LoopMoE: Unifying Iterative Computation with Mixture-of-Experts for Language Modeling

Anonymous ACL submission

Abstract

Mixture-of-Experts (MoE) and looped architectures scale models along two orthogonal axes, namely parameter capacity and effective depth. However, mainstream looped architectures rely on dense backbones that couple parameter count with per-token FLOPs, which makes it impossible to isolate the effect of iterative computation under matched budgets. To this end, we present LoopMoE, a looped MoE language model that integrates sparse routing with iterative weight-shared computation through two designs. The first is Iter-AdaLN, which resolves weight-sharing symmetry via a modulation signal jointly conditioned on the iteration index and the per-token hidden state. The second is a capacity-balancing strategy that recovers the attention-to-FFN active parameter ratio of well-tuned non-looped references. Together, these designs enable the first strictly controlled, head-to-head evaluation of a looped MoE against a Vanilla MoE under identical total parameters, per-token FLOPs, and active sublayer ratios. At the 3B scale, LoopMoE outperforms the Vanilla MoE on 8 of 9 downstream benchmarks with an average improvement exceeding 1 point. At the 9B scale, LoopMoE continues to outperform the matched Vanilla MoE, indicating that the architectural gain persists at larger scale. Our work establishes a controlled synthesis of sparsity and recurrence, and suggests a promising direction for looped language models.

1 Introduction

Capability scaling in modern large language models (LLMs) has been predominantly driven by parameter expansion, with Mixture-of-Experts (MoE) architectures emerging as the de facto standard for achieving this at a tractable cost (Liu et al., 2024; Yang et al., 2025; Zeng et al., 2025). The efficacy of MoE stems from a structural decoupling, in which activating only a sparse subset of experts per token disaggregates total parameter count from per-token active compute and enables parameter growth without commensurate FLOP inflation. However, scaling total parameters is not the sole avenue for enhancing capability per unit of training cost. An orthogonal paradigm investigates how to extract greater computational depth from a strictly bounded parameter budget. Among these approaches, block-recurrent or looped architectures offer a compelling solution by executing a shared block of layers over multiple iterations, thereby increasing depth without introducing new parameters (Dehghani et al., 2018; Lan et al., 2019; Bae et al., 2025; Mohtashami et al., 2025; Bae et al., 2026; Geiping et al., 2026).

While block-recurrent architectures successfully increase computational depth, existing dense loops (Zhu et al., 2025; Zeitoun et al., 2026; Jeddi et al., 2026) suffer from a fundamental structural flaw: parameter count and per-token FLOPs remain tightly coupled. Iterating a dense block multiplies compute without altering parameters, which makes it impossible to simultaneously match a non-looped baseline on both axes. MoE architectures (Cai et al., 2025), which naturally decouple total parameters from active compute, offer a theoretical solution. However, naively integrating loops into an MoE backbone introduces two critical new challenges. Representationally, weight sharing across iterations imposes structural symmetry that severely restricts the model's ability to evolve token states progressively. Structurally, it induces asymmetrical active parameter expansion: attention parameters are reused uniformly across iterations, whereas tokens dynamically route to different experts at each step. As a consequence, the active attention-to-FFN parameter ratio ρ deviates substantially from the operating point $\rho^\star$ of well-tuned baselines.

To fully unlock the potential of iterative sparse computation, we introduce **LoopMoE**, a novel block-recurrent MoE language model designed to overcome both limitations. LoopMoE adopts a

streamlined sandwich layout in which the core loop body evolves through pure recursion. To break the structural symmetry inherent in weight sharing, we introduce **IterAdaLN**, a token-level modulation scheme inspired by adaptive normalization in conditional generation (Perez et al., 2018; Peebles and Xie, 2023). Operating strictly at a token-level granularity, IterAdaLN dynamically generates affine parameters jointly from the iteration index and the per-token hidden state. Replacing static RMSNorm (Zhang and Sennrich, 2019) with this token-level conditioning ensures sufficient differentiation across iterations. It also allows the representational trajectory to remain highly expressive without requiring iteration-specific prefix re-injection.

Crucially, LoopMoE leverages the inherent flexibility of the MoE framework to resolve the asymmetrical expansion problem. To maintain the optimal ratio $\rho^\star$, we adopt a principled balancing strategy that expands the Q/KV LoRA ranks of Multi-head Latent Attention (MLA) (Liu et al., 2024) and correspondingly reduce the expert hidden dimensions. This decoupling allows independent adjustment of attention and FFN capacities. We then recover the baseline's total parameter count $N^\star$ by scaling the routed expert count. Together, these designs turn the MoE structure into the exact degree of freedom needed to construct a rigorously controlled looped architecture.

Our main contributions are summarized below:

- **Novel Architecture**: We propose **LoopMoE**, which couples sparse expert routing with iterative weight-shared computation through two key designs: **IterAdaLN**, a token-level iteration-conditioned modulation that breaks weight-sharing symmetry, and a capacity-balancing strategy that maintains the attention-to-FFN active-parameter ratio of well-tuned non-looped MoEs.

- **Strictly Controlled Evaluation**: We provide the first head-to-head comparison of a looped architecture against a Vanilla MoE under rigorously matched total parameters, per-token FLOPs, and active sublayer ratios, cleanly isolating the architectural contribution from parameter or compute inflation.

- **Consistent and Scale-Robust Empirical Gains**: Under this matched setting at 3B scale, LoopMoE outperforms the Vanilla MoE on 8 of 9 benchmarks while accessing fewer physical active parameters per token, with the average improvement exceeding 1.0 point. On reasoning and mathematics, LoopMoE further remains competitive with OLMoE-1B-7B, despite the latter having over twice the total parameter count. At the 9B scale, LoopMoE continues to outperform the matched Vanilla MoE at an early-training checkpoint, indicating that the architectural benefit is preserved at a larger scale rather than being a small-scale artifact.

2 Related Work

2.1 Mixture-of-Experts Language Models

Mixture-of-Experts has become a standard approach for scaling LLMs under a controlled compute budget (Lepikhin et al., 2020; Fedus et al., 2022; Cai et al., 2025). Recent systems explore fine-grained experts, shared experts, and auxiliary-loss-free load balancing (Jiang et al., 2024; Xue et al., 2024; Liu et al., 2024; Yang et al., 2025; Zeng et al., 2025). These architectures all assume standard non-shared per-layer parameters, and their interaction with iterative weight sharing remains largely unexplored. Our work targets exactly this intersection.

2.2 Looped and Weight-Shared Transformers

Looped and weight-shared architectures decouple parameter count from effective depth by reusing the same parameters for iterative refinement (Hutchins et al., 2022). Universal Transformer (Dehghani et al., 2018) ties a single block across depth and applies per-token adaptive halting, whereas ALBERT (Lan et al., 2019) shares one block across all layers of an otherwise standard encoder purely for parameter efficiency. More recent designs revisit input-dependent depth and halting (Bae et al., 2025, 2026), explicit iterative reasoning (Mohtashami et al., 2025), and conditional modulation of shared weights across iterations (Jeddi et al., 2026). A closely related line of work also studies latent reasoning through recurrence in the hidden state (Hao et al., 2024; Saunshi et al., 2025). A separate line of work (McLeish et al., 2025; Zeitoun et al., 2026; Geiping et al., 2026) wraps a recurrent middle block with non-looped layers on either side, a prefix-loop-suffix design that we also adopt.

