
Mixture-of-RecurSIONs: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation

Sangmin Bae^{1*} Yujin Kim^{1*} Reza Bayat^{2*} Sungnyun Kim¹ Jiyoun Ha³ Tal Schuster⁴
Adam Fisch⁴ Hrayr Harutyunyan⁵ Ziwei Ji⁴ Aaron Courville^{2,6†} Se-Young Yun^{1†}

¹KAIST AI ²Mila ³Google Cloud ⁴Google DeepMind

⁵Google Research ⁶Université de Montréal

https://github.com/raymin0223/mixture_of_recurSIONs

Abstract

Scaling language models unlocks impressive capabilities, but the accompanying computational and memory demands make both training and deployment expensive. Existing efficiency efforts typically target either parameter sharing or adaptive computation, leaving open the question of how to attain both simultaneously. We introduce *Mixture-of-RecurSIONs* (MoR), a unified framework that combines the two axes of efficiency inside a single Recursive Transformer. MoR reuses a shared stack of layers across recursion steps to achieve parameter efficiency, while lightweight routers enable adaptive token-level thinking by dynamically assigning different recursion depths to individual tokens. This allows MoR to focus quadratic attention computation only among tokens still active at a given recursion depth, further improving memory access efficiency by selectively caching only their key-value pairs. Beyond these core mechanisms, we also propose a KV sharing variant that reuses KV pairs from the first recursion, specifically designed to further decrease memory footprint. Across model scales ranging from 135M to 1.7B parameters, MoR forms a new Pareto frontier: at equal training FLOPs and smaller model sizes, it significantly lowers validation perplexity and improves few-shot accuracy, while delivering higher throughput compared with vanilla and existing recursive baselines.

1 Introduction

Scaling Transformer networks to hundreds of billions of parameters has unlocked impressive few-shot generalization and reasoning abilities [10, 14, 19, 30, 31, 65, 71]. However, the accompanying memory footprint and computational requirements make both training and deployment outside hyperscale data centers challenging [68, 74]. This has motivated researchers to seek alternative “efficient” designs [89, 92]. Among the different axes of efficiency, *parameter efficiency* [6, 20, 24, 55, 83]—reducing or sharing model weights—and *adaptive computation* [24, 57, 79, 81]—spending more compute only when it is needed—are promising, actively studied research directions.

One proven route to parameter efficiency is *layer tying*, in which a shared set of weights is reused across multiple layers [6, 20, 32, 54, 87]. For adaptive computation, a common approach is *early-exiting*, which dynamically allocates compute by exiting earlier in the network when predicting simpler tokens [5, 21, 22, 81]. Despite the progress achieved along each of these individual efficiency axes, an architecture that effectively unifies both parameter efficiency and adaptive computation is still missing. Recursive Transformers [3, 6, 23, 29, 33, 80, 100], models that repeatedly apply the same set of shared layers multiple times, offer a strong foundation due to their built-in weight sharing. However, prior attempts at dynamic recursion have often been constrained by practical hurdles, such as requiring additional specialized training procedures or facing challenges in efficient

*Equal contribution. †Corresponding authors. The Google co-authors performed only an advisory role.

