Hyperloop Transformers

Abbas Zeitoun **Lucas Torroba-Hennigen** **Yoon Kim**

Massachusetts Institute of Technology

{zeitoun,lucastor,yoonkim}@mit.edu

Abstract

LLM architecture research generally aims to maximize model quality subject to fixed compute/latency budgets. However, many applications of interest such as edge and on-device deployment are further constrained by the model's memory footprint, thus motivating *parameter-efficient* architectures for language modeling. This paper describes a simple architecture that improves the parameter-efficiency of LLMs. Our architecture makes use of looped Transformers as a core primitive, which reuse Transformer layers across depth and are thus more parameter-efficient than ordinary (depth-matched) Transformers. We organize the looped Transformer into three blocks—begin, middle, and end blocks—where each block itself consists of multiple Transformer layers, and only the middle block is applied recurrently across depth. We augment the looped middle block with *hyper-connections* (Xie et al., 2026), which expand the residual stream into matrix-valued residual streams. Hyper-connections are applied only after each loop, and therefore add minimal new parameters and compute cost. Across various model scales, we find that our *Hyper-Connected Looped Transformer (Hyperloop Transformer)* is able to outperform depth-matched Transformer and mHC Transformer baselines despite using approximately 50% fewer parameters. The outperformance persists through post-training weight quantization, thus positioning Hyperloop Transformers as an attractive architecture for memory-efficient language modeling.

1 Introduction

Pushing the Pareto frontier of performance and efficiency is a major goal of modern LLM architecture research. In cloud deployment, efficiency is measured primarily by latency, which depends on both compute and memory movement. Because memory (DRAM) is relatively abundant in such environments, parameter efficiency is typically not the primary concern. This makes parameter-*in*efficient architectures such as mixture-of-experts (MoE; Shazeer et al., 2017) viable for cloud deployment. In contrast, edge and on-device deployments are often constrained not only by compute, but also by the total amount of available memory, which is often orders of magnitude smaller. For example, modern smartphones typically have 8GB–16GB of RAM, and only a portion of this memory is available for deployment of machine learning workloads. In such settings, a model's memory footprint becomes a major bottleneck, since it directly affects whether a model can be stored and executed at all. Even for cloud deployment, future frontier models may become so large that fitting the full model across a practical number of accelerators becomes challenging, making total parameter memory a key concern. This motivates the study of *parameter-efficient architectures* for language modeling, where the goal is to push the performance-memory frontier for a given compute constraint.

Looped Transformers[1] are Transformers that share parameters across depth, and thus enable greater parameter-efficiency than ordinary Transformers. When the number of loops is variable, they have also been shown to overcome certain theoretical limitations of fixed-depth

[1]Other terminology used to describe looped Transformers include *universal Transformers* (Dehghani et al., 2018; Tan et al., 2023), *recursive Transformers* (Bae et al., 2024; 2025) , and *recurrent-depth Transformers* (Geiping et al., 2025; Pappone et al., 2025).

Transformers (Giannou et al., 2023; Yang et al., 2023; Xu & Sato, 2024), and recent empirical work suggests that they can perform particularly well on some real-world reasoning tasks (Geiping et al., 2025; Zhu et al., 2025b). However, when matched for depth, looped Transformers still generally underperform unlooped baselines especially from a perplexity standpoint (Saunshi et al., 2025).

This paper develops a simple looped architecture that outperforms depth-matched Transformer baselines while using approximately half the parameters. Following prior work (Bae et al., 2025), we adopt a "middle cycle" strategy where we organize the Transformer into begin, middle, and end blocks, and only loop the middle block. We then incorporate a variant of *hyper-connections* (Zhu et al., 2025a; Xie et al., 2026), which expand the residual stream into multiple streams, into (only) the looped block. Specifically, we apply hyper-connections at the loop level (i.e., only after each loop iteration) instead of at the layer-level, thus incurring minimal additional parameters and compute. We find that our *Hyper-Connected Looped Transformer (Hyperloop Transformer)* improves the performance-parameter frontier, achieving lower perplexities than depth-matched ordinary Transformers with 240M, 1B, and 2B parameters, despite using 50% fewer parameters. These gains persist through post-training quantization of the model's weights, thus positioning Hyperloop Transformers as an attractive alternative to ordinary Transformers for memory-efficient language modeling.

2 Background

2.1 Looped Transformers

For a length T input, a Transformer transforms input representations at layer $\mathbf{X}^{(l)} \in \mathbb{R}^{T\times C}$ to obtain the output $\mathbf{X}^{(l+1)} \in \mathbb{R}^{T\times C}$ through an attention layer followed by an MLP layer,

$$\mathbf{H}^{(l)} = \text{Attention}(\mathbf{X}^{(l)}; \theta^{(l)}_{\text{attn}}) + \mathbf{X}^{(l)}, \qquad \mathbf{X}^{(l+1)} = \text{MLP}(\mathbf{X}^{(l)}; \theta^{(l)}_{\text{mlp}}) + \mathbf{H}^{(l)}.$$

Here $\theta^{(l)}_{\text{attn}}, \theta^{(l)}_{\text{MLP}}$ are the layer-specific parameters for multiheaded attention and the feed-forward layers respectively.[2] Letting $\mathcal{F}_l(\cdot)$ be the application of a Transformer layer l, an L-layer Transformer then obtains the final output via $\mathbf{X}^{(L)} = \mathcal{F}_L(\dots \mathcal{F}_2(\mathcal{F}_1(\mathbf{X}^{(1)}))\dots)$. Looped Transformers share parameters across depth, e.g., a fully looped model would have $\mathbf{X}^{(L)} = \mathcal{F}_1(\dots \mathcal{F}_1(\mathcal{F}_1(\mathbf{X}^{(1)}))\dots)$. More recent works have shown that a "middle cycle" strategy, which partitions the Transformer layers into beginning, middle, and end blocks[3] and only loops the middle block, is particularly effective (Bae et al., 2025; Saunshi et al., 2025). We also adopt this middle cycle strategy in our architecture.

