. Each recurrence reuses the entire physical layer stack; increasing R increases the effective depth without adding Transformer layers. Depth-attention queries and normalization parameters are specific to each effective sublayer. At the final effective depth, we apply a separate attention-residual aggregation to produce the representation passed to the language modeling head. We use a scaled-zero initialization, which is the same strategy as Prairie et al. (2026). The additional baselines retain the same model dimensions and use their respective residual updates and initialization rules.

Value residuals. Following the default setup of Karpathy (2025), we use value residuals (Zhou et al., 2025) in all experiments. Specifically, at alternating effective layers (e.g., the 1st, 3rd, and 5th layers), we inject a gated token embedding into the value vectors of the self-attention sublayer. These token-specific embedding parameters are distinct from the input embeddings. Value residual learning has been shown to improve LLM pretraining (Zhou et al., 2025) and has also been adopted in recent looped-transformer architectures (Prairie et al., 2026).

Although value residuals are not a contribution of our work, we find them important for obtaining strong looped-transformer performance, particularly at smaller scales. Importantly, removing value residuals substantially degrades our sliding-window depth-attention variant. They are even more important for Sandwich-Norm looped transformers: without value residuals, training becomes unstable and the loss remains near 7.5 rather than continuing to decrease. This pattern is consistent with the failed Sandwich-Norm training run reported by Geiping et al. (2025, Figure 5, Bad Run 1). We hypothesize that this instability is related to representation collapse at large effective depths, where representations of different tokens become increasingly similar, and repeated injection of token-specific information through value residuals mitigates this effect. Empirically, adding value residuals eliminates the observed training failure and makes the Sandwich-Norm variant trainable in our setting.

Training configuration. We train on 11.2B tokens at 140M and 29.6B tokens at 370M, using a 32,768 vocabulary and packed FineWeb-Edu data. We train the vocabulary using BPE (Sennrich et al., 2016) on FineWeb-Edu data, following Prairie et al. (2026). The global batch size is 256 sequences of length 2,048, or 524,288 tokens per optimizer update. We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient clipping at 1.0. The cosine learning-rate schedule has 1,000 warmup updates. Runs use BF16 mixed precision. Validation uses a fixed set of 4,096 sequences at the model's training recurrence, which is approximately 10M tokens.

When comparing against residual designs from related work, we empirically observe that Sandwich-Norm requires a lower learning rate than PreNorm. We therefore sweep learning rates over $\{8 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}, 4 \times 10^{-3}, 6 \times 10^{-3}, 8 \times 10^{-3}\}$ and select the best learning rate separately

		140M		370M	
		PreNorm	ARLOOP	PreNorm	ARLOOP
Architecture	Physical Transformer layers L	6	6	12	12
	Hidden dimension d_{model}	768	768	1,024	1,024
	MLP dimension $d_{\text{intermediate}}$	3,072	3,072	4,096	4,096
	Attention heads (query / key–value)	6 / 6	6 / 6	8 / 8	8 / 8
	Head dimension	128	128	128	128
Shared details	Attention	Causal multi-head attention with QK-Norm			
	MLP activation	ReLU^2			
	Position embeddings	RoPE ($\theta = 50{,}000$)			
	Vocabulary size	32,768			
	Normalization	RMSNorm ($\epsilon = 10^{-5}$)			
	Context length	2,048			
	Linear biases	None			
	Tied input/output embeddings	Yes			
Recurrence	Training / evaluation recurrence R	Fixed, $R \in \{1, 2, 3, 4, 5\}$ (depending on comparison)			
	Effective Transformer depth LR	$6R$	$6R$	$12R$	$12R$
	Residual block size (sublayers)	–	6	–	12
	Transformer layers per AttnRes block	–	3	–	6
	Sliding window W	–	3 + embedding	–	3 + embedding

Table 5: Architecture details for the controlled 140M and 370M experiments. Each Transformer layer contains an attention sublayer and an MLP sublayer. The AttnRes block and window settings apply to the corresponding residual variants; Table 2 lists the evaluated configurations. PreNorm uses additive pre-normalized residual updates, while ARLOOP learns the input mixture at each effective sublayer.

Benchmark	**Settings / reported metric**	**Framework**
	Language modeling	
FineWeb-Edu (held out)	PPL	
	Zero-shot downstream evaluation	
ARC-Challenge	Log-likelihood, 0-shot; accuracy	lm-eval-harness
ARC-Easy	Log-likelihood, 0-shot; accuracy	lm-eval-harness
HellaSwag	Log-likelihood, 0-shot; normalized accuracy	lm-eval-harness
LAMBADA (OpenAI)	Last-word prediction, 0-shot; accuracy	lm-eval-harness
LAMBADA (standard)	Last-word prediction, 0-shot; accuracy	lm-eval-harness
OpenBookQA	Log-likelihood, 0-shot; normalized accuracy	lm-eval-harness
PIQA	Log-likelihood, 0-shot; normalized accuracy	lm-eval-harness
RACE	Log-likelihood, 0-shot; accuracy	lm-eval-harness
SciQ	Log-likelihood, 0-shot; accuracy	lm-eval-harness
WinoGrande	Log-likelihood, 0-shot; accuracy	lm-eval-harness

Table 6: Evaluation protocol for the controlled 140M and 370M experiments.

for PreNorm and Sandwich-Norm (Geiping et al., 2025). For the 140M models, we use a learning rate of 6×10^{-3} for PreNorm and 2×10^{-3} for Sandwich-Norm. For the 370M models, we use 4×10^{-3} for PreNorm and 1×10^{-3} for Sandwich-Norm. Parcae (Prairie et al., 2026) and our attention residual methods use the same learning rate as PreNorm to enable comparison against the strongest baseline. DeepLoop (Li et al., 2026) uses the same learning rate as Sandwich-Norm, as both methods apply residual normalization and empirically favor lower learning rates.