2.3 Adaptive LayerNorm and Conditional Modulation

Conditional modulation (Huang and Belongie, 2017; Perez et al., 2018), and in particular Adaptive

LayerNorm (Peebles and Xie, 2023), lets shared parameters behave differently under different conditions. It is now standard in diffusion transformers. Looped LLMs face an analogous need across iterations. LoopFormer (Jeddi et al., 2026) conditions on the iteration index, but its condition vector is shared across all tokens in a pass. This is restrictive for language modeling where tokens at different positions warrant different updates (Ainslie et al., 2023; Heakl et al., 2025). Our proposed IterAdaLN addresses this by conditioning on both the iteration and the per-token state (Section 3.3).

3 Methods

3.1 Overview

We propose LoopMoE, a looped MoE language model. Following DeepSeek-V3 (Liu et al., 2024), each layer pairs a Multi-head Latent Attention (MLA) (DeepSeek-AI, 2024) sublayer with a fine-grained MoE sublayer, and the first post-embedding layer uses a dense FFN. Each sublayer adopts a sandwich normalization scheme, with pre-normalization before the sublayer and an additional RMSNorm on its output prior to the residual addition. On top of this backbone, we follow the sandwich-loop layout from prior weight-shared architectures (McLeish et al., 2025; Zeitoun et al., 2026; Geiping et al., 2026), organizing layers into non-shared prefix and suffix blocks around a shared loop body executed K times. Inside the loop body, we introduce IterAdaLN (Section 3.3) to supply per-iteration, per-token modulation, and a capacity-balancing strategy (Section 3.4) to align total parameters, per-token FLOPs, and active sublayer ratios with a non-looped MoE reference of matched capacity (the Vanilla MoE). Figure 1 illustrates the overall layout and the per-iteration update inside one loop-block layer.

3.2 Sandwich-Loop Backbone

LoopMoE organizes its layers into three regions, namely a non-shared prefix layers L_p, a loop block of L_b shared layers executed K times, and non-shared suffix layers L_s. This configuration yields an effective depth of $D = L_p + K \cdot L_b + L_s$ while using only $U = L_p + L_b + L_s$ unique layers. The non-shared boundary layers provide dedicated capacity for input-embedding adaptation and output projection, while grouping the shared layers into one multi-layer block enables implicit intra-block information exchange across iterations. In particular, a refined representation produced by a later layer in iteration k becomes available to an earlier layer in iteration $k+1$, allowing the block as a whole to revisit and rewrite its intermediate states.

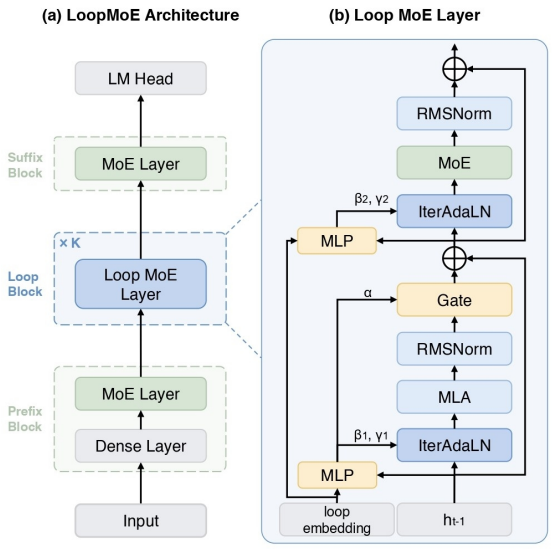

Figure 1: Architecture of LoopMoE. **Left:** the sandwich-loop layout with prefix layers, a loop block of shared layers executed K times, and suffix layers. **Right:** the per-iteration update inside the Loop MoE Layer, featuring IterAdaLN at both pre-normalization sites and an asymmetric residual gate α on the attention branch only.

A central design question is how information flows across iterations. Prior sandwich-loop models (McLeish et al., 2025; Geiping et al., 2026) re-inject the prefix output h_{pre} additively at every step, requiring an ill-defined h_0 at the first iteration and allowing a constant h_{pre} to dominate the residual stream, thereby suppressing iteration-to-iteration differentiation. We therefore remove the re-injection pathway so that the loop body takes h_{pre} as its starting point and evolves through K steps of pure recursion.

$$h_0 = h_{\text{pre}} \tag{1}$$

$$h_k = f_{\text{body}}(h_{k-1}; \mathbf{c}_{k,t}), \quad k = 1, 2, \ldots, K \tag{2}$$

where $\mathbf{c}_{k,t}$ is the conditioning vector for token t at step k and is subsequently consumed directly by IterAdaLN (Section 3.3). Each forward pass is therefore a strict recursive refinement, and IterAdaLN becomes the sole pathway for per-iteration differentiation.

3.3 IterAdaLN

Weight sharing constrains the model to reuse identical parameters across all steps, so the computation

lacks iteration awareness without explicit conditioning. A natural remedy is to condition each iteration on a global iteration signal via Adaptive Layer Normalization (AdaLN) (Peebles and Xie, 2023). This is sufficient for image generation, where the diffusion timestep serves as a genuinely global conditioning target. Applying the same recipe to an MoE language model backbone, however, introduces a spatial bottleneck. A purely iteration-conditioned AdaLN forces a uniform affine modulation across all sequence positions at each step k, whereas natural language sequences are intrinsically heterogeneous. Syntactic markers, function words, and dense semantic entities demand qualitatively different representational updates at the same iteration. With only a global signal available, all sequence-level heterogeneity must be absorbed by the weights of the shared body.

We therefore introduce **IterAdaLN**, which generates modulation parameters jointly from the iteration index and the current token state, unlocking per-iteration and per-token degrees of freedom directly within the conditioning pathway. For token position t at loop iteration k, IterAdaLN forms branch-specific joint conditions $c^{\text{attn}}_{k,t}$ and $c^{\text{moe}}_{k,t}$ by fusing two streams of information. The token stream supplies local semantic context through a linear projection of the token's current state in the forward pass. For the attention branch, this state is the initial state $h_{k-1,t}$, while for the MoE branch, it is the intermediate state $m_{k,t}$ produced by the preceding attention update. The iteration stream follows the DiT-style time encoding (Peebles and Xie, 2023), mapping the iteration index k through a fixed sinusoidal positional encoding $\mathrm{PE}(k)$ and a small learnable MLP, $\mathbf{v}_k = \mathrm{MLP}_{\text{iter}}(\mathrm{PE}(k))$. The two streams are combined by broadcast addition for each branch,

$$c^{\text{attn}}_{k,t} = \mathrm{Linear}_{\text{attn}}(h_{k-1,t}) + \mathrm{MLP}_{\text{iter}}\big(\mathrm{PE}(k)\big) \tag{3}$$

$$c^{\text{moe}}_{k,t} = \mathrm{Linear}_{\text{moe}}(m_{k,t}) + \mathrm{MLP}_{\text{iter}}\big(\mathrm{PE}(k)\big) \tag{4}$$

so that both conditions inherit sequence-length variation from the token term and iteration variation from the time term.