2.2 Hyper-Connected Transformers

As shown above, each layer of a Transformer adds to the C-dimensional *residual stream*. Hyper-connected Transformers (Zhu et al., 2025a) expand the residual stream to an $n \times C$ dimensional matrix through "hyper-connections". In the more recent *manifold-constrained hyper-connections* (mHC; Xie et al., 2026), the residual stream at time step t at depth l (given by $\mathbf{x}^{(l)}_t \in \mathbb{R}^C$) is expanded by an expansion factor n to yield n parallel residual streams $\mathbf{y}^{(l)}_t \in \mathbb{R}^{n\times C}$. This expanded residual stream is then read from, written to, and mixed using input-dependent projections $\mathbf{H}^{\text{pre}}_{l,t}$, $\mathbf{H}^{\text{post}}_{l,t}$, and $\mathbf{H}^{\text{res}}_{l,t}$. Specifically, the transformations at depth l can be computed as follows:

$$\begin{aligned}
\mathbf{z}^{(l)}_t &= \text{RMSNorm}(\text{flatten}(\mathbf{y}^{(l)}_t)),\\
\mathbf{H}^{\text{pre}}_{l,t} &= \sigma(\alpha^{\text{pre}}_l \cdot (\mathbf{W}^{\text{pre}}_l \mathbf{z}^{(l)}_t) + \mathbf{b}^{\text{pre}}_l),\\
\mathbf{H}^{\text{post}}_{l,t} &= 2 \cdot \sigma(\alpha^{\text{post}}_l \cdot (\mathbf{W}^{\text{post}}_l \mathbf{z}^{(l)}_t) + \mathbf{b}^{\text{post}}_l),\\
\mathbf{H}^{\text{res}}_{l,t} &= \text{sinkhorn}(\alpha^{\text{res}}_l \cdot \text{reshape}(\mathbf{W}^{\text{res}}_l \mathbf{z}^{(l)}_t) + \mathbf{b}^{\text{res}}_l).
\end{aligned}$$

[2]The LayerNorm parameters are absorbed into the attention/MLP layers.

[3]The begin/end layers are also called prelude/coda or encoder/decoder blocks in the literature.

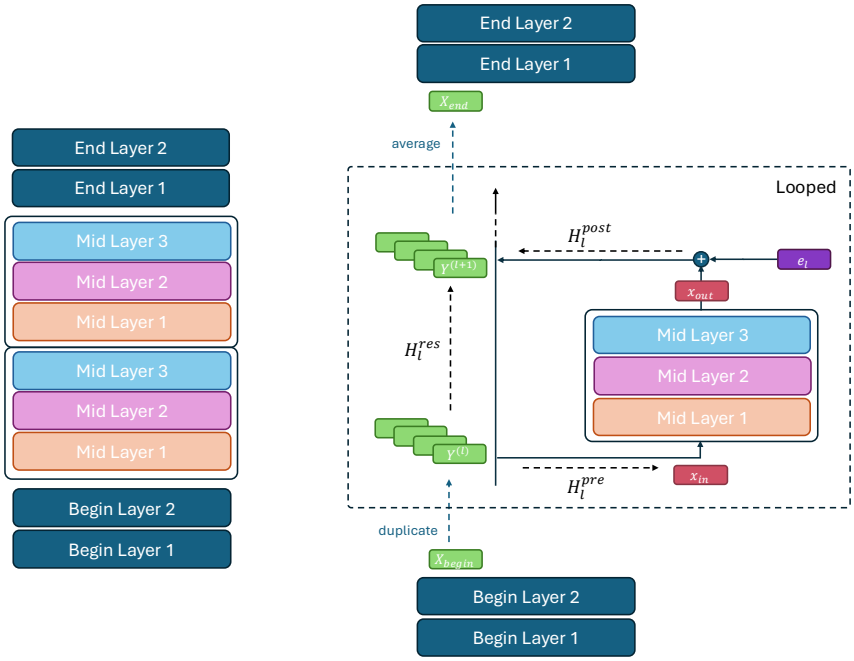

Figure 1: (Left) A vanilla middle-cycle looped Transformer architecture with two loops. (Right) A Hyper-Connected Looped Transformer, which uses parallel residual streams that are written to after each loop using hyper-connections.

Here $\mathbf{W}_l^{\text{pre}} \in \mathbb{R}^{n \times nC}$, $\mathbf{W}_l^{\text{post}} \in \mathbb{R}^{n \times nC}$, $\mathbf{W}_l^{\text{res}} \in \mathbb{R}^{n^2 \times nC}$ are linear projections, α_l^{pre}, α_l^{post}, $\alpha_l^{\text{res}} \in \mathbb{R}$ are learned scalars, $\mathbf{b}_l^{\text{pre}} \in \mathbb{R}^n$, $\mathbf{b}_l^{\text{post}} \in \mathbb{R}^n$, $\mathbf{b}_l^{\text{res}} \in \mathbb{R}^{n \times n}$ are learned biases, and $\text{reshape}(\cdot)$ is an operator that converts an n^2-dimensional vector to an $n \times n$ matrix. Finally, $\text{sinkhorn}(\cdot)$ applies the Sinkhorn-Knopp algorithm, which exponentiates the input and iteratively performs column- and row-normalization, ensuring that $\mathbf{H}_{l,t}^{\text{res}}$ is doubly stochastic (i.e., on the Birkhoff polytope) in the limit. Xie et al. (2026) find that 20 Sinkhorn-Knopp iterations are sufficient.

Given the input-dependent matrices $\mathbf{H}_{l,t}^{\text{pre}} \in \mathbb{R}^{1 \times n}$, $\mathbf{H}_{l,t}^{\text{post}} \in \mathbb{R}^{n \times 1}$, $\mathbf{H}_{l,t}^{\text{res}} \in \mathbb{R}^{n \times n}$ and a sub-layer $\mathcal{F}_l \in \{\text{Attention}_l, \text{MLP}_l\}$ of a Transformer layer, mHC applies attention/MLP layers in a smaller residual stream of dimension C via,[4]

$$\mathbf{y}_t^{(l+1)} = \mathbf{H}_{l,t}^{\text{res}} \mathbf{y}_t^{(l)} + \mathbf{H}_{l,t}^{\text{post}} \mathcal{F}_l(\mathbf{H}_{l,t}^{\text{pre}} \mathbf{y}_t^{(l)}).$$

Thus, mHC Transformers make it possible to work with a larger matrix-valued residual stream without incurring much additional compute (since the compute-heavy attention/MLP layers still work with C-dimensional inputs/outputs).