A.2 EVALUATION SETUP

Table 6 specifies the metrics used in Section 4. For downstream evaluation, we use the language model evaluation harness (Gao et al., 2024) with the same task definitions across residual designs. Normalized accuracy divides candidate log-likelihoods by the task's length normalization before selecting an answer.

A.3 Ablation without value residuals

Table 7 reports results for the 140M, $R = 3$ setting with gated value embeddings disabled. The corresponding results with value residuals enabled are reported in Table 1. *Collapse* indicates that the training loss fails to decrease, with token representations becoming nearly identical, due to residual normalization. Even without value residuals, ARLoop continues to outperform prior residual designs, showing that its gains are not attributable to the use of value residuals.

Model	140M, $R = 3$, w/o value res. (Val PPL ↓)
PreNorm	18.19
Sandwich-Norm (Geiping et al., 2025)	Collapse
DeepLoop (Li et al., 2026)	18.44
Parcae + input injection (Prairie et al., 2026)	17.83
ARLoop$_{\text{full}}$	**17.52**

Table 7: Validation perplexity without value residuals. Bold marks the best available result.

A.4 Per-task results for controlled pretraining

Table 8 reports the individual task accuracies underlying the averages in Table 2, grouped by model scale.

R	Residual design	ARC-C	ARC-E	HellaSwag	LAMBADA OpenAI	LAMBADA Standard	OBQA	PIQA	RACE	SciQ	WinoGrande	Avg.
					140M							
1	PreNorm	21.84	**56.36**	**34.49**	**25.77**	**19.13**	**32.20**	63.49	28.33	82.40	50.04	41.41
	ARLoop$_{\text{full}}$	**23.72**	55.22	33.90	25.58	18.18	**32.20**	**64.15**	**28.90**	**83.90**	**52.49**	**41.82**
2	PreNorm	23.55	56.40	35.40	26.97	**21.56**	28.40	**63.93**	**32.34**	82.30	**51.38**	42.22
	ARLoop$_{\text{full}}$	**23.63**	55.51	35.64	**27.19**	19.85	29.80	63.87	30.72	**83.20**	50.20	41.96
	ARLoop$_{\text{sw}}$	22.61	**57.53**	**35.77**	26.99	21.35	**31.80**	63.11	30.53	81.80	51.22	**42.27**
3	PreNorm	24.40	56.23	36.08	29.58	22.43	29.80	**65.13**	29.95	82.10	51.46	42.72
	ARLoop$_{\text{full}}$	**24.74**	**58.67**	36.88	29.90	**23.04**	**32.00**	64.85	**30.14**	84.50	**52.17**	**43.69**
	ARLoop$_{\text{sw}}$	23.38	57.95	**37.23**	**30.70**	22.24	30.60	64.85	30.05	**85.00**	48.93	43.09
4	PreNorm	23.55	57.74	37.08	28.72	23.54	32.00	64.74	30.81	83.60	50.91	43.27
	ARLoop$_{\text{full}}$	24.32	**58.50**	**38.16**	**31.21**	23.75	**33.80**	**64.80**	30.14	**83.90**	**52.33**	**44.09**
	ARLoop$_{\text{sw}}$	**25.26**	57.83	38.12	30.31	**24.47**	32.00	64.36	**31.00**	82.70	51.22	43.73
					370M							
1	PreNorm	26.37	62.08	43.86	35.38	28.57	**35.40**	**68.93**	**32.25**	85.30	51.85	47.00
	ARLoop$_{\text{full}}$	**28.92**	**62.33**	**44.20**	**36.72**	**29.90**	33.20	67.63	31.48	**85.50**	**54.62**	**47.45**
2	PreNorm	30.03	64.14	46.57	37.34	29.15	36.40	**70.18**	**32.63**	87.50	**55.88**	48.98
	ARLoop$_{\text{full}}$	28.16	65.07	**47.15**	**37.45**	30.29	**36.60**	68.23	32.54	87.10	54.38	48.70
	ARLoop$_{\text{sw}}$	**30.12**	**66.20**	46.07	36.68	**30.47**	36.20	69.04	32.44	**88.70**	54.54	**49.05**
3	PreNorm	27.47	64.94	48.23	38.17	32.64	**38.20**	68.55	31.77	**88.90**	53.99	49.29
	ARLoop$_{\text{full}}$	29.86	**67.42**	49.63	37.76	**34.56**	37.60	69.70	**32.92**	88.40	54.46	50.23
	ARLoop$_{\text{sw}}$	**30.89**	65.82	**49.72**	**40.35**	34.17	37.40	**69.80**	**32.92**	88.30	**57.38**	**50.67**
4	PreNorm	30.46	64.90	49.31	39.12	34.74	36.80	69.10	33.88	88.00	**55.56**	50.19
	ARLoop$_{\text{full}}$	**30.89**	**65.74**	**50.92**	40.29	**34.78**	**39.00**	69.48	**33.97**	**89.10**	55.09	**50.93**
	ARLoop$_{\text{sw}}$	30.46	65.40	50.30	**41.80**	34.68	38.40	**70.95**	32.82	87.30	55.49	50.76

