Sparse Layers are Critical to Scaling Looped Language Models

Ryan Lee[1*] **Jacob Biloki**[2] **Edward J. Hu**[3] **Jonathan May**[1]

[1]USC Information Sciences Institute
[2]Netflix
[3]Independent Researcher

Abstract

Looped language models repeat a set of transformer layers through depth, reducing memory costs and providing natural early-exit points at loop boundaries. However, looped models do not scale as favorably as standard transformers with unique layers. We compare standard and Mixture-of-Experts (MoE) transformers, with and without looping, and find two main results. First, we find Looped-MoE models scale better than the standard baseline while dense looped models do not. We trace this to routing divergence between loops: in Looped-MoE models, different experts are activated on each pass through the same shared layers, recovering expressivity without additional parameters. Our second finding is that looped models have better compute-quality trade-offs with early exits than standard models. Because each loop ends with the same layers that produce the final output, loop boundaries are superior exit points, as confirmed by earlier output convergence at these points. In sum, we provide a clear direction for scaling looped models: a Looped-MoE model with early exits can not only beat standard transformers at scale, but also enable significant memory and inference savings with minimal degradation in quality.

1 Introduction

Language models (LMs) are expensive to store and slow during inference due to their size. This stems from a fundamental architectural limitation: LMs can consist of billions of parameters arranged as a sequential stack of computational layers, yet each parameter is only used once during an inference step. Re-using parameters through compute depth, as explored in universal transformers [1], LoopLMs [2], and recurrent models [3], is an appealing alternative, however such architectures have been found to under-perform baselines when compared on equal training compute [4].

The most common pattern in parameter re-use is to loop transformer layers through depth. We hypothesize the expressiveness bottleneck for looped LMs is the dense feed-forward network (FFN), which applies the same operation to every token, an invariance that compounds when looped. Sparse Mixture-of-Experts (MoE) [5] layers offer a natural remedy by routing tokens to specialized experts, introducing input-dependent computation that may recover expressiveness lost to weight tying. MoE has been combined with looping before [6], but alongside other architectural changes.

In this work, we isolate the effect of sparse experts on looped scaling laws with a controlled study. We find that **(i) sparse experts resolve the looped scaling deficit** and achieve the best downstream accuracy. Although looping reduces storage, it does not on its own address the inference cost of deep transformers. Thus we explore early exiting [7–9], allowing tokens to exit before full depth, and find that **(ii) looped models have better compute-quality trade-offs** than standard transformers using training-free early exit. This property directly translates to inference savings: looped transformers can skip more computation for the same acceptable drop in performance.

*Corresponding author. ryantlee@usc.edu

Preprint.

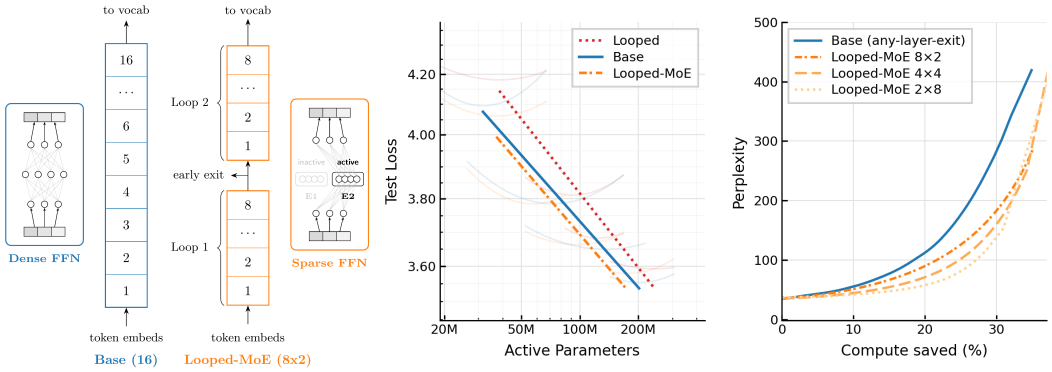


Figure 1: Overview of our models and main results. **Left:** Base and Looped-MoE. Base models have unique layers and dense FFNs, while Looped-MoE models have repeated layer stacks with sparse MoE layers. **Middle:** IsoFLOP curves. Given the same number of parameters, Looped-MoE (8×2) achieves lower test loss than Base across compute budgets. **Right:** Early-exit Pareto. Looped-MoE has a better compute-quality trade-off than Base models, and this improves with more looping.

We identify two key properties of sparse looped models which enable superior scaling and early exit trade-offs: **(i) Loop-specific expert routing.** In Looped-MoE models, the router selects a different set of experts for the same physical layer on each loop pass, enabling shared layers to implement distinct computations across iterations. **(ii) Earlier convergence at loop boundaries.** In looped models, more tokens reach near-final predictions after the first loop layers, meaning early exits are more accurate than in non-looped models.

Practical Recommendation. Replace dense FFNs with top-k MoE layers in looped LMs. The resulting Looped-MoE architecture scales better than a standard dense LM, stores fewer weights for the same accuracy, and enables more compute savings via early exits at loop boundaries.

2 Model Descriptions

The architectures compared in this work share a common backbone: a decoder-only transformer with multi-head self-attention (MHSA) using rotary positional embeddings (RoPE) [10], SwiGLU feed-forward networks [11], pre-RMSNorm [12], and a residual stream (Figure 8). To explore the impact of sparse layers on looped scaling laws, we vary model architecture by whether the FFN is dense or sparse and whether the layers are looped (Table 1).

2.1 Layer Looping

As illustrated in Figure 1 (left), in a looped transformer, a stack of L unique transformer layers is repeated R times in sequence during the forward pass, producing an effective depth of $L \times R$. The looped variants in the scaling study use L=8, R=2, matching the 16-layer effective depth of their non-looped counterparts. In Section 6.3, we explore the impact of more looping with fewer layers.

Table 1: Scaling Study Configurations.