3 Hyperloop Transformers

Our architecture, shown in Figure 1, is extremely simple. We partition the Transformer into begin, middle, and end blocks, and then apply (a modification of) hyper-connections at the loop-level when we loop the middle block.

Concretely, let $\mathbf{X}_{\text{begin}} \in \mathbb{R}^{T \times C}$ be the residual stream after applying the begin block. We expand this to n parallel streams by simply copying it n times, thus giving $\mathbf{Y}^{(0)} \in \mathbb{R}^{T \times n \times C}$, which will serve as input to the hyper-connected looped block. We then compute the input-dependent matrices $\mathbf{H}_{0,t}^{\text{pre}}, \mathbf{H}_{0,t}^{\text{post}}, \mathbf{H}_{0,t}^{\text{res}} \in \mathbb{R}^{n \times n}$ for all $\{\mathbf{y}_t^{(0)}\}_{t=1}^T$ as above, but using a simpler

[4] In practice mHC uses different input-dependent matrices for attention and MLP layers.

parameterization of $\mathbf{H}_{0,t}^{\text{res}}$ given by,

$$\mathbf{H}_{0,t}^{\text{res}} = \text{diag}(\sigma(\alpha_0^{\text{res}} \cdot (\mathbf{W}_0^{\text{res}} \mathbf{z}_t^{(0)}) + \mathbf{b}_0^{\text{res}})),$$

where $\mathbf{W}_0^{\text{res}}$ is now an $n \times nC$ matrix (instead of $n^2 \times nC$) and $\mathbf{b}_0^{\text{res}} \in \mathbb{R}^n$.

We use $\{\mathbf{H}_{0,t}^{\text{pre}}\}_{t=1}^T$ on $\mathbf{Y}^{(0)}$ to obtain the C-dimensional input to the middle block, apply the middle block, and then use $\{\mathbf{H}_{0,t}^{\text{post}}\}_{t=1}^T$ to project out into the $n \times C$ residual stream. We add a "loop position embedding" $\mathbf{e}_l \in \mathbb{R}^C$ after the middle block, resulting in the recurrence,

$$\mathbf{y}_t^{(l+1)} = \mathbf{H}_{l,t}^{\text{res}} \mathbf{y}_t^{(l)} + \mathbf{H}_{l,t}^{\text{post}} \left(\mathcal{F}(\mathbf{H}_{l,t}^{\text{pre}} \mathbf{y}_t^{(l)}) + \mathbf{e}_l \right).$$

This process continues for L loops to obtain $\mathbf{Y}^{(L)}$. Finally we average $\mathbf{Y}^{(L)}$ across the parallel streams to obtain $\mathbf{X}_{\text{end}} \in \mathbb{R}^{T \times C}$, which is used as input to the end block.

Our approach differs from the original mHC in that (1) we use a simpler parameterization of $\mathbf{H}_{l,t}^{\text{res}}$ that substitutes the sinkhorn$(\cdot)$ operator over a dense matrix with a sigmoid over a diagonal matrix (which we found to be sufficient performance-wise while being more efficient), (2) we add a loop position embedding, which, when viewing the architecture as a "depth-wise RNN" with matrix-valued hidden states $\mathbf{Y}^{(0)}$, acts as the input at each time (i.e., loop) step, and (3) we only apply hyper-connections at the loop level, instead of after every attention/MLP layer (so an architecture with 3 loops would have 3 hyper-connections). Our architecture can also be seen as a more flexible parameterization of looped Transformers, which allows parameters to vary slightly across loop iterations. Concretely, we have loop-specific parameters $\{\mathbf{W}_l^\tau, \mathbf{b}_l^\tau, \alpha_l^\tau, \mathbf{e}_l\}$ for $\tau \in \{\text{pre}, \text{post}, \text{res}\}$ that can vary across loop iterations l. While the number of additional parameters here is still minimal, we posit that this parameterization allows model representations to change in a more flexible manner compared to ordinary looped Transformers which strictly enforce parameter sharing across each loop iteration.

4 Empirical Study

4.1 Experimental Setup

We train Hyperloop Transformers at various scales along with depth-matched vanilla, looped, and mHC Transformer baselines on the FineWeb-Edu dataset (Lozhkov et al., 2024). All models make use of SwiGLU MLP layers (Shazeer, 2020) and RoPE embeddings (Su et al., 2024). We use 4 parallel residual streams for both the mHC and Hyperloop Transformers. For looped models, we allocate (roughly) 25% of the available parameters to the begin block, 25% of the parameters to the end block, and the remaining 50% to the middle block, which is looped three times. This results in looped models that contain half as many parameters as their depth-matched baselines. We ablate on these choices in our ablation study.

We train models on 2.5× the Chinchilla-optimal token count of the vanilla Transformer corresponding to their size (Hoffmann et al., 2022). We use the Llama-2 tokenizer to tokenize our data and AdamW as our optimizer, with a linear warmup and cosine decay learning rate schedule. Our full hyperparameters can be found in Appendix A. These hyperparameters are generally off-the-shelf hyperparameters that have been found to work well for ordinary Transformers, i.e., we did not do any hyperparameter tuning for our architecture.

4.2 Main Results

For perplexity we evaluate our models on a held-out set consisting of 50M tokens from the FineWeb-Edu dataset. These are shown in Table 1. Our results show that while vanilla Looped Transformers can underperform depth-matched Transformer baselines, the Hyperloop Transformer only needs 150-300K extra parameters (compared to the vanilla Looped Transformer) to outperform both looped and non-looped depth-matched baseline models.