Table 8: Zero-shot downstream accuracy (%) for the 140M and 370M models in Table 2. HellaSwag, OpenBookQA (OBQA), and PIQA use normalized accuracy; all other tasks use accuracy. Training and evaluation use the same recurrence R; $R = 1$ is non-recurrent. ARLoop$_{\text{full}}$ and ARLoop$_{\text{sw}}$ use residual blocks of 6 sublayers at 140M and 12 at 370M; ARLoop$_{\text{sw}}$ retains the input embedding and the three most recent blocks. Bold indicates the highest accuracy within each scale and recurrence group.

A.5 Additional analysis of sliding-window depth attention

Here we present the complete analysis of sliding-window depth attention, extending Figure 2 to other model scales and recurrence settings. We also examine whether sliding-window depth attention is similarly effective for standard non-recurrent transformers under the same controlled experimental setup.

Depth-attention weights and residual block similarity across scales. Figure 4 reports the remaining ARLoop$_{\text{full}}$ configurations at 140M, $R = 3$, and 370M, $R \in \{3, 4\}$. Depth-attention weights are normalized over visible sources for visualization. Gray cells denote unavailable sources. Green outlines identify the sliding window we used in the corresponding ARLoop$_{\text{sw}}$, including the current partial block when present. Cosine similarity is computed token-wise between residual-block representations and averaged over 512 validation sequences of length 2,048. Across model scales and recurrence settings, both the attention weights and pairwise cosine similarities exhibit patterns consistent with those observed in Section 4.1.

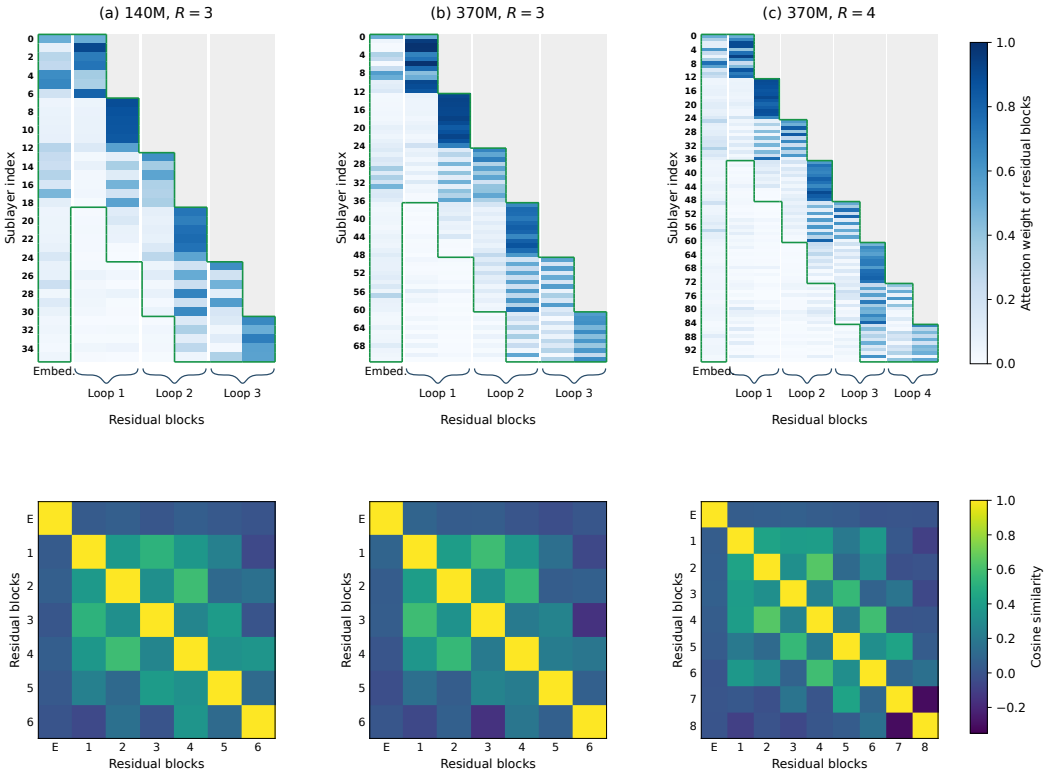


Figure 4: Additional full-history diagnostics: (a) 140M, $R = 3$; (b) 370M, $R = 3$; and (c) 370M, $R = 4$. Top: residual-source attention weights, with green outlines marking the embedding plus the three most recent blocks. Bottom: pairwise residual-block cosine similarity, with E denoting the embedding.