Architecture	FFN	Layers
Looped	Dense	8×2
Base	Dense	16
Looped-MoE	Sparse	8×2
MoE	Sparse	16

2.2 Sparse Mixture-of-Experts

In MoE, the SwiGLU FFN [11] with feed-forward dimension d_{ff} is replaced with a set of experts (smaller SwiGLU FFNs). To select which expert FFNs to use per token, we use top-k token-choice routing [5]: a learned router $h(x) = W_{\text{router}}x$ assigns each token embedding x to its top-k experts. Defining $W_{\text{router}} \in \mathbb{R}^{d_{\text{model}} \times E}$, where E is the total number of experts, the forward pass from x to y is:

$$\mathcal{T} = \text{top-}k(h(x)), \quad p_i(x) = \frac{e^{h(x)_i}}{\sum_{j \in \mathcal{T}} e^{h(x)_j}}, \quad y = \sum_{i \in \mathcal{T}} p_i(x)\,\text{FFN}_i(x)$$

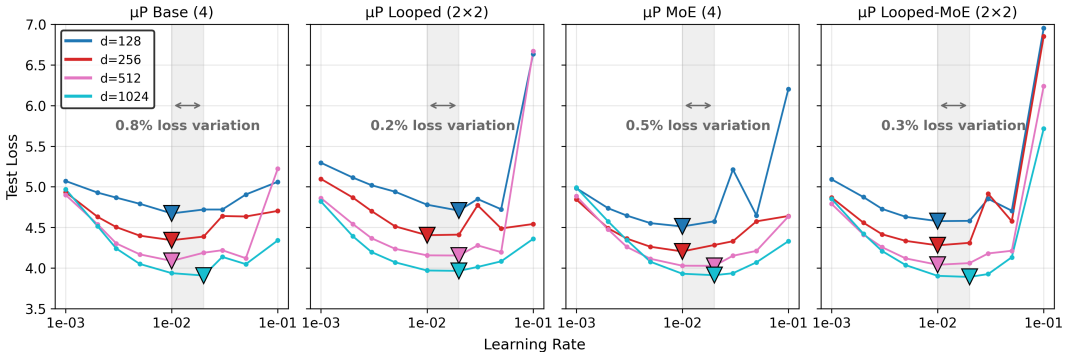


Figure 2: μP Transfer test. The best learning rate for the smallest model remains optimal across larger sizes, validating our μP implementation. If the optimal learning rate for a given width does differ from the base learning rate, the loss difference is < 1%.

Table 2: μP scaling rules for all weights except the embedding ($w_{\text{ratio}} = d_{\text{model}}/d_{\text{base}}$). We extend these to router, expert, and unembedding weights, removing the original input/output multipliers.

Init. Variance	**Learning Rate**	**New for MoE, Looped**
$\sigma^2_{\text{base}}/w_{\text{ratio}}$	$\eta_{\text{base}}/w_{\text{ratio}}$	$W_{\text{unembed}}, W_{\text{router}}, W_{\text{experts}}$

To prevent expert collapse, we apply two standard MoE auxiliary losses: a load balancing loss [13], which encourages uniform token assignment across experts, and a router z-loss [14], which penalizes large logits to stabilize training:

$$\mathcal{L}_{\text{LB}} = E \sum_{i=1}^{E} f_i \cdot \overline{p}_i, \qquad \mathcal{L}_{\text{RZ}} = \frac{1}{B} \sum_{j=1}^{B} \left(\log \sum_{i=1}^{E} \exp(h(x_j)_i) \right)^2$$

where f_i is the fraction of tokens routed to expert i, $\overline{p}_i$ is its mean routing probability, $h(x_j)_i$ is the router logit for expert i on token j, and B is the number of tokens in the batch. For all MoE configurations in this study, we use k=2 active experts out of E=8 total, following Mixtral [15].

3 Experimental Set-up

3.1 Maximal Update Parameterization for Looped and Sparse Layers

We use Maximal Update Parameterization (μP) [16] to reduce the cost of our scaling studies. With μP, the optimal learning rate tuned at a small base width (we use $d_{\text{base}} = 128$) transfers to larger widths. This is efficient for isoFLOP studies, which require training models across a range of sizes: with μP, it is not necessary to re-tune the learning rate at every new width.

The core idea is to scale initialization variance and learning rates inversely with a width ratio $w_{\text{ratio}} = d_{\text{model}}/d_{\text{base}}$, so that activation and update magnitudes remain consistent across model scales. We apply standard μP to all non-embedding weight matrices (Table 2) then extend these scalings to the unembedding matrix W_{unembed} in place of the output multiplier used in the original formulation.

We treat MoE expert weights and the router as additional hidden weights and apply width-scaled initialization and learning rates. Unlike prior approaches [17], we find it sufficient to only use two scaling mechanisms, initialization variance and per-layer learning rate (without additional forward-pass multipliers). For looped layers, no additional modification is needed: weight-tied layers receive gradient contributions from multiple positions but their scale is unchanged.

We validate our parameterization with a μP transfer test on 4-layer effective depth models. We find that the best learning rate at the smallest width is also near-optimal for the larger widths ($d_{\text{model}} \in \{128, 256, 512, 1024\}$), with at most 0.8% loss difference across all architectures (Figure 2).

Table 3: Active vs. stored non-embedding parameters. In this table we compare the number of parameters used in the forward pass (N_{active}) against the number of parameters stored (N_{unique}), accounting for inactive expert weights in MoE. L is effective layers (counting repeated layers), A is attention parameters per layer, F is FFN parameters per layer, and $F_{\text{expert}} = F/k$ is parameters per expert for k active experts. In our total N_{active}, we also include embedding parameters (not shown).