While perplexity provides a more robust measure of performance at this scale, we also evaluate our models on downstream tasks. Specifically, we evaluate our models on ARC (Clark et al., 2018), COPA (Gordon et al., 2012), HellaSwag (Zellers et al., 2019), LAMBADA

Model	Dim	Unrolled Depth	Train Tokens	Params	PPL (BF16)	PPL (INT4)	Task Acc	Training Toks/s
Transformer	1024	16	12.5B	238.0 M	14.65	14.85	41.1%	786K
mHC				241.0 M	14.55	14.73	41.1%	514K
Looped [2L → 4L (×3) → 2L]				135.5 M	14.85	15.18	41.4%	786K
Hyperloop [2L → 4L (×3) → 2L]				135.7 M	**14.40**	**14.68**	**41.6%**	750K
Transformer	2048	18	50B	990.5 M	10.19	10.27	48.0%	367K
mHC				997.5 M	10.07	10.16	48.6%	237K
Looped [3L → 4L (×3) → 3L]				579.4 M	10.02	10.24	49.2%	367K
Hyperloop [3L → 4L (×3) → 3L]				579.7 M	**9.65**	**9.81**	**49.8%**	354K
Transformer	2048	38	100B	2018.0 M	8.60	8.71	52.8%	181K
mHC				2033.0 M	8.57	8.62	53.7%	109K
Looped [4L → 10L (×3) → 4L]				990.5 M	8.68	8.97	53.3%	183K
Hyperloop [4L → 10L (×3) → 4L]				990.8 M	**8.49**	**8.59**	**54.6%**	180K

Table 1: Main results of our architecture and baselines pretrained on FineWeb-Edu. For looped models, [2L → 4L (×3) → 2L] means we have 2 begin layers, 4 middle layers looped 3 times, and 2 end layers. Perplexities are computed with both BF16 and INT4, where we use GPTQ to quantize to INT4. Task accuracies are based on BF16. Training throughput measures tokens/second and is based on eight H100s with NVLink.

(Paperno et al., 2016), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), RACE (Lai et al., 2017), SciQ (Welbl et al., 2017), and WinoGrande (Sakaguchi et al., 2019). Interestingly, we find that the looped Transformer also outperforms the vanilla Transformer on most tasks, despite using 50% fewer parameters and despite underperforming the Transformer model in perplexity terms. This outperformance corroborates similar findings reported in the literature (Saunshi et al., 2025). Hyperloop Transformer outperforms all other baselines overall. Results broken down by task can be found in Appendix B.

Post-training Quantization. Post-training quantization of a model's weights is a standard approach for reducing a model's memory footprint. While looped models are *parameter*-efficient, models that are harder to quantize would be practically *memory*-inefficient. Insofar as models trained with more tokens have been shown to be generally harder to quantize (Huang et al., 2024; Ouyang et al., 2024), it is possible that looped models would also be harder to quantize because the looped layers are trained on "more" inputs.[5] We quantize our models (originally trained in mixed precision with BF16 weights) using GPTQ (Frantar et al., 2022) to INT4, where we modify the GPTQ algorithm so that the Hessian estimation for a looped layer aggregates activations across all iterations of that looped layer. We use a calibration set of 1024 sequences from FineWeb-Edu, and use a group size of 128 for all model sizes. The perplexities of the resulting INT4 models are presented in Table 1. Our results show that while looped Transformers can indeed be somewhat more sensitive to lower-precision quantization compared to non-looped models, hyper-connections help alleviate some of the performance degradation resulting from quantization. As a result, Hyperloop Transformers continue to perform well in the weight-only quantization setting.

Training efficiency. Do the extra hyper-connections add training overhead? We measure the training throughput of each of the pretrained models on a single node with 8 × H100 GPUs with NVLink and present the results in Table 1. The models used for these measurements were implemented in PyTorch and compiled with `torch.compile` but without any further optimizations. Our results show that a straightforward PyTorch implementation of our approach only incurs minimal slowdown compared to the Transformer and Looped Transformer baselines. This can be attributed to only applying hyper-connections at the *inter-loop* level, and also using a simpler structure for $\mathbf{H}^{\text{res}}$ resulting in very little memory and compute overhead. On the other hand, a straightforward implementation of the mHC Transformer results in nontrivial slowdowns. This overhead can be brought down in theory with the proper low-level optimizations—for example, Xie et al. (2026) report a 6.7% overhead with their specialized training kernel. However, this kernel is not publicly available to the best of our knowledge, and the fact that our approach adds almost no overhead without requiring any sophisticated systems engineering is an additional benefit.

[5]To our knowledge there has been no prior work that quantizes looped models.

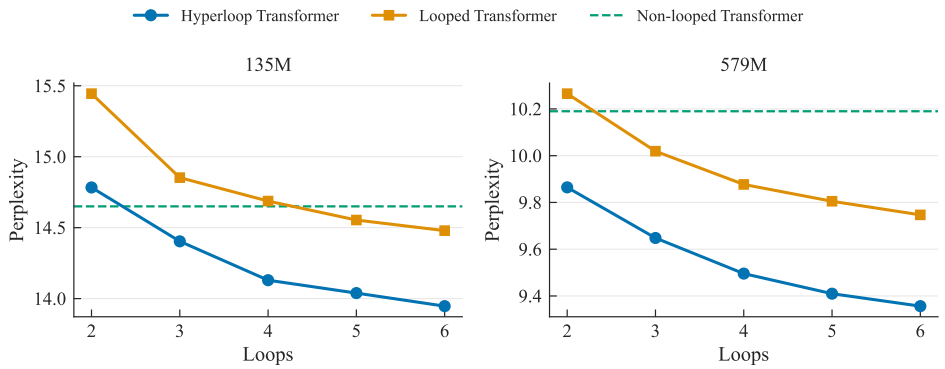

Figure 2: Perplexity numbers as the number of loops is varied for the 135M (left) and 579M (right) parameter looped models. The non-looped Transformer baselines have 238M (left) and 991M (right) parameters. Each loop consists of 4 Transformer layers.