Non-recurrent comparison at matched effective depth. We additionally evaluate a non-recurrent transformer with 24 distinct layers, matching the 48 effective sublayers of the six-layer 140M looped model with $R = 4$. It retains the same 768-dimensional hidden states, six attention heads, block size of six sublayers, context length of 2,048, and 11.2B-token training budget. This comparison therefore matches the effective depth and width of the looped model, but not its parameter count.

As shown in Table 9, sliding-window depth attention slightly improves the looped model, whereas directly applying the same window to the non-recurrent transformer substantially degrades performance. To better understand this difference, we further visualize the pairwise cosine similarity between residual blocks in the non-recurrent model in Figure 5. The non-recurrent transformer exhibits a markedly

Architecture	Full history	Sliding window ($W = 3$)	Δ PPL
Looped: $L = 6$, $R = 4$	16.07	16.03	-0.04
Non-recurrent: $L = 24$, $R = 1$	13.65	16.78	$+3.13$

Table 9: Validation perplexity at matched effective depth. Both models use the same training and architecture setups, except that non-recurrence models use more unique parameters to match the effective depth. The sliding window retains the embedding plus three blocks.

different pattern from the looped models, with substantially lower similarity across residual blocks. This suggests that distinct depths in standard transformers contain less redundant representations, consistent with our hypothesis that the high redundancy across recurrences in looped transformers makes sliding-window depth attention particularly well suited to their recurrent-depth structure.

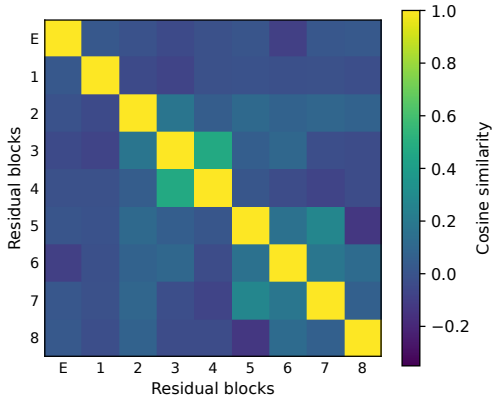


Figure 5: Residual-block similarity for the non-recurrent 24-layer, $W = 3$ model, measured on the same validation positions as the looped-model cosine diagnostics. E is the embedding; blocks 1–8 belong to one forward pass, not separate recurrences.

A.6 The effect of residual block and sliding window sizes

We additionally study the effect of residual block size, i.e. the number of sublayers contained in each residual block, and sliding-window size, i.e. the number of recent residual blocks retained by sliding-window depth attention. We primarily use the 140M ARLoop for these studies.

Table 10 reports the validation perplexities of models trained with different residual block sizes across recurrence settings. We find that using finer-grained residual blocks (i.e. smaller block sizes) does not consistently improve validation perplexity. While finer-grained blocks are beneficial for non-recurrent transformers, consistent with Kimi Team (2026), we find that their benefit diminishes as the number of recurrences increases, and smaller blocks can even hurt performance at larger R. We hypothesize that this is related to the same phenomenon observed in Section 4.1: residual-block representations are highly redundant across recurrences, and using smaller blocks introduces more residual sources from earlier recurrences, potentially further increasing this redundancy.

Table 11 reports the results of $\text{ARLoop}_{\text{sw}}$ with sliding windows retaining the most recent 2, 3, 4, or 5 residual blocks. Among the settings we evaluate, retaining three residual blocks performs best overall. We leave a more systematic exploration of the optimal sliding-window design to future work.

Model	Recurrence	Residual block size (sublayers)		
		2	4	6
140M 6 layers	Non-recurrent	**18.20**	18.25	18.26
	Loop $= 2$	17.19	**17.18**	17.27
	Loop $= 3$	16.59	16.48	**16.46**
	Loop $= 4$	16.13	**16.02**	16.07

Table 10: Validation PPL for completed 140M full-history block-size experiments. The best result within each model and recurrence is bolded.

Model	Recurrence	Sliding-window blocks W			
		2	3	4	5
140M ARLOOP$_{sw}$	4	16.17	**16.03**	16.07	16.14

Table 11: Validation PPL versus sliding-window size for 140M ARLOOP$_{sw}$ models with residual blocks of six sublayers. W counts retained residual blocks; the input embedding is always retained.

B LARGE-SCALE PRETRAINING DETAILS

B.1 ARCHITECTURE AND TRAINING

Architecture. Table 12 details the model backbones used in Section 5. Compared with the controlled setup, we replace multi-head attention with grouped-query attention to reduce the number of key and value heads, and replace the ReLU^2 MLP with SwiGLU. The full transformer stack is reused at each recurrence. For both model scales, ARLOOP uses the embedding and the three most recent residual blocks for sliding-window depth attention.