Given the corresponding condition c, IterAdaLN replaces standard RMSNorm (Zhang and Sennrich, 2019) at every pre-normalization site with

$$\mathrm{IterAdaLN}(h;\, c) = \frac{\big(1 + \gamma(c)\big) \odot h}{\sqrt{\mathbb{E}[h^2] + \epsilon}} + \beta(c) \tag{5}$$

Note that IterAdaLN employs an affine-free RMSNorm. We omit the default static learnable scale of RMSNorm, since IterAdaLN dynamically supplies a token- and iteration-conditional scale.

To provide fine-grained, independent modulation, the affine parameters are generated by two separate, zero-initialized MLPs,

$$\big[\gamma^{\text{attn}}_{k,t},\, \beta^{\text{attn}}_{k,t},\, \alpha_{k,t}\big] = \mathrm{MLP}^{\text{attn}}_{\mathbf{0}}\big(c^{\text{attn}}_{k,t}\big) \tag{6}$$

$$\big[\gamma^{\text{moe}}_{k,t},\, \beta^{\text{moe}}_{k,t}\big] = \mathrm{MLP}^{\text{moe}}_{\mathbf{0}}\big(c^{\text{moe}}_{k,t}\big) \tag{7}$$

where the subscript $\mathbf{0}$ denotes zero initialization of the output projection.

We apply the residual gate $\alpha_{k,t}$ only to the attention branch. A symmetric gate on the MoE branch would be absorbed into the routing weights and interfere with load-balancing calibration (see Appendix A.3 for justification).

We accordingly express the per-token, per-iteration layer update as the sequential composition of an attention residual sub-block and an MoE residual sub-block. The intermediate state $m_{k,t}$ is first produced by the attention block,

$$m_{k,t} = h_{k-1,t} + \alpha_{k,t}\, \mathcal{A}\big(\mathrm{IterAdaLN}(h_{k-1,t};\, \gamma^{\text{attn}}_{k,t},\, \beta^{\text{attn}}_{k,t})\big) \tag{8}$$

where $\mathcal{A}$ is the MLA attention sublayer. This updated state $m_{k,t}$ is then used both to compute the condition $c^{\text{moe}}_{k,t}$ and as the input to the MoE block, yielding the final layer output,

$$h_{k,t} = m_{k,t} + \mathrm{MoE}\big(\mathrm{IterAdaLN}(m_{k,t};\, \gamma^{\text{moe}}_{k,t},\, \beta^{\text{moe}}_{k,t})\big) \tag{9}$$

These update rules reflect the asymmetric placement of $\alpha_{k,t}$. Following AdaLN-Zero (Peebles and Xie, 2023), the output projection of $\mathrm{MLP}_{\mathbf{0}}$ is zero-initialized, so that $\gamma_{k,t} = \beta_{k,t} = \alpha_{k,t} = 0$ at initialization. The K-step loop therefore begins as a sequence of near-identity transformations, from which per-iteration and per-token differentiation gradually emerges during optimization. Appendix A.8 empirically verifies that the trained model exploits this per-iteration, per-token freedom, exhibiting a back-loaded, attention-concentrated update schedule.

3.4 Capacity Balancing Strategy

Looped MoEs exhibit a structural distortion in active capacity allocation. The dense MLA sublayer reuses the same parameters at every iteration, while a token may route to different experts across the K iterations of a shared MoE layer, so that its unique

active FFN parameters accumulate with K. The resulting ratio $\rho_{\text{loop}} = A_{\text{attn}}/A_{\text{ffn}}$ drops well below the operating point $\rho^\star$ of well-tuned non-looped MoEs, leaving attention under-parameterized relative to the FFN pool (Section 5.3). Let N and F denote a model's total parameter count and per-token FLOPs, with $N^\star, F^\star$ the corresponding baseline values. As noted in Section 1, MoE sparsity decouples N from F, so once ρ is corrected we can independently restore N and F to $N^\star$ and $F^\star$. We achieve this alignment through a two-step procedure.

Measurement We first experimentally investigate the expected number of unique experts a token activates across the K iterations of one shared layer under top-k routing. The resulting A_{ffn} scales sublinearly but non-trivially with K, while A_{attn} remains constant (see Appendix A.4). We adopt a Vanilla MoE, a non-looped MoE, as our matched reference. We characterize it by its attention-to-FFN active ratio $\rho^\star = A^\star_{\text{attn}}/A^\star_{\text{ffn}}$, together with $N^\star$ and $F^\star$ introduced above, which jointly specify the target operating point.

Rebalancing and capacity restoration To elevate ρ toward $\rho^\star$, we expand the MLA low-rank projections to increase A_{attn}, and shrink each expert's hidden dimension to decrease A_{ffn}, while keeping top-k routing fixed. This preserves F but reduces N. We then restore N to $N^\star$ by scaling up the routed expert pool, exploiting the MoE decoupling of N from F.

4 Experimental Setup

4.1 Data

We pre-train all models on the publicly released OLMo-3 pre-training corpus Dolma3Mix (Olmo et al., 2025), a large-scale mixture spanning web documents, code, academic text, and curated high-quality sources. To enable a strictly controlled comparison, all our trained models consume an identical 200B-token subset of this corpus under the same document ordering and packing configuration.

4.2 Training and Evaluation Setting

All models are trained with the AdamW optimizer (Loshchilov and Hutter, 2017a) under a cosine learning-rate schedule (Loshchilov and Hutter, 2017b) with linear warmup. For MoE layers, we adopt top-k routing with k=6 together with a shared expert strategy following (Dai et al., 2024), and this routing configuration is identical between LoopMoE and Vanilla MoE. Our main 3B LoopMoE comprises 6 unique layers, organized as 2 prefix layers, a 2-layer block that is recurrently applied for 4 iterations with shared weights, and 2 suffix layers, which yields an effective depth of 12 layers. The 9B variants follow the same architectural recipe with proportionally scaled hidden dimensions and expert counts. Full hyperparameters for both scales are provided in Appendix A.1.

To assess downstream performance, we evaluate the models on a suite of 9 standard benchmarks covering two broad dimensions. The first dimension targets knowledge and language understanding, including MMLU (Hendrycks et al., 2021a), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), TriviaQA (Joshi et al., 2017), and RACE (Lai et al., 2017). The second dimension focuses on reasoning and mathematics, covering BBH (Kazemi et al., 2025), GSM8K (Cobbe et al., 2021), and MATH (Hendrycks et al., 2021b). All evaluations are conducted with lm-evaluation-harness (Gao et al., 2024) under the standard zero-shot or few-shot protocols prescribed for each benchmark. The evaluation settings are detailed in Appendix A.2. For external baselines whose full training exceeds 200B tokens, we report results at their 200B-token intermediate checkpoints when available, so that all numbers reflect a token-matched comparison.