Training for more tokens. The number of tokens in our training set is 50× the number of parameters of the non-looped baseline models, i.e., 2.5× that of the compute-optimal 20× recipe suggested by Hoffmann et al. (2022). However, modern models are typically trained for many more tokens than is Chinchilla-optimal. For example, LLaMA3-8B (Grattafiori et al., 2024) was trained on 15T tokens, while OLMo3-7B (Olmo et al., 2025) was trained on 6T tokens. Would the benefits of looped models diminish in such overtraining regimes? To investigate this, we train our smallest class of models, corresponding to 240M non-looped parameters, on 100B tokens from FineWeb-Edu. This corresponds to 20× the Chinchilla-optimal number of tokens, or approximately 400 training tokens per parameter. The results are presented in Table 2. We find that while looped Transformers underperform non-looped baselines, Hyperloop Transformers remain competitive with these baselines despite being in an overtrained setting.

Model	Params	Train Tokens 12.5B	100B
Transformer	238.0 M	14.65	12.15
mHC	241.0 M	14.55	12.16
Looped	135.5 M	14.85	12.56
Hyperloop	135.7 M	14.40	12.19

Table 2: Perplexity results for our smallest (16 layer) models trained on more tokens.

4.3 Ablations

Number of loops. Our main experiments use 3 loops. How does performance vary as a function of the number of loops, if we hold the parameters constant? We vary the number of loops from 2 to 6 for the 136M- and 579M-parameter looped and Hyperloop Transformers, and show the results in Figure 2. We observe diminishing returns as we increase the number of loops, although Hyperloop Transformer dominates the Looped Transformer in all cases.

The above experiments hold the parameter count constant and vary the depth by changing the number of loops. We next experiment with restructuring the middle loop by making it smaller but looping it more to keep the depth constant. The results are shown in Table 3. We find that we can obtain even greater parameter-efficiency at the cost of a small performance hit. For example, our 477M parameter model still outperforms the full 1B Transformer.

We now conduct more ablations focusing on the 136M-parameter/12.5B-token setting.

Number of parallel streams. We pick $n = 4$ parallel residual streams as recommended by the original mHC work. Insofar as the number of parallel streams provides a parameter-efficient axis with which to scale up the model, can we get further gains by increasing n? Table 4 shows the results on a 135M-parameter Hyperloop Transformer with 3 loops and a varying number of parallel residual streams, where we observe diminishing returns on n. Thus, while having matrix-valued residual streams does improve performance, this axis of scaling rapidly faces diminishing returns.

Number of Streams n	PPL
2	14.43
4	14.40
6	14.38
8	14.39
10	14.35

Table 4: Hyperloop Transformer performance as we vary the number of parallel residual streams.

Model	Structure	Unrolled Depth	Params	PPL
Looped Transformer	$2L \to 4L\ (\times 3) \to 2L$	16 layers	136 M	14.85
	$2L \to 3L\ (\times 4) \to 2L$		123 M	15.18
	$2L \to 2L\ (\times 6) \to 2L$		110 M	15.76
Hyperloop Transformer	$2L \to 4L\ (\times 3) \to 2L$	16 layers	136 M	14.40
	$2L \to 3L\ (\times 4) \to 2L$		123 M	14.62
	$2L \to 2L\ (\times 6) \to 2L$		110 M	15.06
Looped Transformer	$3L \to 4L\ (\times 3) \to 3L$	18 layers	579 M	10.02
	$3L \to 3L\ (\times 4) \to 3L$		528 M	10.12
	$3L \to 2L\ (\times 6) \to 3L$		477 M	10.36
Hyperloop Transformer	$3L \to 4L\ (\times 3) \to 2L$	18 layers	579 M	9.65
	$3L \to 3L\ (\times 4) \to 3L$		528 M	9.72
	$3L \to 2L\ (\times 6) \to 3L$		477 M	9.86

Table 3: Performance of the looped and Hyperloop Transformers as we vary the looping structure while keeping the depth fixed at 16 or 18 layers.

Number of hyper-connections. Recall that our Hyperloop Transformer only applies hyper-connections after each loop, which results in minimal additional parameter/compute overhead. We ablate on the number of hyper-connections used within the looped block of a 135M-parameter Hyperloop Transformer. To do so, we fix the number of loops and only vary the number of hyper-connections, or equivalently, the number of Transformer blocks skipped over by a single hyper-connection. This results in some hyper-connections being applied within loops or across Transformer blocks from different loops in some setups. The sub-layers within a Transformer block retain their own original skip-connections, even when hyper-connections are applied after every block. Table 5 shows that applying hyper-connections after every loop (instead of every layer) is the most performant setup. This is perhaps unintuitive, given that the every-layer setup uses the most compute/parameters. Our results potentially indicate that at least in the looped case, one must be more careful about choosing where to apply hyper-connections.

Hyper connections	PPL
12 [every layer]	14.45
6 [every 2 layers]	14.50
4 [every 3 layers]	14.50
3 [every loop (ours)]	14.40
2 [every 6 layers]	14.50
1 [every 12 layers]	14.63

Table 5: Performance as we vary the number of HCs.

$\mathbf{H}^{\text{res}}$ parameterization. The Hyperloop Transformer simplifies the mHC formulation by using a simpler diagonal transition matrix $\mathbf{H}^{\text{res}}$ for the parallel residual stream, in contrast to the doubly-stochastic structure used in mHC. This leads to fewer parameters and less compute. Does this potentially hurt the performance of our approach? As shown in Table 6, we find that this is not the case. The simplest identity parameterization only slightly underperforms the Sinkhorn parameterization, while our (data-dependent) diagonal parameterization further improves performance.

Parameterization	PPL
Identity	14.61
Sinkhorn	14.59
Diagonal [ours]	14.40

Table 6: Ablations on parameterization of the transition matrix $\mathbf{H}^{\text{res}}$.

Comparison vs. LoRA-Looped Transformers. Since the weights for the hyper-connections are different across loops, our Hyperloop Transformer allows for the looped block to be slightly different across loops. Would using LoRA to modify the parameters across loops (as in relaxed recursive Transformers; Bae et al., 2024) perform better? The results of these experiments are shown in Table 7. We find that allowing parameters to change across loops with LoRA does help slightly, but the Hyperloop Transformer provides a much more parameter-efficient approach to improving performance.

LoRA Rank	Params	PPL
0 [Looped Transformer]	135.5M	14.85
4	136.0M	14.85
8	137.0M	14.81
16	139.0M	14.80
32	143.0M	14.77
Transformer	238.0M	14.65
Hyperloop	136.0M	14.40

Table 7: Experiments on allowing Transformer layers to change across loop iterations with LoRA.