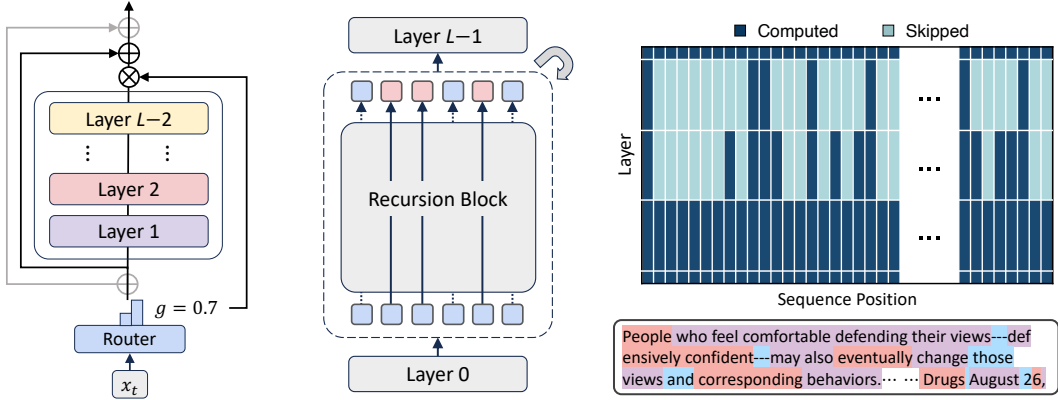


Figure 1: Overview of Mixture-of-Recursions (MoR). (Left) Each recursion step consists of a fixed stack of layers and a router that determines whether each token should pass through or exit. This recursion block corresponds to the gray box in the middle. (Middle) The full model structure, where the shared recursion step is applied up to N_r times for each token depending on the router decision. (Right) An example routing pattern showing token-wise recursion depth, where darker cells indicate active computation through the recursion block. Below shows the number of recursion steps that each subword token undergoes to predict the *next* token, shown in colors: 1, 2, and 3.

deployment. This has led most approaches to still default to a simpler fixed-depth recursion, which applies the same amount of computation to every token and is thus incapable of delivering truly adaptive token-level compute allocation.

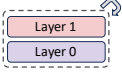
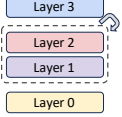
In this work, we introduce *Mixture-of-Recursions* (MoR), a unified framework that fully leverages the potential of Recursive Transformers (see Figure 1). MoR trains lightweight routers end-to-end to assign token-specific recursion depths: it decides how many times a shared parameter block is applied to each token according to its required depth of “thinking”, thereby directing computation to where it is most needed. This dynamic, token-level recursion inherently facilitates recursion-wise key-value (KV) caching, selectively storing and retrieving key-value pairs corresponding to each token’s assigned recursion depth. This targeted caching strategy reduces memory traffic, thereby improving throughput without relying on post-hoc modifications. Therefore, MoR simultaneously (i) ties weights to cut parameters, (ii) routes tokens to cut redundant FLOPs, and (iii) caches key-values recursion-wise to cut memory traffic—all inside a single architecture.

Conceptually, MoR provides a *pre-training* framework for latent space reasoning—performing non-verbal thinking by iteratively applying a single parameter block [29, 34, 38]. However, unlike approaches that deliberate on augmented continuous prompts before generation [34, 38, 60, 84], MoR enables this latent thinking directly during the decoding of each token [102]. Furthermore, the routing mechanism facilitates adaptive reasoning along the model’s vertical axis, moving beyond the uniform, fixed thinking depth common in prior work [29, 86]. In essence, MoR enables models to efficiently adjust their thinking depth on a per-token basis, unifying parameter efficiency with adaptive computation.

Contributions. In summary, our key contributions in this paper are as follows.

- **Unified framework for efficient language modeling:** We present *Mixture-of-Recursions* (MoR), the first architecture to unify efficiency paradigms—parameter sharing (§2.1), token-level adaptive thinking depth (§2.2.1), and memory-efficient KV caching (§2.2.2)—within a single framework.
- **Dynamic recursion routing:** We introduce a router trained from scratch to assign dynamic per-token recursion depths. This aligns training with inference-time behavior and eliminates the need for costly, performance-degrading post-hoc routing stages used in conventional early-exit methods.
- **Extensive empirical validation:** Across models from 135M to 1.7B parameters under equal compute budgets, MoR establishes a new Pareto frontier by improving validation loss and few-shot accuracy relative to vanilla and recursive baselines (§3.1, §3.2).
- **Efficient architecture:** MoR dramatically reduces training FLOPs by selectively engaging only essential sequences in attention operations. Simultaneously, reduction in KV cache sizes leads to enhanced inference throughput itself, further boosted by continuous depth-wise batching (§3.3).

Table 1: Parameter-sharing strategies in Recursive Transformers. This table shows *Cycle* and *Middle-Cycle* schemes with cyclic layer reuse, where Middle-Cycle retains unique first and last layers.