5 Results

5.1 Main Results

Table 1 compares LoopMoE against dense, dense-loop, and MoE baselines across nine benchmarks spanning knowledge, general language understanding, commonsense reasoning, multi-step reasoning, and mathematical problem solving. Vanilla MoE is the matched non-loop MoE described in Section 3.4, sharing total parameters, per-token FLOPs, active sublayer ratio, and training tokens with LoopMoE.

LoopMoE delivers consistent general improvements over the Vanilla MoE. It outperforms the baseline on 8 of the 9 benchmarks with an average gain of over 1 point, the only exception being a marginal regression on MMLU. These improvements span both evaluation dimensions, covering knowledge and language understanding as well as reasoning and mathematics. This shows that intro-

Model	Params (T/A)	Train Tokens	Language & Knowledge						Reasoning & Math			Avg.
			MMLU	HSwag	PIQA	WGrd	TQA	RACE	BBH	GSM8K	MATH	
Dense Models												
Pythia-1.4B†	1.4B / 1.4B	200B	24.63	51.93	70.46	56.59	16.37	35.22	23.71	1.90	1.80	31.30
OLMo2-1B†	1.0B / 1.0B	200B	24.27	58.23	73.72	59.75	26.13	33.88	9.45	2.43	0.42	32.12
Qwen-3-1.7B⋆	1.7B / 1.7B	36T	62.46	67.09	–	66.30	–	–	53.51	70.28	25.80	–
Dense Loop / Recursive Models												
Hyperloop⋆	1B / 1B	100B	–	46.22	68.72	53.04	–	33.59	–	–	–	–
Griffin⋆	1B / 1B	300B	29.5	67.2	77.4	65.2	–	–	–	–	–	–
Ouro-1.4B-R4⋆	1.4B / 1.4B	7.7T	67.35	74.29	–	72.30	–	–	71.02	78.92	82.40	–
MoE Models												
DeepSeekMoE-2B⋆	2B / 0.3B	100B	–	54.8	72.3	–	16.6	–	–	–	–	–
OLMoE-1B-7B†	7B / 1B	200B	24.90	68.14	76.88	60.62	39.28	35.31	26.85	2.82	1.66	37.38
PowerMoE-3B	3B / 0.8B	3T	43.24	73.07	79.00	64.80	28.57	35.79	33.07	38.21	14.94	45.63
Ours												
Vanilla MoE	3B / 0.8B	200B	**27.86**	58.68	74.12	49.92	36.19	35.00	26.12	4.12	1.53	34.84
LoopMoE	3B / 0.8B‡	200B	26.81	**60.98**	**75.22**	**51.42**	**37.63**	**35.77**	**27.94**	**5.26**	**1.88**	**35.89**

Table 1: Main results across 9 downstream benchmarks. **(T/A)**: total / active parameters. † evaluated at the 200B-token intermediate checkpoint for token-matched comparison. ⋆ numbers reported from the original paper. ‡ Active parameters are reported as 0.8B to match the FLOPs-per-token of the Vanilla MoE for fair comparison; the actual physical active parameter count of LoopMoE is 0.6B, since shared attention parameters across K loop iterations contribute once to the physical count but K times to per-token FLOPs. HSwag = HellaSwag; WGrd = WinoGrande; TQA = TriviaQA. Best results between Vanilla MoE and LoopMoE are in **bold**.

ducing the loop architecture into a modern MoE stack yields a broad net benefit under a strictly controlled compute budget. While the gains are broad, they are not uniform in magnitude. The largest improvements concentrate on reasoning and mathematics, where all datasets improve by a substantial margin relative to the average. We attribute this concentration to the additional iterative computation that the loop provides. Tasks that reward multi-step reasoning benefit most directly from repeated refinement of the hidden state. Section 5.3 decomposes these contributions across the individual architectural components.

For broader context, Table 1 also lists external systems trained on comparable token budgets (Pythia-1.4B (Biderman et al., 2023), OLMo2-1B (OLMo et al., 2024), Hyperloop (Zeitoun et al., 2026), DeepSeekMoE-2B (Dai et al., 2024)) and on substantially larger ones (Qwen-3-1.7B (Yang et al., 2025), Griffin (De et al., 2024), Ouro-1.4B-R4 (Zhu et al., 2025), PowerMoE-3B (Shen et al., 2024)). Against token-matched dense baselines, LoopMoE substantially outperforms them under identical training volume. On reasoning and mathematics, it further remains competitive with OLMoE-1B-7B (Muennighoff et al., 2025), despite using less than half the total parameters. We include the trillion-token systems only to mark the current performance frontier and do not treat them as head-to-head competitors, since their numbers conflate architecture with training scale and pipeline maturity. We regard large-scale training as complementary to the architectural axis isolated in this work.

Model	Params (T/A)	Avg.
Vanilla MoE	9B / 1.9B	35.02
LoopMoE	9B / 1.9B‡	**36.17**

Table 2: Scaling results at 9B parameters under a token-matched comparison at an early-training checkpoint of 100B tokens. Average is over the same 9 benchmarks as Table 1 (see full per-benchmark results in Appendix A.5). The actual physical active parameter count for ‡ is 1.3B (see in in A.1).

5.2 Scaling Behavior

A central question for any looped architecture is whether the advantage observed at small scale persists as model capacity grows. To address this, we train 9B variants of both LoopMoE and the Vanilla MoE under the same training recipe and corpus, and report a strictly token-matched head-to-head comparison at an early-training checkpoint of 100B tokens. Table 2 reports results across the same nine benchmarks used in Section 5.1. LoopMoE outperforms the matched Vanilla MoE by an average of

Architecture	MMLU	HellaSwag	BBH	GSM8K
Vanilla MoE	**27.86**	58.68	26.12	4.12
Loop Base	27.15	58.87	27.33	**5.34**
+ IterAdaLN	25.96	59.19	27.26	**5.34**
++Balancing (LoopMoE)	26.81	**60.98**	**27.94**	5.26

Table 3: Ablation study on the core components of the proposed architecture.

Task	Vanilla MoE	Loop Base	LoopMoE
hyperbaton	31.25	50.83	**51.25**
navigate	40.83	52.50	**57.08**
salient trans. err. det.	**22.50**	16.67	15.00
logical deduct. 5 obj.	**22.50**	17.50	16.25

Table 4: Representative BBH subtask scores for Vanilla MoE, Loop Base, and LoopMoE. Here, "salient trans. err. det." and "logical deduct. 5 obj." stand for "salient translation error detection" and "logical deduction five objects", respectively. Full results across all 27 subtasks are provided in Appendix A.6.

1.15 points, marginally exceeding the 1.05 point gap observed at the 3B scale, indicating that the architectural benefit is preserved, and even slightly amplified, at larger scale.

Two observations follow. First, the gap between LoopMoE and the Vanilla MoE does not collapse with scale and remains positive at 9B. This directly addresses the common concern that loop architectures stop helping once parameter counts grow large. Second, the qualitative profile of where LoopMoE helps most, namely reasoning-heavy and multi-step tasks, is preserved across scales. This consistency suggests that the iterative-depth advantage is a structural property of the architecture rather than an artifact of any particular capacity regime.