Architecture	N_{active}	N_{unique}
Looped (R loops)	$L(A+F)$	$> \frac{L}{R}(A+F)$
Base	$L(A+F)$	$= L(A+F)$
Looped-MoE (R loops)	$L(A+F)$	$< \frac{L}{R}(A + E \cdot F_{\text{expert}})$
MoE (E experts, k active)	$L(A+F)$	$\ll L(A + E \cdot F_{\text{expert}})$

3.2 FLOPs Accounting for Looped and Sparse Layers

For calculating compute, we follow the $C = 6ND$ approximation [4, 18], where C is a compute budget in FLOPs, N is the number of total parameters, and D is data in tokens. Here, we use N_{active}, not N_{unique} as N in the equation. Concretely, we define N_{active} as the total parameters *used* in a single forward pass, counting repeated (looped) layers at each invocation and counting only the k active experts in MoE layers. This is distinct from N_{unique}, the number of stored parameters as described in Table 3. By design, all four architectures have the same N_{active} parameters (neglecting the small router weights) and therefore the same compute budget at matched token counts. We include embedding and unembedding parameters in our N_{active} count for compute calculations, an additional $(2 \cdot V \cdot d_{\text{model}})$ parameters, where V is vocabulary size.

4 Experiments

4.1 IsoFLOP Scaling Study

We find scaling laws [18] for each of our four architectures to compare them. Given a fixed compute budget C, we train models at several widths (Table 5). For each width, the number of training tokens is determined by $D = C/(6N_{\text{active}})$, so that all runs at a given budget use the same total compute. Each run produces a test loss at its (N, D) configuration; we fit a quadratic to these losses and take the minimum as the compute-optimal model for that budget. Repeating this across four budgets $C \in \{5 \times 10^{16},\ 2 \times 10^{17},\ 5 \times 10^{17},\ 10^{18}\}$ FLOPs yields a set of compute-optimal points, through which we fit a power law for loss vs. parameters, $L \propto N^{-\alpha}$.

We pretrain on the 10B token sample of FineWeb-Edu [19] using the GPT-2 tokenizer ($V = 50{,}257$), with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, independent weight decay 1.0×10^{-4}). We use a Warmup-Stable-Decay (WSD) learning rate schedule [20] with sqrt-decay cooldown [21] to 5% of peak over the final 10% of training steps. The peak learning rate is determined via μTransfer from the $d_{\text{base}} = 128$ proxy model. We provide additional hardware and train time details in Appendix A.

To test whether our hypothesis, that sparse routing restores the expressiveness lost to weight tying, holds beyond test loss, we evaluate the compute-optimal Base, Looped, and Looped-MoE models at our largest compute budget on the AI2 OLMES benchmark suite [22].

4.2 Early Exit Evaluation

To understand how looped models can save inference time in addition to memory, we evaluate the theoretical compute savings of early exit using training-free criteria. We measure the compute-quality tradeoff using entropy of the output distribution [7, 8] as the exit criterion. As shown in Figure 4 (left), at each candidate exit point, we project the hidden state to the vocabulary and exit if the entropy falls below a threshold τ. If tokens exit early, we count the unused depth as FLOPs saved.

For looped models, exit is permitted only at loop boundaries after each complete pass through the unique layer stack. For non-looped models, exit is permitted at any intermediate layer. We sweep τ to trace a Pareto frontier of compute saved (as a percentage of the full forward pass) versus perplexity. We do not measure end-to-end inference throughput. All early exit experiments use compute-optimal models at 10^{18} training FLOPs.

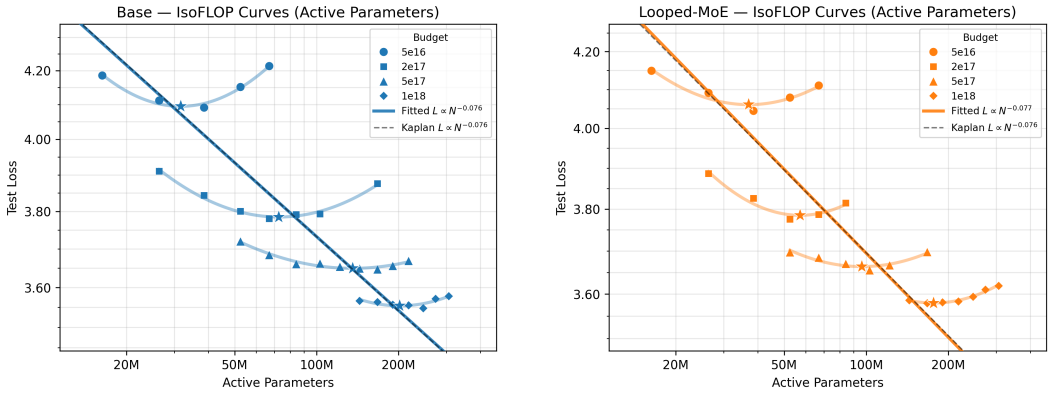


Figure 3: **Left:** IsoFLOP curves for Base. **Right:** IsoFLOP curves for Looped-MoE. Stars mark compute-optimal model sizes at each budget; solid lines show fitted $L \propto N^{-\alpha}$ ($\alpha = 0.076$ for Base, 0.077 for Looped-MoE). The dashed line shows Kaplan et al. [4] scaling exponent ($\alpha = 0.076$).

Table 4: AI2 OLMES benchmark results for compute-optimal models at 10^{18} FLOPs. Looped-MoE achieves the highest average score with fewer stored parameters than Base.

Model	Params	ARC-E	ARC-C	BoolQ	CSQA	HellaSwag	OBQA	PIQA	SIQA	WinoG	Core 9
Looped	168M	39.1	24.6	49.6	27.1	25.6	24.4	57.1	38.2	51.2	37.4
Base	246M	39.3	**25.3**	51.4	29.3	**26.2**	**27.6**	**57.8**	**40.5**	50.5	38.7
Looped-MoE	216M	**40.4**	24.3	**63.9**	**30.9**	25.4	26.8	55.2	38.0	**51.7**	**39.6**

5 Results

5.1 IsoFLOP Scaling Laws

We find that Looped-MoE scales better than Base, while Looped models scale strictly worse (Figure 1, middle), supporting our hypothesis that sparse routing restores expressiveness lost to weight tying. Looping alone is detrimental, with MoE scaling better than Looped-MoE and Base better than Looped (Figure 10), consistent with prior work [4, 23]. Yet looped models remain appealing for their memory savings and early-exit points, and we show that sparse layers close the scaling gap. Figure 3 shows the individual isoFLOP curves for Base and Looped-MoE; fits for Looped and MoE are in Figure 9. Fitting a power law for Loss, $L \propto N^{-\alpha}$ through the compute-optimal points yields $\alpha = 0.076$ (Base) and 0.077 (Looped-MoE), consistent with the 0.076 scaling exponent reported by Kaplan et al. [4].