	PreNorm 0.6B	ARLOOP 0.6B	ARLOOP 1.3B
Physical Transformer layers L	18	18	24
Hidden dimension	1,536	1,536	2,048
SwiGLU intermediate dimension	4,608	4,608	5,632
Query / key–value heads	16 / 4	16 / 4	16 / 4
Head dimension	96	96	128
Residual block size (sublayers)	–	18	24
Fixed recurrence R	3	3	3
Effective depth (sublayers)	108	108	144
Residual blocks per recurrence	–	2	2
Attention	Causal grouped-query attention with QK-Norm		
MLP	SwiGLU		
Position embeddings	RoPE ($\theta = 50{,}000$)		
Tokenizer / vocabulary	SmolLM2 / 49,152		
Normalization	RMSNorm ($\epsilon = 10^{-5}$)		
Context length	4,096		
Linear biases	None		
Tied input/output embeddings	Yes		
Depth-attention window	–	Embedding + three recent blocks	
Training tokens	180B	180B	500B
Global batch (sequences)	256	256	512
Peak / minimum learning rate	3×10^{-3} / 3×10^{-5}	3×10^{-3} / 3×10^{-5}	2×10^{-3} / 2×10^{-5}

Table 12: Architecture and optimization settings for large-scale pretraining. Residual block/window settings apply only to ARLOOP. Each physical layer contains one attention and one MLP sublayer.

Pretraining recipe. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient clipping at 1.0. A cosine learning-rate schedule follows 1,000 warmup updates and decays to 1% of the peak learning rate. Table 12 reports the scale-specific peak learning rates and global batch sizes. Training uses BF16 mixed precision with packed sequences of 4,096 tokens.

Our pretraining recipe largely follows SmolLM (HuggingFaceTB, 2025) and consists of three training stages, with progressively larger proportions of math and coding data. Figure 6 summarizes how the domain mixture changes throughout training. The stage boundaries are placed at 60% and 80% of the total token budget. General web data dominate the first stage, while later stages increase the proportions of mathematical and coding data; the final stage additionally introduces reasoning-oriented data sources. The learning-rate schedule runs continuously across the full training budget, without restarting warmup or cosine decay at stage transitions.

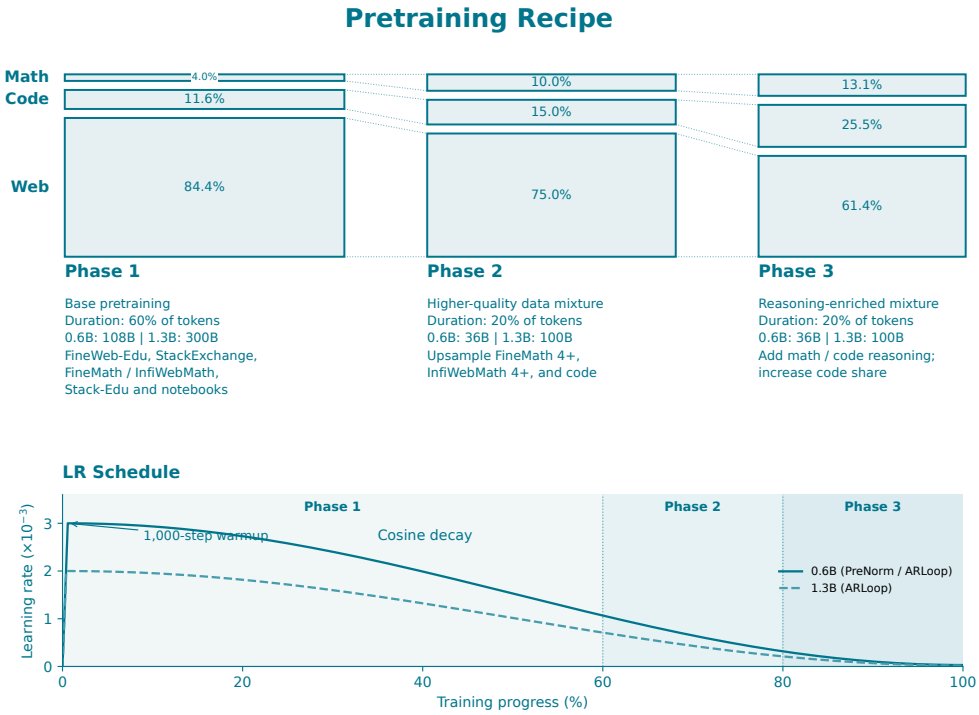


Figure 6: Three-stage pretraining recipe. The upper panels show the domain sampling shares at each stage. The lower panel shows the learning rate schedules: 1,000 warmup updates followed by cosine decay to 1% of the peak. Source-level allocations are listed in Table 14.

Data source mixture. The 0.6B and 1.3B models use the same data-source proportions within each stage, with total training budgets of 180B and 500B tokens, respectively. Stage 1 accounts for 60% of each training run, while Stages 2 and 3 each account for 20%. Table 14 reports the resulting per-stage token allocations. StackExchange is grouped with general data, while notebooks, GitHub issues, OpenCodeReasoning, and Stack-Edu are grouped with coding data. Dolmino Math aggregates several of its mathematical sub-sources, and Stack-Edu aggregates 15 programming-language subsets.

B.2 Evaluation setup

Table 13 summarizes the benchmark-specific evaluation settings. For ARC-Challenge, HellaSwag, and WinoGrande, we use few-shot evaluation to follow commonly used evaluation protocols for existing LLMs (Zhu et al., 2025). We use lm-evaluation-harness for evaluating general and mathematical reasoning tasks, and EvalPlus (Liu et al., 2023) for coding tasks.