5.3 Ablation Study

We dissect the contribution of each architectural component through a controlled ablation, presented in Table 3. We select four representative benchmarks that span the major capability dimensions evaluated in Section 5.1: MMLU (knowledge), HellaSwag (commonsense), BBH (multi-step reasoning), and GSM8K (mathematical problem solving). Each row adds one component on top of the previous, starting from the Vanilla MoE.

Moving from the Vanilla MoE to Loop Base introduces weight sharing across iterations. We observe clear gains on reasoning and math benchmarks, together with mixed but mostly small improvements on the knowledge and language-oriented tasks. This pattern indicates that the loop architecture, even before any further conditioning, already contributes most of the reasoning and mathematics improvement.

Adding IterAdaLN introduces token-level conditioning on the iteration state. The model can then modulate its computation across loop iterations based on per-token context. We observe a modest improvement in the commonsense dataset. This indicates that fine-grained per-token modulation captures meaningful distinctions in how different tokens benefit from iterative refinement. Multi-step reasoning and math remain essentially unchanged at this stage, suggesting that the computational benefit of loop iterations on these tasks is already realized without explicit conditioning. The remaining movement is a drop on MMLU, which the next component addresses by directly targeting the underlying structural imbalance.

The final component, Balancing, restores the attention-to-FFN parameter ratio to that of a standard non-loop transformer, correcting the FFN-heavy skew induced by weight sharing (Section 3). HellaSwag improves substantially, and MMLU and BBH recover relative to the previous row. A small regression on GSM8K accompanies this gain, consistent with prior observations that math relies more heavily on FFN computation (Geva et al., 2021; Jin et al., 2025). Across the aggregate of benchmarks, the trade-off is clearly favorable, and the resulting model achieves the best overall performance among all ablated configurations.

6 Analysis

6.1 BBH Subtask Analysis

The aggregate BBH gain of LoopMoE over the Vanilla MoE conceals a structured heterogeneity across subtasks. This offers finer-grained evidence on where iterative depth contributes most. Table 4 reports the two subtasks with the largest gains and the two with the largest regressions.

The two largest improvements, hyperbaton (+20.00) and navigate (+16.25), both require answers to be constructed through a sequence of dependent reasoning steps in which each step builds on context accumulated from prior iterations. The Loop Base alone already accounts for the bulk of these gains, capturing nearly the entire improvement of hyperbaton and roughly three-quarters of the improvement of navigate. This pattern indicates

that iterative depth is the dominant mechanism, while IterAdaLN and capacity balancing provide a smaller secondary contribution on tasks whose reasoning chains benefit from per-token modulation. The finding offers subtask-level corroboration of the main result that loop iterations supply the additional computation required for complex multi-step reasoning.

The two largest regressions, salient translation error detection and five-object logical deduction, exhibit a monotonic decline that is already present in Loop Base. We attribute this to reduced per-step activation breadth, since matching the Vanilla MoE in compute forces LoopMoE to activate fewer physical parameters per iteration.

Across the full 27-subtask distribution, gains and regressions align consistently with this principle. LoopMoE improves on tasks where reasoning is compositional and benefits from iterative refinement and regresses on tasks whose correctness depends on broad, one-shot information access within each step.

6.2 Iterative Routing Dynamics

Beyond aggregate task performance, we next examine the iterative structure of a looped MoE block, which raises a natural question about expert allocation, namely, whether successive iterations activate similar or dissimilar expert subsets. Activating largely disjoint subsets would let each iteration perform a functionally distinct computation and effectively expand the active parameter budget, whereas converging to the same subset would form a stable expert coalition that applies a consistent transformation under repeated application. To probe which regime the loop operates in, we measure two pairwise similarities between iterations. Cosine similarity of router inputs tracks continuous representation drift, and Jaccard similarity of selected expert sets tracks discrete routing change.

The heatmaps in Figure 2 reveal a three-phase structure that interpolates between the two extremes at different points in the loop. In the entry phase, iter 0 stands apart from all subsequent iterations and is especially distant in cosine similarity. This is the only iteration that reads directly from the non-loop prefix rather than from the previous block iteration, and it behaves as an adapter that projects the prefix distribution into the recurrent state. In the recurrent core, iters 1 and 2 form the tightest cluster and reach the highest pairwise Jaccard similarity, approaching the fixed-coalition regime. Once the

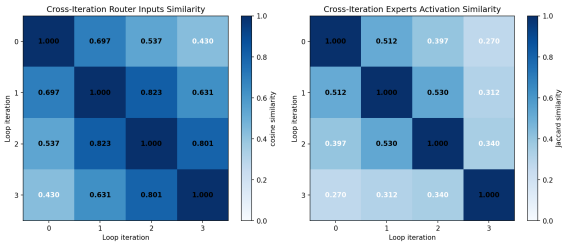

Figure 2: Cross-layer pairwise similarity between loop iterations on the BBH dataset. Router-input cosine (left) and expert-activation Jaccard (right). Per-layer matrices are reported in Appendix A.7.

prefix has been absorbed, the loop settles into a stationary internal recurrence carried by a consistent expert subset (Blayney et al., 2026). In the exit phase, the terminal iteration exhibits a notable decoupling between the two metrics. Representations remain close to those of the previous iteration, yet the router selects a substantially different expert subset, which suggests that the final pass reformats the consolidated state for the downstream non-loop layers rather than continuing to refine it. A per-layer decomposition that shows how the two loop layers contribute asymmetrically to this structure is provided in Appendix A.7.

7 Conclusion

We introduced LoopMoE, a looped MoE language model that couples sparse expert routing with iterative weight-shared computation. Two designs make this coupling viable. IterAdaLN breaks the symmetry imposed by weight sharing through a modulation signal jointly conditioned on the iteration index and the per-token hidden state. A capacity-balancing strategy recovers the attention-to-FFN active-parameter ratio of a well-tuned non-looped reference. Together, they enable the first rigorously matched evaluation of a looped architecture against a Vanilla MoE. Within this controlled setting, LoopMoE outperforms the Vanilla MoE on the vast majority of downstream tasks at the 3B scale while activating fewer physical parameters per token. At the 9B scale, LoopMoE continues to outperform the matched Vanilla MoE at an early-training checkpoint, indicating that the architectural benefit is preserved at larger scale and not confined to small models. These results establish looped sparse computation as an effective architecture for improving overall model capability within a fixed capacity budget.

Limitations

We acknowledge several limitations in our work. First, the dataset used for evaluation does not cover all domains, which may limit assessment of the model on specific or underrepresented fields. Second, our evaluation uses English as the main language, without consideration of multilingual scenarios. Third, while our results at 3B and 9B scales suggest that the architectural advantage is robust, confirming this trend at substantially larger scales remains an important direction for future work. All experiments are conducted within a limited training budget.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David C Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, and 1 others. 2023. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 5085–5100.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *International Conference on Learning Representations*, volume 2025, pages 34282–34327.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and 1 others. 2026. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, and 1 others. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2025. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, and 1 others. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, and 1 others. 2024. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

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

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

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2026. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38:41340–41391.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495.