5.2 Downstream Evaluation

The downstream results also confirm finding (i): sparse experts resolve the looped scaling deficit. Table 4 reports AI2 OLMES benchmark results for the compute-optimal Base, Looped, and Looped-MoE models at the 10^{18} FLOP budget. Looped-MoE achieves the highest average score (39.6) across the Core 9 benchmarks, outperforming Base (38.7) despite storing fewer unique parameters (216M vs. 246M). Looped scores lowest (37.4), consistent with its scaling deficit in test loss. Full benchmark results including MoE are reported in Table 6.

5.3 Early Exit

Figure 4 shows compute-quality trade-off under a training-free entropy criterion; Table 7 reports it in tabular format. Looped models degrade less than their non-looped counterparts: at 10% FLOPs saved, Looped and Looped-MoE reach perplexity 50.2 and 51.0, compared to 55.4 for Base and 75.7 for MoE. Notably, MoE degrades fastest, suggesting that sparse routing is not the primary contributor to favorable early-exit trade-offs. We hypothesize the benefit comes from looping itself: the last layers that produce the final output are also the last layers in each loop, and investigate this in Section 6.2.

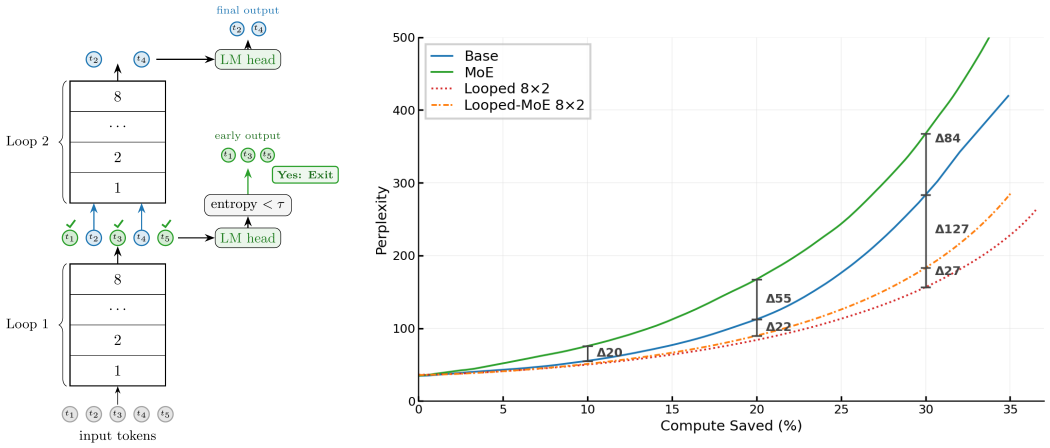


Figure 4: **Left:** Schematic of entropy-based early exit. For Looped/Looped-MoE, tokens exit at loop boundaries. **Right:** Looped/Looped-MoE models have the best compute-quality trade-offs.

6 Analysis

In this section, we conduct experiments to understand why replacing the dense FFN of a looped transformer with a MoE layer results in better scaling laws and early-exit trade-offs. At a high level we find the reasons are: (1) loop-specific expert assignments (Sec 6.1) and (2) earlier convergence to final activations at loop points, due to the final layers at the end of loops being the same layers trained to produce the final activations (Sec 6.2). For Looped-MoE, we find that more looping further improves early-exit trade-offs (Sec 6.3); we leave the impact on scaling for future work. All experiments use models at 10^{18} training FLOPs, evaluated on 800K test tokens.

6.1 Why do MoE Layers Improve Looped Scaling Laws?

Our scaling laws and downstream evaluations reveal that Looped-MoE architectures consistently outperform dense baselines at equivalent compute budgets. We hypothesize that MoE routing overcomes the expressive limitation of weight tying by activating different expert sub-networks on each loop pass. If we are correct, when the same token passes through the same physical layer on the second loop iteration, the router should assign it to different experts. If expert assignments diverge across loops, then MoE layers effectively specialize by loop iteration and the same layers have depth-unique computations.

Setup. For the compute-optimal Looped-MoE model at 10^{18} training FLOPs, we track the top-k expert assignments on both loop passes. With $k = 2$ active experts out of $E = 8$ total, we record whether each token's expert set on pass 2 fully overlaps, partially overlaps, or is entirely disjoint from its pass 1 assignment.

Routing predominantly diverges across loops. Figure 5 presents the per-layer breakdown of expert assignment overlap. Across layers 1–6 and 8, 25–53% of tokens receive entirely non-overlapping expert assignments between passes, while only 4–14% receive identical assignments. For a majority of tokens, exactly one of the two active experts is shared between loops—the other changes. This demonstrates that MoE routing enables shared layers to deploy substantially different expert sub-networks on each loop iteration for the majority of tokens.

Layer-specific behavior. Layer 7 is a notable outlier, exhibiting markedly higher overlap (37% exact match and only 3% zero overlap). We speculate this loop-invariance reflects a structural role for layer 7, possibly stabilizing embeddings before vocabulary projection at the loop boundary. Overall, we observe unique computations per loop, unlike the fixed computation of a dense repeated layer.