For our models, lm-evaluation-harness evaluation is performed in BF16 with a 4,096-token context length, and no chat template. For mathematical reasoning and coding generation, we use greedy

decoding with a maximum of 1,024 new tokens. GSM8K reports the flexible-extract exact-match metric, while MATH reports Math-Verify scores. MBPP and HumanEval use the original EvalPlus completion prompts, stop conditions, and sanitization procedures. For the re-evaluated scores of prior LLMs reported in Table 3, we use each model's native tokenizer and the same task protocols specified in Table 13.

Benchmark	Settings / reported metric	Framework
General knowledge and reasoning		
ARC-Challenge	Log-likelihood, 25-shot; normalized accuracy	lm-eval-harness
HellaSwag	Log-likelihood, 10-shot; normalized accuracy	lm-eval-harness
WinoGrande	Log-likelihood, 5-shot; accuracy	lm-eval-harness
OpenBookQA	Log-likelihood, 0-shot; normalized accuracy	lm-eval-harness
Mathematical reasoning		
GSM8K	Generation, 5-shot CoT; flexible-extract EM	lm-eval-harness
MATH	Generation, 4-shot Minerva; Math-Verify	lm-eval-harness
Code generation		
MBPP	Generation, 0-shot; pass@1	EvalPlus
HumanEval	Generation, 0-shot; pass@1	EvalPlus

Table 13: Evaluation setup for large-scale pretraining. We evaluate pretrained base checkpoints without chat templates. CoT denotes chain-of-thought demonstrations. MATH uses the full 5,000-example test set. These settings also apply to the external checkpoints re-evaluated for Table 3.

C LAYER EFFECTS AND OUTPUT EFFECTS ON MBPP AND MATH

We follow the causal-intervention formulation of Csordás et al. (2025), adapted to our recurrent architecture. We compare the final ARLOOP-0.6B and PRENORM-0.6B checkpoints, each with 18 physical layers and three recurrences, yielding 54 effective layers. Within each task, both models are evaluated on the same randomly sampled 50 test examples, drawn from 378 examples for MBPP and 5,000 for MATH. We use zero-shot prompts and teacher forcing over the canonical solution code for MBPP or the full reference solution for MATH. For each unrolled layer s, we perform an intervention by ablating its self-attention and MLP sublayer contributions and rerun the forward pass on the same input.

Layer effect metric. Following Csordás et al. (2025), we quantify the effect of an intervention by measuring the change it induces relative to the original run. Let $\boldsymbol{a}_{\ell,t}$ and $\boldsymbol{m}_{\ell,t}$ denote the attention and MLP sublayer outputs, respectively, at unrolled layer ℓ and token position t. We define the local layer contribution as

$$\boldsymbol{c}_{\ell,t} = \boldsymbol{a}_{\ell,t} + \boldsymbol{m}_{\ell,t}. \tag{5}$$

For PreNorm, this corresponds to the additive update contributed by the transformer layer. We use the same quantity for ARLOOP so that the two architectures are compared using the same local computation, independently of how residual sources are subsequently aggregated.

Let $\bar{\boldsymbol{c}}^{(s)}_{\ell,t}$ denote the corresponding contribution after ablating layer s. For each later layer $\ell > s$, we measure the relative change in its computation as

$$E^{\text{layer}}_{s\to\ell,t} = \frac{\left\| \boldsymbol{c}_{\ell,t} - \bar{\boldsymbol{c}}^{(s)}_{\ell,t} \right\|_2}{\left\| \boldsymbol{c}_{\ell,t} \right\|_2}. \tag{6}$$

This measures the effect of intervening on layer s on the local contribution of a subsequent layer ℓ. For an example e, let $\mathcal{T}_e$ denote its teacher-forced solution positions. For each pair of intervention and later layers (s, ℓ), we first take the maximum $E^{\text{layer}}_{s\to\ell,t}$ over these positions, and then the maximum across the 50 examples. The layer effect of intervention layer s is then obtained by averaging these values over all strictly later layers.

Domain	Dataset source	0.6B model (180B tokens)				1.3B model (500B tokens)			
		Stage 1	Stage 2	Stage 3	Total	Stage 1	Stage 2	Stage 3	Total
General	FineWeb-Edu (Penedo et al., 2024)	89.02	26.36	22.06	137.44	247.29	73.22	61.27	381.78
	StackExchange	2.16	0.64	0.04	2.84	6.00	1.78	0.10	7.88
Math	FineMath 3+ (Allal et al., 2025)	1.08	0.36	0.07	1.51	3.00	1.00	0.20	4.20
	InfiWebMath 3+ (Allal et al., 2025)	1.08	0.36	0.07	1.51	3.00	1.00	0.20	4.20
	FineMath 4+ (Allal et al., 2025)	1.08	1.96	0.90	3.94	3.00	5.44	2.50	10.94
	InfiWebMath 4+ (Allal et al., 2025)	1.08	0.92	0.72	2.72	3.00	2.56	2.00	7.56
	OpenMathReasoning (Moshkov et al., 2025)	–	–	0.36	0.36	–	–	1.00	1.00
	Natural Reasoning (Yuan et al., 2025)	–	–	0.36	0.36	–	–	1.01	1.01
	OpenMathInstruct2 (Toshniwal et al., 2024)	–	–	2.04	2.04	–	–	5.66	5.66
	Dolmino Math (Team OLMo et al., 2024)	–	–	0.18	0.18	–	–	0.51	0.51
Coding	OpenCodeReasoning (Ahmad et al., 2025)	–	–	2.47	2.47	–	–	6.87	6.87
	Kaggle notebooks (Lozhkov et al., 2024)	0.37	0.02	0.02	0.41	1.02	0.07	0.06	1.15
	Stack v2 Jupyter (Lozhkov et al., 2024)	1.08	0.36	0.54	1.98	3.00	1.00	1.51	5.51
	GitHub Issues (Lozhkov et al., 2024)	0.18	0.04	0.07	0.30	0.51	0.12	0.20	0.83
	Stack-Edu (Allal et al., 2025)	10.87	4.97	6.09	21.93	30.19	13.82	16.91	60.91
Total		**108.00**	**36.00**	**36.00**	**180.00**	**300.00**	**100.00**	**100.00**	**500.00**