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

Ahmed Heakl, Martin Gubri, Salman Khan, Sangdoo Yun, and Seong Joon Oh. 2025. Dr.llm: Dynamic layer routing in llms. *arXiv preprint arXiv:2510.12773*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. 2022. Block-recurrent transformers. *Advances in neural information processing systems*, 35:33248–33261.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. 2026. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations (ICLR)*.

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, and 1 others. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2025. Disentangling memory and reasoning ability in large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1681–1701.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Yuanzhu Peter Chen, and 1 others. 2025. Bigbench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26473–26501.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 785–794.

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

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Ilya Loshchilov and Frank Hutter. 2017b. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. 2025. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. 2025. Cotformer: A chain of thought driven architecture with budget-adaptive computation cost at inference. In *International Conference on Learning Representations*, volume 2025, pages 11503–11520.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, and 1 others. 2025. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pages 62061–62121.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, and 1 others. 2025. Olmo 3. *arXiv preprint arXiv:2512.13961*.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, and 1 others. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4172–4182. IEEE.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

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

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

Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D Cox, and Rameswar Panda. 2024. Power scheduler: A batch size and token number agnostic learning rate scheduler. *arXiv preprint arXiv:2408.13359*.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. 2024. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. 2026. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in neural information processing systems*, 32.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, and 1 others. 2025. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*.

A Appendix

A.1 Training Hyperparameters for LoopMoE

Table 5 summarizes the architectural and training hyperparameters of LoopMoE and the Vanilla MoE. Values not explicitly differentiated are shared across the two models to ensure a strictly controlled comparison. Due to the computational cost of pretraining 3B MoE models on 200B tokens, all reported results correspond to a single training run, with the same random seed for both Vanilla MoE and LoopMoE.

A.2 Detail LM Eval

We evaluate our models using the `lm-evaluation-harness` framework. To provide a clear overview of our evaluation protocol, the detailed settings for each downstream benchmark are summarized in Table 6.

A.3 Asymmetric Placement of the Residual Gate

A critical design choice concerns where the residual gate $\alpha_{k,t}$ should act. While DiT applies a symmetric gate to both sublayer outputs, we apply $\alpha_{k,t}$ exclusively to the attention branch. This design follows directly from the structure of the MoE operator. For an IterAdaLN-modulated input $\tilde{h}_{k,t}$, the MoE output is a routing-weighted sum over selected experts, so that a branch-level residual gate is absorbed straight into those routing weights,

$$\alpha_{k,t} \cdot \mathrm{MoE}(\tilde{h}_{k,t}) = \sum_{i \in \mathrm{Topk}=} \left(\alpha_{k,t}\, w_{k,t,i}\right) E_i(\tilde{h}_{k,t}). \quad (10)$$

The router therefore already provides the same per-token reweighting that an MoE-branch gate would offer, so the gate adds no representational capacity. It also actively interferes with routing. The routing weights $w_{k,t,i}$ are calibrated by the load-balancing loss and the expert-bias update (Liu et al., 2024) to remain in a well-behaved regime. Rescaling them by a data-dependent $\alpha_{k,t}$ perturbs that calibration along with the running statistics driving the expert-bias controller. The MLA attention sublayer, by contrast, has no analogous internal token-conditional reweighting mechanism, which makes a residual gate there strictly complementary.

A.4 Active Parameter Scaling under Loop Iterations

Figure 3 empirically characterizes how the attention and FFN active parameters evolve as the

Hyperparameter	3B		9B	
	Vanilla MoE	LoopMoE	Vanilla MoE	LoopMoE
Backbone				
Total parameters	3B	3B	9B	9B
Active parameters	0.8B	0.8B	1.9B	1.9B
Physical active parameters	0.8B	0.6B	1.9B	1.3B
Effective depth	12	12	18	18
Unique layers	12	6	18	9
Prefix layers L_p	–	2	–	3
Loop-block layers L_b	–	2	–	3
Suffix layers L_s	–	2	–	3
Loop iterations K	–	4	–	4
MLA				
Q LoRA Rank	384	512	768	896
K/V LoRA Rank	512	640	512	640
MoE-FFN				
routed experts	64	136	64	136
shared experts	2	2	2	2
Top-k	6	6	6	6
FFN Hidden Size	896	864	1280	1216
Optimization				
Optimizer	AdamW			
(β_1, β_2)	(0.9, 0.95)			
LR schedule	cosine			
Warmup steps	2000			
Peak learning rate	8.6e-4	8.6e-4	5.9e-4	5.9e-4
Min. LR ratio	7e-6	7e-6	5e-6	5e-6

Table 5: Architectural and training hyperparameters of LoopMoE and the Vanilla MoE at 3B and 9B scales.

Benchmark	n-shot	Metric
MMLU	5-shot	Accuracy
HellaSwag	10-shot	Accuracy
PIQA	0-shot	Accuracy
WinoGrande	0-shot	Accuracy
TriviaQA	5-shot	Exact Match
RACE	0-shot	Accuracy
BBH	3-shot	Exact Match
GSM8K	5-shot	Exact Match
MATH	4-shot	Exact Match

Table 6: Detailed evaluation settings across all downstream benchmarks.

number of loop iterations K increases, together with the resulting attention-to-FFN active ratio $\rho = A_{\text{attn}}/A_{\text{ffn}}$. The dense MLA sublayer reuses the same parameters across all iterations, so A_{attn} remains constant in K. In contrast, a token may route to different experts at each iteration of the shared MoE layer, so its unique active FFN parameters A_{ffn} accumulate with K. The growth is, however, sublinear rather than $K\times$: as K increases, the probability that a later iteration selects an already-activated expert grows, so the marginal contribution of each additional iteration to A_{ffn} shrinks. As a direct consequence, ρ drops monotonically and falls well below the operating point $\rho^\star$ of well-tuned Vanilla MoE (red dotted line), motivating the capacity-balancing strategy described in Section 3.4.

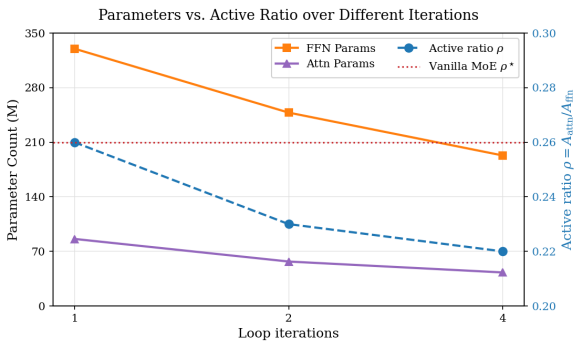

Figure 3: Attention and FFN active parameters (left axis) and the active ratio $\rho = A_{\text{attn}}/A_{\text{ffn}}$ (right axis) as a function of loop iterations K. A_{attn} is constant in K because the MLA sublayer is reused across iterations, while A_{ffn} grows sublinearly due to overlapping expert selections across iterations. The red dotted line marks the Vanilla MoE ratio $\rho^\star$.

A.5 Full 9B Scaling Results

Table 7 reports the complete per-benchmark results for the 9B scaling comparison summarized in Section 5.2. Both LoopMoE and the Vanilla MoE are trained under identical recipes, corpus, and a strictly token-matched budget of 100B tokens, with all evaluation protocols identical to those in Section 5.1.