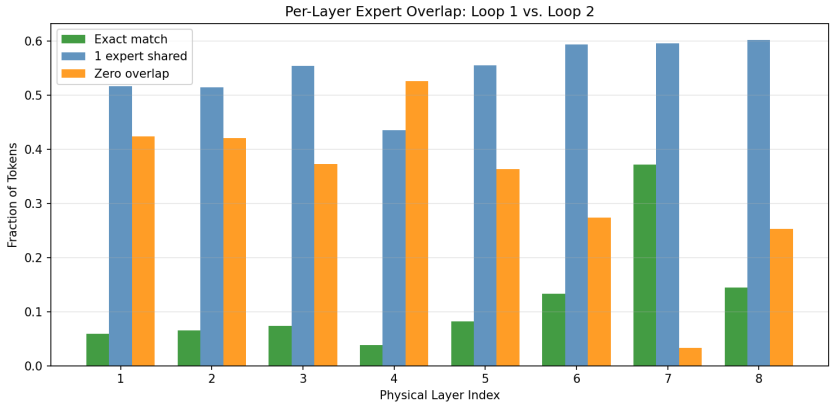


Figure 5: Expert assignment overlap between loop passes 1 and 2 across physical layers in a Looped-MoE model (8 layers $\times$ 2 loops, $k = 2$, $E = 8$). The majority of tokens receive a unique set of experts across loops, with layer 7 as a notable exception exhibiting high routing consistency.

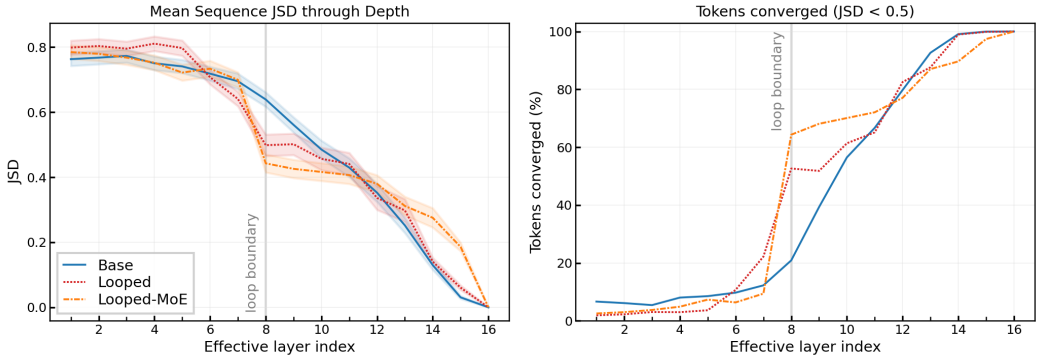


Figure 6: Distributional analysis of intermediate layer outputs relative to the final layer output (JSD), using compute-optimal models at 10^{18} FLOPs. **Left:** Mean JSD at each effective layer. Shaded bands show standard deviation of JSD across sequences. **Right:** Fraction of tokens at a given effective layer with JSD < 0.5. Looped models converge substantially at loop boundaries, supporting our hypothesis that shared exit layers produce well-formed outputs.

6.2 Why are Early Exits for Looped Transformers Favorable?

In looped models the layers at each exit point are the same layers that produce the final output projected to vocabulary space. We hypothesize that this architectural property results in loop boundary activations being better formed for vocabulary projection and thus early exiting. We test this by measuring how similar each layer's output distribution is to the final output, expecting looped models to be closer to final distributions at loop boundaries.

Setup. We evaluate the compute-optimal model for each architecture at the 10^{18} FLOPs budget on our test tokens. We capture hidden states at every effective layer during inference and project each through the final layer norm and language model head [24]. We then compute the Jensen-Shannon divergence (JSD) between each intermediate output distribution and the final layer's output distribution. Concretely, for each token at effective layer ℓ:

$$\text{JSD}(p_\ell \| p_L) = \frac{1}{2} D_{\text{KL}}(p_\ell \| m) + \frac{1}{2} D_{\text{KL}}(p_L \| m), \quad m = \frac{1}{2}(p_\ell + p_L) \tag{1}$$

where p_ℓ is the distribution at layer ℓ, p_L is the final layer's distribution, and m is their average. We normalize JSD to $[0, 1]$ and consider a token converged if its normalized JSD falls below 0.5.

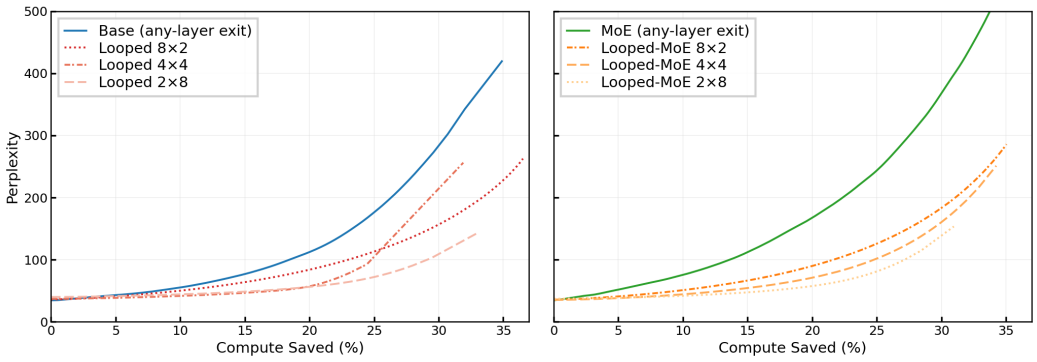


Figure 7: **Left:** More loops improve the Looped compute-quality tradeoff, though not strictly at all savings levels. **Right:** For Looped-MoE, more loops yield a strictly better compute-quality tradeoff, with all configurations better than non-looped MoE.

Looped models converge faster at loop boundaries. Figure 6 shows the mean JSD and fraction of converged tokens (JSD < 0.5) at each effective layer. Looped models converge a substantially larger fraction of tokens at earlier depths than Base, with a sharp jump at the loop boundary. By the end of the first loop iteration, the majority of tokens have already reached near-final output distributions.

The mechanism is architectural, not learned. In a looped model, every loop iteration ends with the same layers that produce the final output. An early exit at the end of loop 1 projects through layers that were trained as the model's output-facing computation. In a non-looped model, an exit at the equivalent depth (layer 8 of 16) projects through layers that were trained as intermediate representations, never optimized to produce well-formed output distributions. This property requires no additional training signal—it is an inherent consequence of layer looping.

6.3 Does More Looping Improve Early-Exit Efficiency?

The looped architecture introduces natural early-exit points at loop boundaries—positions where a full pass through the shared layer stack has completed. A model with L physical layers and R loops has $R - 1$ such exit points: 1 for 8×2, 3 for 4×4, and 7 for 2×8. We ask: does increasing the number of loops (and thus exit points) improve the compute-quality tradeoff?