Table 14: Source-level allocation (billion tokens for each cell) for large-scale pretraining.

Output effect metric We additionally measure how each intervention affects the model output. Let $\boldsymbol{p}_t$ and $\bar{\boldsymbol{p}}_t^{(s)}$ denote the full-vocabulary probability distributions before and after ablating layer s. Following Csordás et al. (2025), we measure the output effect using their L_2 distance:

$$E_{s,t}^{\text{out}} = \left\| \boldsymbol{p}_t - \bar{\boldsymbol{p}}_t^{(s)} \right\|_2 . \tag{7}$$

As above, we take the maximum over teacher-forced solution positions and then across the 50 examples for each intervention layer.

Finally, for both metrics, we normalize the resulting 54-dimensional effect vector by its sum across intervention layers. The resulting curves therefore show how the effects are distributed across the 54 unrolled layers rather than their absolute magnitudes.

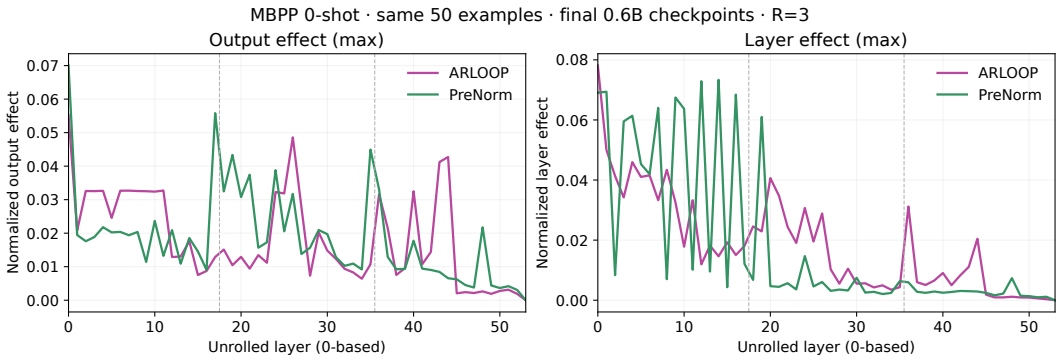


Figure 7: Output influence (left) and layer-contribution influence (right) on 50 shared zero-shot MBPP examples. Curves compare the ARLOOP-0.6B and PRENORM-0.6B checkpoints at $R = 3$, using max aggregation and normalization by each curve's total influence. Dashed lines separate the three recurrences.

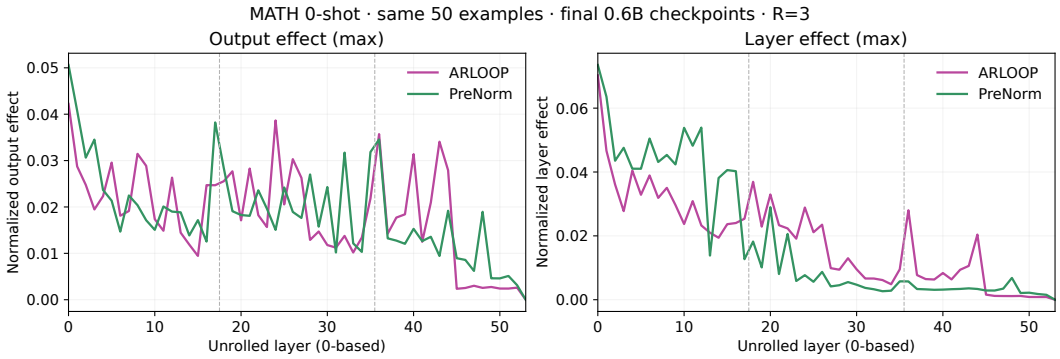


Figure 8: Output influence (left) and layer-contribution influence (right) on 50 shared zero-shot MATH examples, with the same aggregation and normalization as Figure 7. The right panel supplies the data for Figure 3. Dashed lines separate the three recurrences.

D Dynamic recurrence and recurrence extrapolation

This section provides the setup and detailed results for the investigations summarized in Section 6.