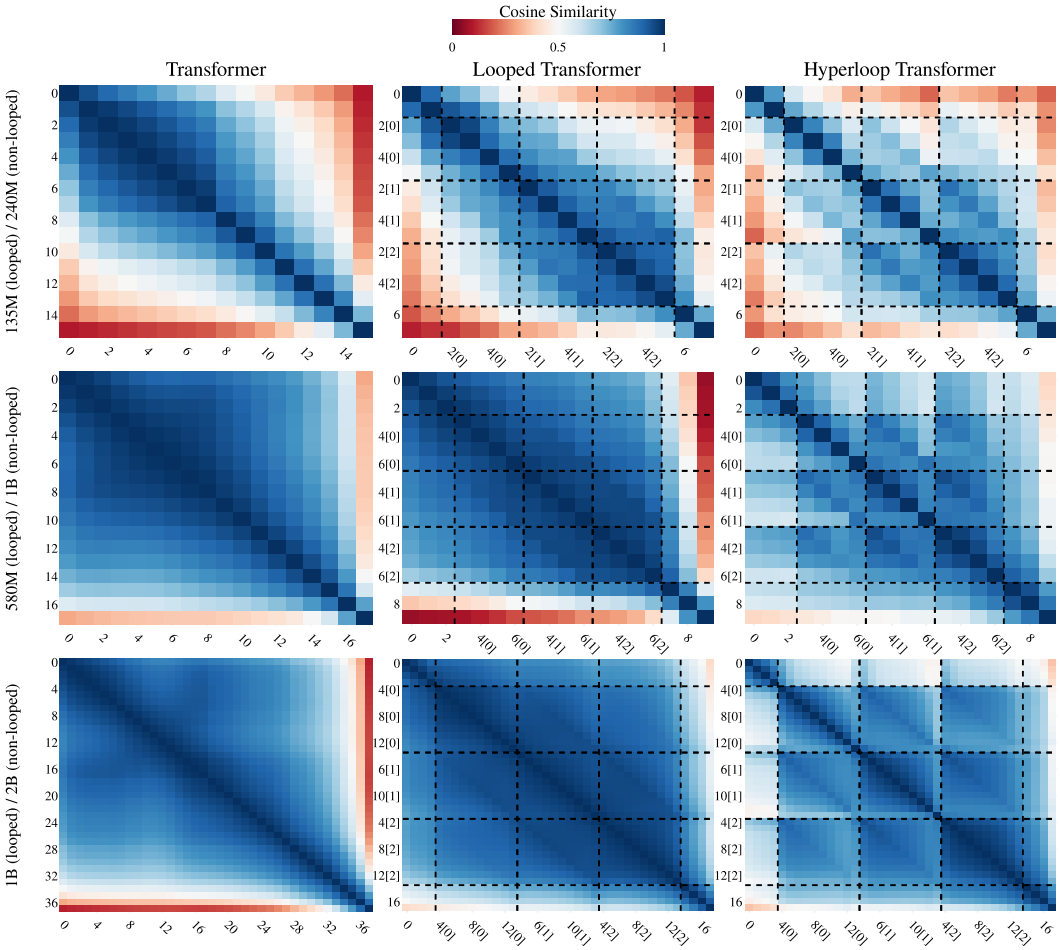

Figure 3: Pairwise cosine similarity between inner residual streams at each (effective) layer, across model scales (rows) and architectures (columns).

4.4 Analysis

To better understand our model's inner workings, we conduct a series of qualitative analyses of its internal representations on 50M tokens from the FineWeb-Edu dataset.

Representation similarity. We hypothesize that the outperformance of the Hyperloop architecture is supported in part by hyper-connections allowing for the model representations to be less constrained than in the ordinary looped case. To investigate this, we analyze the cosine similarity of the residual stream as we vary the depth in Figure 3. We see that both looped models exhibit similarity within the looped blocks; in particular, we also see that the representations output by the same layer *across* loops exhibit higher-than-expected similarity. Table 8 quantifies the average similarity of layers across loops (e.g., comparing representations of middle layer 1 across loops) for all the looped layers. We find that Hyperloop models' representations are indeed less similar, supporting our hypothesis.

Params	Looped	Hyperloop
136M	0.743	0.738
579M	0.915	0.872
991M	0.923	0.871

Table 8: Average cosine similarity between corresponding layers across loop iterations.

Logit lens. We also perform a logit lens-style analysis (nostalgebraist, 2020). We observe that the "outer" residual streams (i.e., the parallel streams in the mHC/Hyperloop Transformers; the regular stream in the other Transformers) are loosely aligned to the vocabulary