Table 7 reports the complete per-benchmark results for the 9B scaling comparison summarized in Section 5.2. Both LoopMoE and the Vanilla MoE are trained under identical recipes, corpus, and a strictly token-matched budget of 100B tokens, with all evaluation protocols identical to those in Section 5.1.

The per-benchmark breakdown confirms the qualitative pattern reported in Section 5.2. LoopMoE outperforms the Vanilla MoE on 8 of the 9 benchmarks, with MMLU being the only regression—mirroring exactly the win pattern observed at the 3B scale. Gains are broadly distributed across the benchmark suite and concentrate on reasoning and mathematics, with GSM8K showing the largest absolute improvement. The aggregate improvement of 1.15 points indicates that LoopMoE's advantage over the Vanilla MoE is preserved, and slightly widens, at the 9B scale relative to the 1.05 point gap at 3B. As this is an early-training snapshot, we leave a full-budget 9B comparison to future work.

A.6 BBH Detailed Subtasks Result

Table 8 provides the complete per-task breakdown across all 27 subtasks within the BIG-bench Hard (BBH) suite. The disaggregated results provide a clearer view of the specific domains where the LoopMoE architecture excels.

A.7 Per-layer Routing Dynamics in the Loop Body

Figure 4 decomposes the three-phase routing structure into the two shared loop layers, revealing that they contribute asymmetrically.

Loop Layer 1 Loop Layer 1 shows monotonically increasing adjacent-pair cosine similarity across iterations, consistent with progressive fixed-point convergence of the recurrent block: once the prefix distribution has been absorbed at iter 0, successive applications drive the representation toward a stable attractor.

Loop Layer 2 Loop Layer 2 remains comparatively stable through the recurrent core but concentrates its functional shift at the read-out boundary, where its routing turnover (Jaccard drop at the terminal iteration) is substantially sharper than that of Loop Layer 1. This is consistent with Loop Layer 2 acting as the interface to the downstream non-loop suffix: rather than continuing to refine the recurrent state, it reformats it into a distribution suitable for the suffix layers.

Summary Neither extreme intuition—fully disjoint per-iteration experts nor a single fixed coalition—holds in isolation. The loop adopts the fixed-coalition strategy within the recurrent core while employing distinct expert subsets at the two interface boundaries, with the entry layer (Loop Layer 1 at iter 0) absorbing the prefix and the exit layer (Loop Layer 2 at the terminal iteration) preparing the hand-off to the suffix. The entry and exit layers thus play complementary roles in mediating the transition between non-loop and recurrent computation.

A.8 Emergent per-iteration allocation of residual updates

Setup and observations. We instrument the IterAdaLN modulation outputs (γ, β, α) of the two shared loop layers (L1 and L2) at every iteration $i \in \{0, 1, 2, 3\}$, separately for the attention branch and the FFN branch, averaged over the BBH dataset (full per-iteration statistics in Table 9). Since the asymmetric residual gating design (Section 3.3) fixes $\alpha = 1$ on the FFN branch, α is not included in ffn branch. Three findings emerge. First γ increases monotonically across loop iterations in all four cells, but the endpoint regime differs by branch. Attention branches cross into amplification by the late loop, while FFN branches remain in mild contraction throughout (never crossing zero). Under the IterAdaLN input scaling, early iterations on every branch suppress the normalised input, while late iterations amplify it for attention but only relax the suppression for FFN. Second, attention α is strongly back-loaded. $\overline{\alpha}$ grows monotonically on both layers, with the final-iteration gate substantially exceeding the residual baseline of 1. The L2 endpoint of 1.613 means the last iteration's attention contribution is amplified by over 60% relative to the residual baseline. Third, per-token dispersion of attention α grows monotonically with loop index. $tok_{std}(\alpha)$ increases roughly 3–6 times, and

Model	Params (T/A)	MMLU	HSwag	PIQA	WGrd	TQA	RACE	BBH	GSM8K	MATH	Avg.
Vanilla MoE	9B / 1.9B	**24.27**	62.74	74.61	51.32	37.53	33.50	26.65	4.45	1.26	35.02
LoopMoE	9B / 1.9B‡	23.37	**64.08**	**75.73**	**51.52**	**38.33**	**35.25**	**27.72**	**7.58**	**1.92**	**36.17**

Table 7: Full per-benchmark results at 9B scale under a token-matched comparison at an early-training checkpoint of 100B tokens. ‡ The physical active parameter count of LoopMoE is 1.3B, while the reported active parameter figure matches the per-token FLOPs of the Vanilla MoE for fair comparison. HSwag = HellaSwag, WGrd = WinoGrande, TQA = TriviaQA. Best results between Vanilla MoE and LoopMoE in each column are in **bold**.

Subtask	Vanilla MoE	Loop Base	LoopMoE
boolean_expressions	49.58	52.08	54.58
causal_judgement	47.16	48.86	50.57
date_understanding	17.92	15.00	12.08
disambiguation_qa	35.00	31.25	40.00
dyck_languages	0.00	0.00	0.83
formal_fallacies	41.25	53.33	48.33
geometric_shapes	20.00	19.17	27.08
hyperbaton	31.25	50.83	51.25
logical_deduction_five_objects	22.50	17.50	16.25
logical_deduction_seven_objects	14.17	13.33	13.33
logical_deduction_three_objects	34.17	26.25	32.50
movie_recommendation	24.17	28.75	19.58
multistep_arithmetic_two	0.42	0.00	0.83
navigate	40.83	52.50	57.08
object_counting	17.08	28.33	22.92
penguins_in_a_table	19.44	18.75	18.06
reasoning_about_colored_objects	12.08	15.83	9.58
ruin_names	22.50	23.33	17.08
salient_translation_error_detection	22.50	16.67	15.00
snarks	44.89	55.68	53.98
sports_understanding	51.25	50.00	53.33
temporal_sequences	25.00	26.67	25.42
tracking_shuffled_objects_five_objects	21.67	15.00	20.83
tracking_shuffled_objects_seven_objects	15.00	1.25	13.33
tracking_shuffled_objects_three_objects	32.50	32.92	33.75
web_of_lies	49.58	54.17	54.17
word_sorting	1.25	0.42	1.67
Average	26.12	27.33	27.94

Table 8: Per-subtask scores on BBH for Vanilla MoE, Loop Base, and LoopMoE.

the per-token range $tok_{range}(\alpha)$ on L2 attention grows from 0.761 to 3.432. It indicates early iterations apply a nearly token-uniform update, and late iterations differentiate tokens strongly. The shift β is negligible throughout ($|\overline{\beta}| < 4 \times 10^{-3}$ across all cells).