Setup. We train Looped and Looped-MoE variants at the 10^{18} FLOPs budget with three loop-depth configurations: 8×2 (8 layers, 2 loops), 4×4 (4 layers, 4 loops), and 2×8 (2 layers, 8 loops). All three share the width of the compute-optimal 8×2 configuration ($d_{\text{model}} = 704$) and have 16 effective layers. We apply the same entropy-based early-exit procedure described in Section 4.2, with tokens exiting at the first loop boundary where entropy falls below τ.

More loops improve the early-exit tradeoff. Increasing the number of loops generally yields a strictly better compute-quality tradeoff, as reported in Figure 7. For Looped-MoE, the 4×4 compute-quality curve lies below the 8×2 curve, and 2×8 lies below both. This improvement arises from two factors: (1) more loop boundaries provide more opportunities to exit where the output is sufficiently converged; and (2) every loop boundary is a high-quality exit point, since the token has passed through the same layers that produce the final output.

Looped-MoE outperforms non-looped MoE. All three Looped-MoE configurations achieve a better early-exit Pareto frontier than the non-looped MoE baseline, despite the non-looped MoE having access to 15 candidate exit layers with no restriction on which layer a token may exit at. The looped variants, by contrast, are restricted to exiting only at loop boundaries. Even with this structural disadvantage, looped models offer a better compute-quality tradeoff.

7 Related Works

Looped Transformer Architectures. MoEUT [6] extends the looping Universal Transformer with σ-MoE [25] but also includes SwitchHead [26] and a modified LayerNorm [27] as part of a broader redesign. We focus on the standard token-choice top-k MoE, and specifically choose to only change this from baseline dense looped model, to investigate whether dense FFNs are the expressive bottleneck for looped architectures. The Ouro model family [2] is full-scale 1.4B and 2.6B looped transformer, however they loop dense FFNs, which we have shown are sub-optimal when truly compared to dense baselines on fixed compute. In their study, they compare performance by the number of unique parameters, but giving their models much more compute (4x the effective depth) than the models in their comparisons. In contrast, we conduct a comparative study in the isoFLOP setting: understanding instead when compute is fixed, which model architectures scale better. Concurrently, Parcae [23] stabilizes looped training via spectral norm constraints and establishes looping as an orthogonal scaling axis to data. Our work is complementary, focusing on the expressivity bottleneck of dense looped layers and early-exit efficiency.

Early Exits and Layer Skipping. Early exits have been explored extensively for deep neural networks and more recently for transformers. LayerSkip [28] trains standard transformers with layer dropout and a shared exit across all layers, enabling early-exit inference. In contrast, we observe looped architectures possess this property by construction: the same stack of layers already handles final activations during normal training. Mixture of Depths [29] dynamically routes tokens to skip layers via a learned top-k mechanism, and Mixture of Recursions [30] extends this to looped architectures by learning per-token recursive depths. Like MoEUT, these works demonstrate favorable scaling for compute-adaptive models, but involve substantially more complex deviations from standard architectures. Our early-exit analysis instead offers a focused study of what enables looped models to scale: we find that switching to sparse MoE layers is the key step change, and that the resulting convergence properties naturally support early exiting as a downstream benefit rather than as a separate architectural modification.

8 Conclusion and Limitations

Conclusion. We present a controlled study comparing dense, MoE, looped, and Looped-MoE transformers, isolating the effect of sparse expert layers on looped model scaling and downstream performance. Our results show that replacing dense FFNs with top-k MoE layers resolves the scaling deficit of looped transformers. On standard benchmarks, the compute-optimal Looped-MoE achieves the best overall accuracy while storing fewer parameters than the dense baseline. We provide a mechanistic explanation for Looped-MoE's advantage: MoE routing diverges across loop iterations, allowing shared layers to implement distinct computations per loop. We also find that looping drives earlier output convergence at loop boundaries, enabling training-free early exits with better compute-quality tradeoffs. Together, these findings suggest that Looped-MoE models are a practical path toward language models that are cheaper to store, faster to run, and competitive in quality.

Limitations. μP transfer does not hold when model depth changes, so we hold depth constant and scale width. Future work could validate with depth-scaling extensions such as CompleteP [31], though this adds another axis of comparison: the number of unique layers repeated by looping would also change with depth. Due to compute constraints, we did not scale our four architectures beyond 305M/711M (active/stored) parameters, but rely on the principle that compute-optimal scaling laws fitted at smaller scales predict larger model performance. We aim to validate this with extended pretraining at 1B and 7B scales. Finally, our early-exit results demonstrate theoretical compute savings via a layer exit criterion, but we have not yet measured end-to-end throughput gains in an optimized auto-regressive inference setting.

References

[1] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers, March 2019. URL http://arxiv.org/abs/1807.03819. arXiv:1807.03819 [cs].

[2] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling Latent Reasoning via Looped Language Models, November 2025. URL http://arxiv.org/abs/2510.25741. arXiv:2510.25741 [cs].

[3] 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, February 2025. URL http://arxiv.org/abs/2502.05171. arXiv:2502.05171 [cs].

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, January 2020. URL http://arxiv.org/abs/2001.08361. arXiv:2001.08361 [cs].

[5] 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, January 2017. URL http://arxiv.org/abs/1701.06538. arXiv:1701.06538 [cs].

[6] Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D Manning. MoEUT: Mixture-of-Experts Universal Transformers.

[7] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, Cancun, December 2016. IEEE. ISBN 978-1-5090-4847-2. doi: 10.1109/ICPR.2016.7900006. URL http://ieeexplore.ieee.org/document/7900006/.

[8] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference, April 2020. URL http://arxiv.org/abs/2004.12993. arXiv:2004.12993 [cs].

[9] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident Adaptive Language Modeling, October 2022. URL http://arxiv.org/abs/2207.07061. arXiv:2207.07061 [cs].