D.1 Training setup and fixed-versus-dynamic results

Setup. The models use the controlled-pretraining architectures and data described in Appendix A.1, with 140M and 370M models, except that the recurrence number is not fixed across training. We consider three recurrence-training strategies: ARLoop$_{\text{full}}$ (i.e. fixed recurrences), ARLoop-Poisson-Seq, and ARLoop-Poisson-Batch. ARLoop uses full-history depth attention trained with fixed $R = 4$, rather than the default sliding-window variant used for deep recurrence in the main comparison. For dynamic recurrence, we consider two strategies: (1) per-sequence recurrence sampled from a Poisson distribution (Prairie et al., 2026); and (2) per-batch recurrence sampled from a Poisson distribution (Geiping et al., 2025).

For the Poisson variants, we sample

$$\tilde{R} \sim \text{Poisson}(4), \qquad R = \min\Big(4, \max\Big(1, \tilde{R}\Big)\Big)\,, \tag{8}$$

In the per-sequence setting, recurrence depth is sampled independently for each sequence, whereas the per-batch variants use a single sampled depth for the entire batch. Because per-batch dynamic training requires fewer FLOPs on updates that sample shallow recurrence depths, we train these models on proportionally more tokens so that the total training FLOPs are matched. Fixed and per-sequence runs use 11.2B/29.6B tokens at 140M/370M. Per-batch Poisson runs use 13.6B/36.2B tokens.

Results. Table 15 presents the results of this comparison. At evaluation recurrence $R = 4$, fixed-depth training achieves the best validation perplexity and downstream average among the tested configurations at each scale. Hence we adopt fixed recurrences in our main experiment setups. Despite the negative results of dynamic recurrences, we do not claim that it should be replaced by fixed recurrences. Such dynamic variants allow flexible computation allocation during inference, and are still a popular choice for looped LLMs.

D.2 Inference-time recurrence extrapolation

We present the details of the study on inference-time recurrence extrapolation of ARLoop.

Setup. We evaluate the per-sequence Poisson model described above at each recurrence $R = 1, \ldots, 8$, using the same validation set as in Section 4. During training, the recurrence is bounded by $R \leq 4$; evaluation at $R > 4$ therefore requires computation at effective depths that are never encountered during the original training. Since our attention-residual mechanism uses a learned query

Eval R	Training recurrence	140M		370M	
		Val PPL ↓	downstream avg. ↑	Val PPL ↓	downstream avg. ↑
4	ARLOOP$_{\text{full}}$ ($R = 4$)	**16.07**	**44.09**	**12.01**	**50.93**
	ARLOOP-POISSON-SEQ	16.69	43.24	12.33	49.62
	ARLOOP-POISSON-BATCH	16.52	43.30	12.36	50.53

Table 15: Fixed versus dynamic recurrence training. Dynamic Poisson draws have mean 4 and are clamped to $[1, 4]$. Bold marks the best result at each scale.

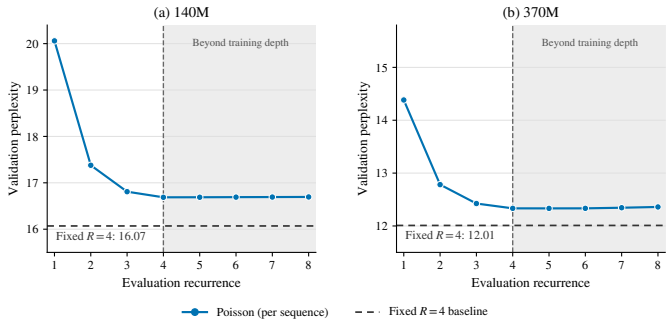


Figure 9: Validation perplexity versus evaluation recurrence for the per-sequence Poisson dynamic full-history ARLOOP models in Table 15. Each curve evaluates the same completed checkpoint at $R = 1, \ldots, 8$. The vertical dashed line at $R = 4$ marks the maximum training recurrence; the gray shaded region denotes evaluation beyond this depth. Horizontal dashed lines retain the fixed-$R = 4$ baseline reference levels (16.07 at 140M and 12.01 at 370M), not recurrence sweeps of the fixed models. Panels use separate vertical scales.

for each effective sublayer, evaluating beyond the maximum training recurrence introduces queries at previously unseen depths. We randomly initialize these additional queries and train only their parameters on an additional 0.5B FineWeb-Edu tokens, while freezing all other model parameters. This provides a favorable setting for recurrence extrapolation: rather than requiring the model to reuse or extrapolate query parameters to unseen depths, the newly introduced queries are explicitly optimized to aggregate the residual history available at those depths.

Results. Figure 9 shows that perplexity does not continue to improve beyond the maximum training recurrence at either model scale. This remains the case even when the query parameters at unseen depths are explicitly trained to learn the optimal aggregation of information from preceding depths. These results support the conclusion in Section 6: in our experiments, improved access to residual history alone is insufficient to produce useful recurrence extrapolation beyond the depths encountered during training.

E TRAINING COST

We train our models on NVIDIA A100 80GB and H100 GPUs. The controlled experiments require approximately 12K GPU hours in total, including early-stage exploration. Pretraining the 0.6B PreNorm and ARLOOP models takes approximately one week on 8 H100 GPUs. The 1.3B ARLOOP model is trained on 32 H100 GPUs for approximately two weeks.