Interpretation and implications We term this pattern a back-loaded update schedule with asymmetric branch roles. In the early loop iterations, both branches operate in a light-touch phase, characterised by input contraction, modest attention gating ($\alpha < 1$), and near-token-uniform updates. The late iterations then diverge by branch. The attention branch enters a heavy-update phase, exhibiting input amplification ($\gamma > 0$), strong gating, and strongly token-differentiated updates. The FFN branch, in contrast, implements persistent contraction, with γ remaining ≤ 0 at every iteration (with one marginal exception). Since FFN $\alpha \equiv 1$ by design, the FFN contribution at every iteration is a consistently down-scaled update that merely becomes less down-scaled as the loop progresses. Effective late-loop computation is therefore concentrated specifically in the attention branch, both via its growing α gate and its growing per-token dispersion. This is consistent with emergent depth allocation under fixed-loop training. Weight sharing prevents the model from skipping iterations, but the model can learn to make early iterations approximate identity transformations and concentrate effective computation at the final iteration, choosing attention as the channel through which to do so. The monotonic growth of $tok_{std}(\alpha)$ and $tok_{range}(\alpha)$ reflects that per-token information

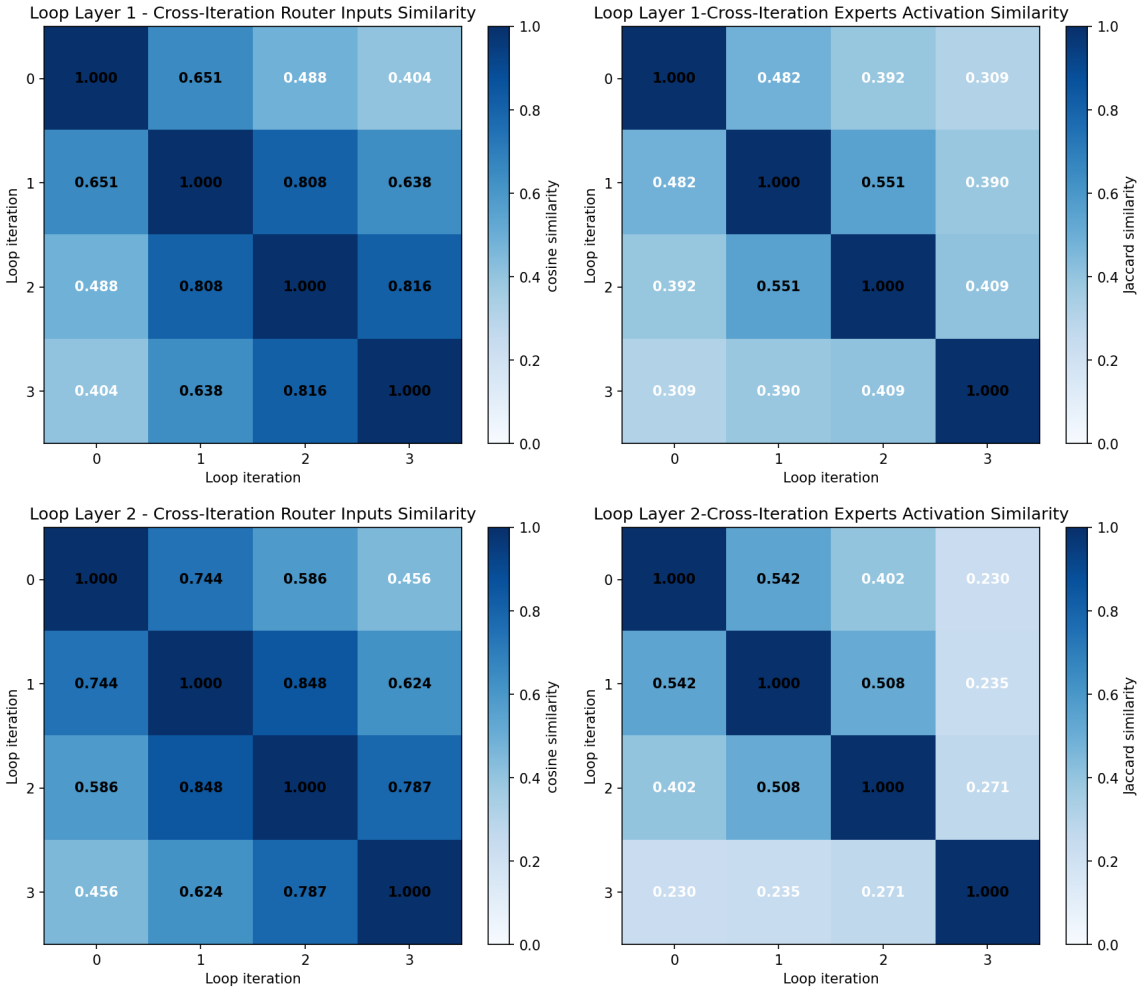

Figure 4: Per-layer cross-iteration routing dynamics within the shared block on the BBH dataset. The heatmaps display the router-input cosine similarity (left column) and expert-activation Jaccard similarity (right column) across the 4 loop iterations.

about update magnitude is encoded precisely where differentiation is highest. Since γ is conditioned on the evolving residual state h rather than a directly zero-initialized scalar, the γ sweep reflects learned behavior rather than initialization bias. This schedule does not yield a prefix-truncation inference speedup, since all four iterations are required to realise the late-loop attention amplification, and depth compressibility, to the extent it exists, is located at the front of the loop.

A.9 Use of AI Assistants

We used AI assistants for polishing the writing of this paper and for assisting with auxiliary coding tasks such as plotting and analysis scripts. All research ideas, experimental design, model implementation, and scientific claims are the authors' own, and all AI-generated content has been verified by the authors.

Module	i	$\overline{\gamma}$	$\overline{\lvert\gamma\rvert}$	$\overline{\beta}$	$\overline{\lvert\beta\rvert}$	$\overline{\alpha}$	tok_{std}	tok_{range}
L1 attn	0	-0.159	0.160	-0.000	0.001	$+0.707$	0.181	0.805
	1	$+0.049$	0.171	-0.000	0.001	$+0.805$	0.227	0.934
	2	$+0.237$	0.311	-0.000	0.001	$+1.044$	0.311	1.273
	3	$+0.503$	0.545	-0.000	0.001	$+1.355$	0.486	1.789
L1 ffn	0	-0.339	0.339	-0.001	0.001	—	—	—
	1	-0.276	0.280	-0.001	0.001	—	—	—
	2	-0.196	0.228	-0.000	0.001	—	—	—
	3	$+0.008$	0.179	$+0.001$	0.002	—	—	—
L2 attn	0	$+0.034$	0.150	$+0.001$	0.001	$+0.601$	0.147	0.761
	1	$+0.137$	0.227	$+0.002$	0.002	$+0.672$	0.225	0.941
	2	$+0.253$	0.322	$+0.002$	0.002	$+0.839$	0.353	1.373
	3	$+0.500$	0.554	-0.001	0.001	$+1.613$	0.922	3.432
L2 ffn	0	-0.252	0.252	$+0.000$	0.000	—	—	—
	1	-0.222	0.230	$+0.001$	0.001	—	—	—
	2	-0.168	0.199	$+0.002$	0.002	—	—	—
	3	-0.075	0.204	$+0.003$	0.003	—	—	—

Table 9: Full per-iteration IterAdaLN modulation statistics for the two shared loop layers, averaged over BBH dataset. $\overline{|\gamma|}$ and $\overline{|\beta|}$ denote the mean of absolute values of γ and β, tok_{std} and tok_{rng} are the per-token standard deviation and range of α.