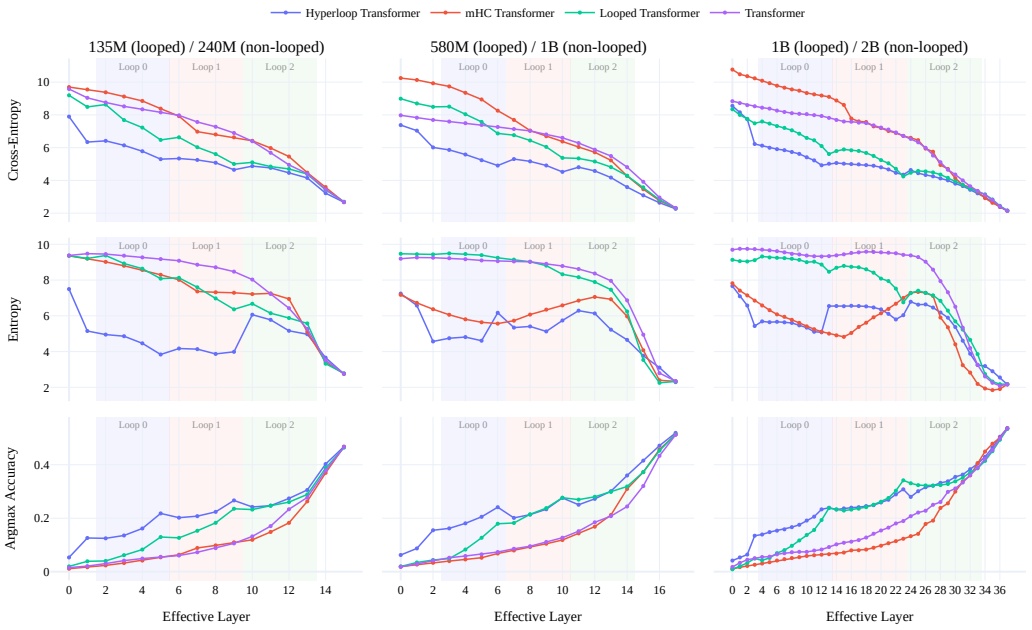

Figure 4: Logit lens-inspired analysis across model scales. Each column corresponds to a model scale, and each row shows a different metric: average cross-entropy (top), average entropy of vocabulary distribution (middle), and greedy decoding accuracy (bottom), computed by mapping the outer residual stream via the language modeling head. Loop boundaries are indicated at the top of each panel, though they only apply to looped models.

space.[6] We are thus able to push these representations through the language modeling head to get a distribution over the next token. From this, we can compute the evolution of the cross-entropy, entropy, and the accuracy of the argmax of the distribution, as shown in Figure 4. We see that both Hyperloop and the vanilla looped transformer produce representations that are more aligned with the vocabulary distribution, likely as the looping forces the models to operate closer to the vocabulary space. The fact that both looped models exhibit maximum alignment toward the vocabulary distribution at the end of a loop further corroborates this claim. Interestingly, our approach produces models with *higher* alignment than the vanilla looped models, suggesting that the hyper-connections offer additional regularization in this direction. This potentially indicates that Hyperloop Transformers could be more amenable to early-exit-style inference strategies and enable compute savings.

5 Discussion

This work shows that combining recent hyper-connections (Xie et al., 2026) with looped Transformers can push the parameter-performance frontier of language models. We study several methods for incorporating hyper-connections, and show that doing so at the loop-level with a simple data-dependent diagonal transition matrix is effective, while incurring minimal additional parameters/compute. We find suggestive evidence that the outperformance of our approach is supported in part by the hyper-connections allowing the model representations of looped layers to deviate more flexibly. While we have primarily focused on parameter-efficiency controlling for compute, our logit lens analysis further suggests that Hyperloop Transformers could enable compute efficiency gains through early-exit style inference strategies.

Our main limitation is scale. While we performed experiments that were reasonable on academic compute, it is unclear as to whether the overall efficiency gain (i.e., Hyperloop Transformers matching Transformers with 50% fewer parameters) would hold at even larger

[6] For the Hyperloop model, we can compute the effective value of the parallel residual streams by performing an early merge operation from the intra-loop residual stream to the parallel stream.

scales, although we did find that Hyperloop Transformers were effective in the overtrained regime for the smaller models. Insofar as increasing the number of loops has been suggested as a new axis of scaling (Geiping et al., 2025; Prairie et al., 2026), it would be interesting to train much deeper Hyperloop Transformers (with the hyper-connection parameters shared across loops to enable generalization to longer loops than seen in training) to see if it can enable effective test-time scaling.

6 Related Work

Our work adds to a growing body of literature investigating the capabilities of looped transformers and augmented residual connection patterns in language modeling.

Looped Transformers. Looped Transformers were first proposed by Dehghani et al. (2018) and applied to BERT-style models in ALBERT (Lan et al., 2019). Modern variants of looped Transformers were initially studied from a theoretical expressivity perspective (Giannou et al., 2023; Yang et al., 2023; Xu & Sato, 2024). However, more recent works have shown the empirical effectiveness of looped models on real language modeling and synthetic reasoning tasks. Saunshi et al. (2025) study the performance of looped architectures on synthetic reasoning and downstream language modeling tasks and find that looped models perform better on certain kinds of reasoning tasks. Kohli et al. (2026) study synthetic multi-hop reasoning with looped models and find that they can generalize to more hops than seen in training. Bae et al. (2025) propose a looped architecture that allocates a dynamic number of loops on a per-token basis. Geiping et al. (2025) train a looped model on a variable number of loops sampled from a log-normal Poisson distribution and show that the model's performance on downstream tasks scales with the number of loops at test-time. Zhu et al. (2025b) train looped language models through all stages of a modern language modeling pipeline and propose an entropy-regularized objective for early exiting after a dynamic number of loops. Prairie et al. (2026) develop a more stable parameterization of looped Transformers and derive scaling laws. Bae et al. (2024) and McLeish et al. (2025) investigate the feasibility of converting existing models to a looped architecture. Finally, Blayney et al. (2026) perform a mechanistic analysis of looped Transformers and find that the latent states follow a cyclic trajectory.

Residual connections in Transformers. Our work is also related to approaches that modify the residual stream connection patterns in Transformers. Zhu et al. (2025a) and Xie et al. (2026) expand the residual stream into a residual matrix, allowing richer connections from earlier to later layers in the model. Pagliardini et al. (2024) expand the residual stream differently, averaging the hidden states at the output of a transformer block with earlier hidden states using different sparsity patterns. Xiao et al. (2025), Heddes et al. (2025), and Team et al. (2026) take that a step further, allowing the model to attend to previous hidden states along the depth axis. We leave the integration of looped Transformers with other residual connection patterns to future work.

7 Conclusion

We propose a simple architecture that combines hyper-connections with looped Transformers and improves the parameter-efficiency of language models while adding almost no additional compute at training and deployment.

Acknowledgments

We thank Junhyun Lee, Munjo Kim, and Oliver Sieberling for helpful discussions. This study was supported by a Samsung Research grant and the AI2050 program at Schmidt Sciences (Grant G-25-67980).

References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. *arXiv preprint arXiv:2410.20672*, 2024.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Michael M. Bronstein andXiaowen Dong. A mechanistic analysis of looped reasoning language models. *arXiv preprint arXiv:2604.11791*, 2026.

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

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

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.