[10] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding, November 2023. URL http://arxiv.org/abs/2104.09864. arXiv:2104.09864 [cs].

[11] Noam Shazeer. GLU Variants Improve Transformer, February 2020. URL http://arxiv.org/abs/2002.05202. arXiv:2002.05202 [cs].

[12] Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

[13] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1):120:5232–120:5270, January 2022. ISSN 1532-4435. URL https://dl.acm.org/doi/10.5555/3586589.3586709.

[14] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing Stable and Transferable Sparse Expert Models, April 2022. URL http://arxiv.org/abs/2202.08906. arXiv:2202.08906 [cs].

[15] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of Experts, January 2024. URL http://arxiv.org/abs/2401.04088. arXiv:2401.04088 [cs].

[16] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer, March 2022. URL http://arxiv.org/abs/2203.03466. arXiv:2203.03466 [cs].

[17] Jan Małaśnicki, Kamil Ciebiera, Mateusz Boruń, Maciej Pióro, Jan Ludziejewski, Maciej Stefaniak, Michał Krutul, Sebastian Jaszczur, Marek Cygan, Kamil Adamczewski, and Jakub Krajewski. μ-parametrization for mixture of experts, October 2025. URL http://arxiv.org/abs/2508.09752. arXiv:2508.09752 [cs].

[18] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models, March 2022. URL http://arxiv.org/abs/2203.15556. arXiv:2203.15556 [cs].

[19] 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, October 2024. URL http://arxiv.org/abs/2406.17557. arXiv:2406.17557 [cs].

[20] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies, June 2024. URL http://arxiv.org/abs/2404.06395. arXiv:2404.06395 [cs].

[21] Aleksandr Dremov, Alexander Hägele, Atli Kosson, and Martin Jaggi. Training Dynamics of the Cooldown Stage in Warmup-Stable-Decay Learning Rate Scheduler, August 2025. URL http://arxiv.org/abs/2508.01483. arXiv:2508.01483 [cs].

[22] Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A Standard for Language Model Evaluations, February 2025. URL http://arxiv.org/abs/2406.08446. arXiv:2406.08446 [cs].

[23] 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. Version Number: 1.

[24] nostalgebraist. interpreting GPT: the logit lens — LessWrong. August 2020. URL https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

[25] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. Approximating Two-Layer Feedforward Networks for Efficient Transformers, November 2023. URL http://arxiv.org/abs/2310.10837. arXiv:2310.10837 [cs].

[26] Róbert Csordás, Piotr Piękos, Kazuki Irie, and Jürgen Schmidhuber. SwitchHead: Accelerating Transformers with Mixture-of-Experts Attention, September 2024. URL http://arxiv.org/abs/2312.07987. arXiv:2312.07987 [cs].

[27] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, July 2016. URL http://arxiv.org/abs/1607.06450. arXiv:1607.06450 [stat].

[28] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A. Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://arxiv.org/abs/2404.16710. arXiv:2404.16710 [cs].

[29] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-Depths: Dynamically allocating compute in transformer-based language models, April 2024. URL http://arxiv.org/abs/2404.02258. arXiv:2404.02258 [cs].

[30] 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, October 2025. URL http://arxiv.org/abs/2507.10524. arXiv:2507.10524 [cs].

[31] Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers, January 2026. URL http://arxiv.org/abs/2505.01618. arXiv:2505.01618 [cs].

A Appendix

Compute. All experiments were run on NVIDIA H100 GPUs (80GB). Approximate GPU-hours by experiment: isoFLOP scaling study (∼1,200), μP transfer validation (∼200), early-exit and loop-depth variants (∼300), analysis (∼100). Total: ∼2,000 GPU-hours. All training data fits within 2TB of storage.

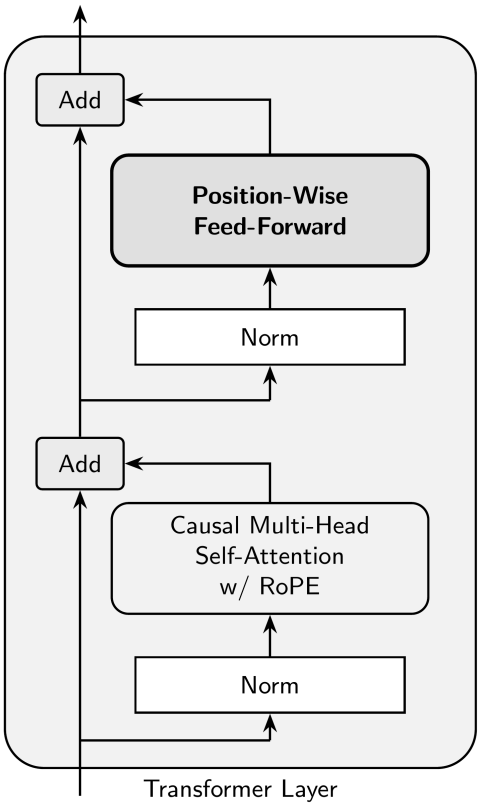
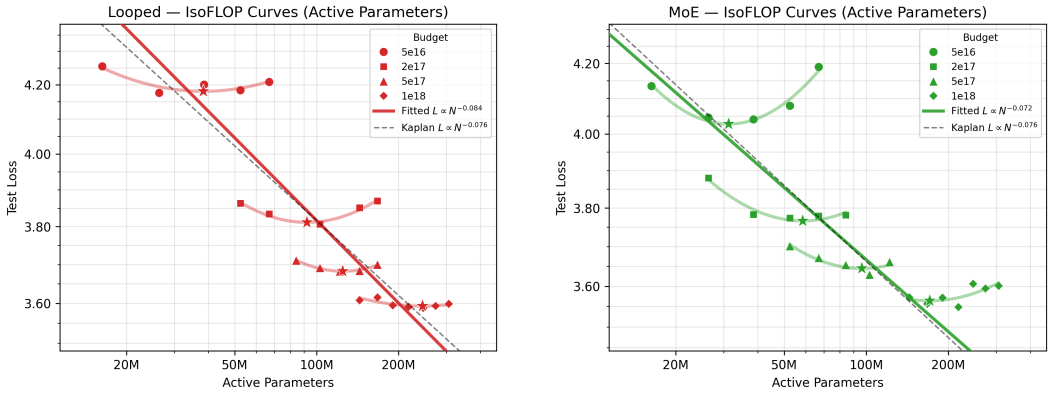


Figure 8: The pre-norm transformer layer, which we use in this study across all models. For our position-wise feed-forward network (FFN), we use SwiGLU. For the norms we use an RMS Norm without a learnable gain.