Layers	Cycle Strategy		Middle-Cycle Strategy	
	Equation	Figure	Equation	Figure
Last	–		$f(\mathbf{h}_t^{L-1}; \Phi_{L-1})$	
Recursion	$f(\mathbf{h}_t^\ell; \Phi'_{\ell \bmod (L/N_r)})$		$f(\mathbf{h}_t^\ell; \Phi'_{(\ell-1 \bmod ((L-2)/N_r))+1})$	
First	–		$f(\mathbf{h}_t^0; \Phi_0)$	

2 Method

2.1 Preliminary

Recursive Transformers. The standard Transformer [91] constructs token representations through a stack of L unique layers, each with a self-attention and a feed-forward network. At time step t , the hidden state h evolves as: $\mathbf{h}_t^{\ell+1} = f(\mathbf{h}_t^\ell; \Phi_\ell)$, where $\ell = 0, \dots, L-1$ and Φ_ℓ represents the parameters of the ℓ -th layer. Recursive Transformers [6, 23, 33, 80, 100] aim to reduce parameter count by reusing layers across depth. Instead of having L distinct sets of weights, they partition the model into N_r recursion *blocks*, where each block uses a shared pool of parameters Φ' . This design allows for more computation (by increasing the effective network depth) without increasing parameter size.

Parameter-sharing strategies. We examine four parameter-sharing strategies: *Cycle*, *Sequence*, and their variants *Middle-Cycle* and *Middle-Sequence*. Table 1 summarizes two main designs, and the full list is provided in Table 5 in the Appendix. In Cycle sharing, recursion blocks are reused cyclically. For example, consider an original non-recursive model with $L=9$ layers and its recursive counterpart using $N_r=3$ recursions. Under the “Cycle” strategy, the layers are shared and unrolled as $[(0, 1, 2), (0, 1, 2), (0, 1, 2)]$. In “Sequence” sharing, each recursion block reuses the same layer consecutively before moving to the next, resulting in $[(0, 0, 0), (1, 1, 1), (2, 2, 2)]$ for the same configuration. Both have the same effective number of layers when unrolled ($L=9$), but with a different order. Furthermore, the “Middle” variants preserve full-capacity parameters at the first and last layers (Φ_0 and Φ_{L-1}), while sharing weights among the intermediate layers.

Enhanced training and inference efficiency in recursive models. Parameter sharing strategies can reduce the number of unique trainable parameters by a factor of the recursion number, effectively amortizing the memory footprint of the model. From a distributed training perspective, this becomes highly efficient when using Fully Sharded Data Parallel (FSDP) [108]. While a single all-gather operation would only support one iteration previously (i.e., 1 iter/gather), a recursive model reuses the same gathered parameters across all recursive steps (i.e., N_r iter/gather). Furthermore, recursive architectures enable a novel inference paradigm, continuous depth-wise batching [6, 39]. This technique allows tokens at different stages to be grouped into a single batch, as they all utilize the same block of parameters. This can eliminate the bubbles—idle periods spent waiting for other samples to complete—thereby leading to significant throughput gains.

Limitations in prior works. Although model parameters are tied, the distinct KV caches are typically used for each depth. This design fails to reduce the cache sizes, meaning the high retrieval latency still remains a severe inference bottleneck. Moreover, most existing recursive models simply apply a fixed recursion depth to all tokens, ignoring the varying complexity. While post-hoc methods like early-exiting methods can introduce some adaptivity, they often require separate training phases that can degrade performance [6, 22, 81]. Ideally, the recursion depth should be learned dynamically during pretraining, allowing the model to adapt its computational path to each token’s difficulty in a data-driven manner. However, such dynamic paths introduce a new challenge: exited tokens will have missing KV pairs at subsequent recursion depths. Addressing this would require a parallel decoding mechanism [5, 22, 51] to efficiently compute the actual KV pairs, but this requires separate, complex engineering and complicates the system.