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

Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. SemEval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret (eds.), **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 394–398, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics. URL https://aclanthology.org/S12-1052/.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Mike Heddes, Adel Javanmard, Kyriakos Axiotis, Gang Fu, MohammadHossein Bateni, and Vahab Mirrokni. Deepcrossattention: Supercharging transformer residual connections. *arXiv preprint arXiv:2502.06785*, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Wei Huang, Xingyu Zheng, Xudong Ma, Haotong Qin, Chengtao Lv, Hong Chen, Jie Luo, Xiaojuan Qi, Xianglong Liu, and Michele Magno. An empirical study of llama3 quantization: From llms to mllms. *Visual Intelligence*, 2(1):36, 2024.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. *arXiv preprint arXiv:2604.07822*, 2026.

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. *arXiv preprint arXiv:1909.11942*, 2019.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh 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. URL https://arxiv.org/abs/2511.07384.

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 *EMNLP*, 2018.

nostalgebraist. interpreting GPT: the logit lens, August 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. LessWrong blog post.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Xu Ouyang, Tao Ge, Thomas Hartvigsen, Zhisong Zhang, Haitao Mi, and Dong Yu. Low-bit quantization favors undertrained llms: Scaling laws for quantized llms with 100t training tokens. *arXiv preprint arXiv:2411.17691*, 2024.

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging, 2024. URL https://arxiv.org/abs/2402.02622.

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/.

Francesco Pappone, Donato Crisostomi, and Emanuele Rodolà. Two-scale latent dynamics for recurrent-depth transformers. *arXiv preprint arXiv:2509.23314*, 2025.

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.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

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

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

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

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 169–179, 2023.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026. URL https://arxiv.org/abs/2603.15031.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *NUT@EMNLP*, 2017.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. Muddformer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections, 2025. URL https://arxiv.org/abs/2502.12170.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mhc: Manifold-constrained hyper-connections, 2026. URL https://arxiv.org/abs/2512.24880.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. *arXiv preprint arXiv:2410.01405*, 2024.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *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/.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=9FqARW7dwB.

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*, 2025b.

Model Parameters	Task	Transformer	mHC	Looped	Hyperloop
240M (non-looped) / 136M (looped)	ARC-Challenge	19.45%	**21.25%**	19.71%	20.56%
	ARC-Easy	49.24%	49.79%	49.45%	**50.63%**
	COPA	62.00%	60.00%	62.00%	**63.00%**
	HellaSwag⋆	31.96%	31.87%	31.37%	**32.00%**
	LAMBADA (OpenAI)	24.14%	24.76%	**25.17%**	25.09%
	LAMBADA (Standard)	17.95%	17.66%	**18.03%**	17.89%
	OpenBookQA⋆	30.60%	**31.40%**	30.80%	31.20%
	PIQA⋆	61.53%	61.37%	60.94%	**63.33%**
	RACE	29.19%	26.32%	**29.86%**	28.71%
	SciQ	76.60%	**77.00%**	74.50%	75.20%
	WinoGrande	49.88%	50.12%	**53.04%**	50.20%
	Average	*41.14%*	*41.05%*	*41.35%*	***41.62%***
1B (non-looped) / 579M (looped)	ARC-Challenge	25.68%	27.90%	27.73%	**28.07%**
	ARC-Easy	59.30%	60.14%	**62.46%**	62.08%
	COPA	70.00%	71.00%	**73.00%**	68.00%
	HellaSwag⋆	42.07%	42.80%	43.98%	**46.22%**
	LAMBADA (OpenAI)	36.64%	36.11%	37.16%	**39.05%**
	LAMBADA (Standard)	28.20%	27.94%	28.18%	**30.62%**
	OpenBookQA⋆	**34.40%**	33.40%	33.40%	33.80%
	PIQA⋆	66.43%	67.08%	67.14%	**68.72%**
	RACE	30.53%	31.10%	**31.77%**	**31.77%**
	SciQ	82.80%	83.90%	84.70%	**86.30%**
	WinoGrande	52.17%	52.96%	51.62%	**53.04%**
	Average	*48.02%*	*48.58%*	*49.19%*	***49.79%***
2B (non-looped) / 1B (looped)	ARC-Challenge	31.66%	31.74%	32.17%	**33.70%**
	ARC-Easy	68.18%	66.96%	68.60%	**69.02%**
	COPA	70.00%	**74.00%**	72.00%	**74.00%**
	HellaSwag⋆	51.44%	51.87%	52.21%	**53.93%**
	LAMBADA (OpenAI)	42.89%	43.94%	43.37%	**45.55%**
	LAMBADA (Standard)	35.16%	35.44%	34.06%	**37.96%**
	OpenBookQA⋆	**37.80%**	37.40%	36.20%	36.20%
	PIQA⋆	**71.49%**	70.89%	71.33%	71.27%
	RACE	32.54%	**34.64%**	32.34%	33.59%
	SciQ	85.50%	87.60%	**88.50%**	88.20%
	WinoGrande	53.83%	56.35%	55.41%	**57.06%**
	Average	*52.77%*	*53.71%*	*53.29%*	***54.59%***

Table 9: Accuracies on downstream tasks across three model-size settings. For tasks marked with a ⋆, we normalize the log-likelihood of the different multiple-choice continuations by the number of tokens.

A Hyperparameters

We use a model dimension of 1024 for our 240M/136M-parameter models and a model dimension of 2048 for our larger models. We set our SwiGLU feed-forward dimension to be 2.75× the model dimension across all model sizes. All model sizes use Multi-Head Attention with 16 attention heads and a RoPE base of 10000 for their position embeddings. We use a weight-untied unembedding matrix and include its number of parameters in our reported model sizes.

Our models are trained on batches of 256 sequences of length 2048, corresponding to 524K tokens per batch. Across all training runs, we use a max learning rate of 4×10^{-4}, with cosine decay down to 4×10^{-5}. We use 1000 warmup steps for our 240M/136M models and 2000 warmup steps for the larger models. For AdamW, we use $(\beta_1, \beta_2) = (0.9, 0.95)$ and a weight decay of 0.1. We use gradient clipping for gradient norms above 1.0.

B Downstream Task Evaluations

Table 9 shows the downstream task results broken down by tasks.