Table 5: Model configurations in this study. d_{ff} is rounded up to the nearest multiple of 64. Active parameters (including embeddings) are equal across all four architectures at each scale; stored parameters vary by architecture (see Table 3). Some isoFLOP runs use intermediate widths between the sizes in this table. Parameter counts for MoE architectures are with k=2 active experts out of E=8 total.

d_{model}	d_{ff}	n_{heads}	w_{ratio}	Active (M)	Stored LMoE (M)	Stored MoE (M)
128	384	2	1.0	16	18	23
256	704	4	2.0	39	45	65
384	1024	6	3.0	67	81	124
512	1408	8	4.0	103	129	207
640	1728	10	5.0	144	184	303
768	2048	12	6.0	190	247	417
896	2432	14	7.0	246	325	560
1024	2752	16	8.0	305	407	711



(a) Looped (b) MoE

Figure 9: **Left:** IsoFLOP curves for Looped. **Right:** IsoFLOP curves for MoE. Power law fit to compute budgets from 5×10^{16} to 10^{18} FLOPs. Dashed lines show fitted power-law scaling relations. Kaplan et al. scaling exponent is shown in dotted line.

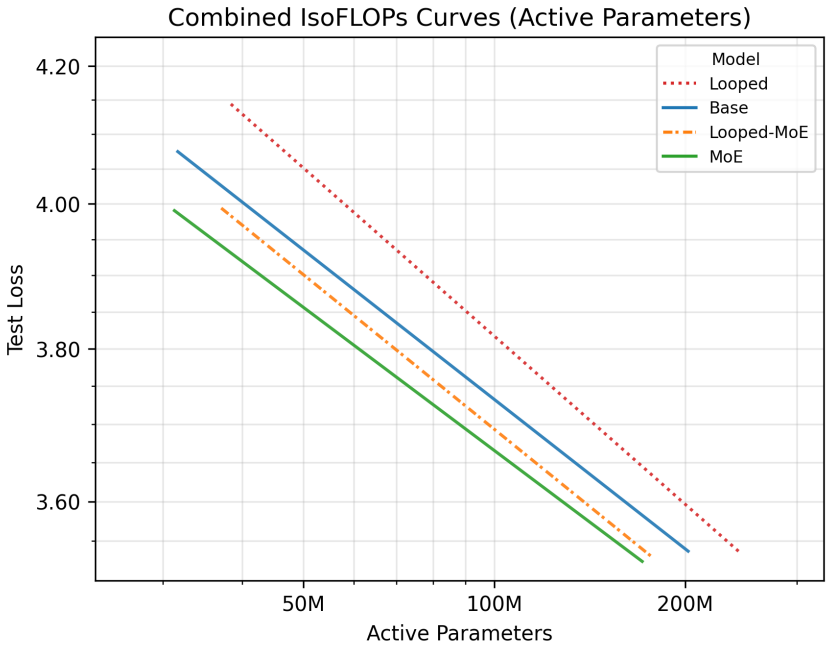


Figure 10: Combined scaling laws for all models in the study. Lower test loss is better. All architectures have roughly the same scaling slope, but are offset differently. MoE scales the best, followed by Looped-MoE, then Base and lastly Looped. Looping the same architecture seems thus to strictly reduce expressivity. However, in this paper we focus on how adding looping and MoE improves upon dense Base scaling and also Looped scaling.

Table 7: Perplexity at increasing levels of FLOPs saved via entropy-based early exit (training-free). The *Full depth* column shows baseline perplexity with no early exit; each subsequent column shows perplexity when that fraction of FLOPs is skipped. Sparse experts alone do not improve early-exit tolerance — MoE degrades faster than Base. The benefit is attributable specifically to looping: Looped achieves lower perplexity than both Base and MoE at every savings level, and the advantage grows with the number of loops. At 10% FLOPs saved, Looped-MoE 2×8 reaches perplexity 42.0 versus 55.4 for Base and 75.7 for MoE.

Model	Full depth	5%	10%	20%	30%
		FLOPs Saved			
Base	34.8	43.1	55.4	112.6	272.6
MoE	34.8	51.9	75.7	167.5	369.2
Looped	36.2	40.9	50.2	84.3	156.0
Looped-MoE	35.9	41.1	51.0	89.7	182.6
Looped-MoE 4×4	35.4	38.2	44.3	71.0	160.1
Looped-MoE 2×8	37.3	38.8	42.0	57.6	153.5

Table 6: Full AI2 OLMES benchmark results at 10^{18} FLOPs for all four architectures. MoE scores lowest on average (36.4) despite achieving the best test loss. We hypothesize this reflects narrower per-token expert access: each token consults only k=2 experts per layer in a single pass, whereas Looped-MoE tokens access 3–4 unique experts per physical layer across loops due to routing divergence (Section 6.1), providing broader per-token coverage on knowledge-intensive tasks.

Model	**Params**	**ARC-E**	**ARC-C**	**BoolQ**	**CSQA**	**HellaSwag**	**OBQA**	**PIQA**	**SIQA**	**WinoG**	**Core 9**
Looped	168M	39.1	24.6	49.6	27.1	25.6	24.4	57.1	38.2	51.2	37.4
Base	246M	39.3	**25.3**	51.4	29.3	26.2	**27.6**	**57.8**	**40.5**	50.5	38.7
Looped-MoE	216M	**40.4**	24.3	**63.9**	**30.9**	25.4	26.8	55.2	38.0	**51.7**	**39.6**
MoE	366M	38.9	23.0	39.0	30.0	**26.7**	25.2	56.1	39.2	49.7	36.4