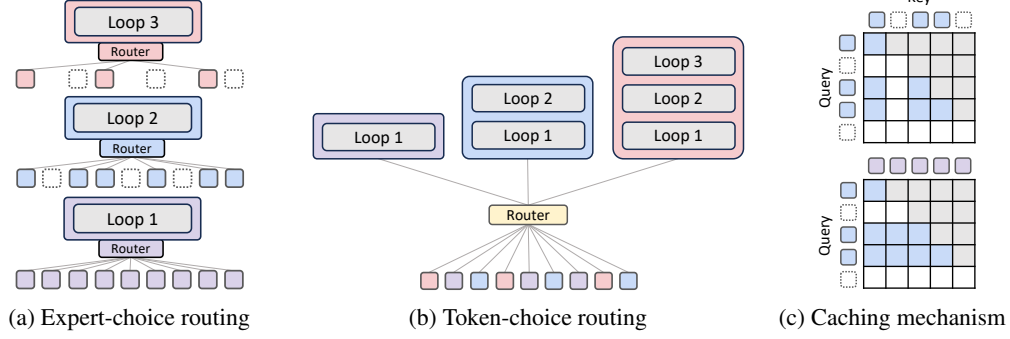


Figure 2: Architectural components of Mixture-of-Recursions (MoR). (a) Expert-choice routing: At each recursion step, a router selects top- k tokens to continue, progressively narrowing the set of active tokens with depth. (b) Token-choice routing: Each token is assigned a fixed recursion step at the outset via a single routing decision, defining its complete compute path through the model. (c) KV caching strategies: Each square in the matrix represents whether a token (row) attends to another token’s cached key (column). In “recursion-wise KV caching” (Top), only the keys of currently selected (non-dropped) tokens at each recursion step are cached (blue), and attention is restricted only to these entries. In “recursive KV sharing” (Bottom), all keys of previous tokens are cached at the first recursion step (purple) and shared across subsequent recursion steps for attention operations.

2.2 Mixture-of-Recursions

We propose *Mixture-of-Recursions* (MoR)—a framework that dynamically adjusts recursion step for each token during *pretraining* and *inference*. The core of MoR lies in two components: a routing mechanism that assigns token-specific recursion steps to adaptively concentrate computation on more challenging tokens, and a KV caching strategy that defines how KV pairs are stored and selectively utilized for attention at each recursive step.

2.2.1 Routing Strategies: Expert-choice vs. Token-choice

Expert-choice routing. (Figure 2a) Inspired by top- k gating in MoD models [79], in expert-choice routing, each recursion depth becomes an expert and selects their preferred top- k tokens (e.g., for $N_r = 3$, we have three experts: Expert 1 applies the first recursion step, Expert 2 applies the second recursion step, and so on). At each recursion step r , the corresponding router uses the hidden state $\mathcal{H}_t^r$ (input to the r -th recursion block) and its routing parameters θ_r to compute a scalar score $g_t^r = \mathcal{G}(\theta_r^\top \mathcal{H}_t^r)$ for token t . Here, $\mathcal{G}$ represents an activation function like sigmoid or tanh. Then, the top- k tokens are selected to pass through the recursion block:

$$\mathcal{H}_t^{r+1} = \begin{cases} g_t^r f(\mathcal{H}_t^r, \Phi') + \mathcal{H}_t^r, & \text{if } g_t^r > P_\beta(G^r) \\ \mathcal{H}_t^r, & \text{otherwise} \end{cases} \quad (1)$$

where $P_\beta(G^r)$ is the β -percentile threshold over all scores at recursion step r .

To ensure coherent progression through steps, we adopt *hierarchical filtering*: only tokens selected at recursion step r can be re-evaluated at $r+1$. This simulates early-exit behavior while learning from scratch. As deeper layers tend to encode increasingly abstract and sparse information [58, 69, 99], this mechanism prioritizes computation for only the most demanding tokens.

Token-choice routing. (Figure 2b) Unlike expert-choice, where token selection is made at each recursion step, token-choice commits each token to a full sequence of recursion blocks from the start. Formally, given the hidden state $\mathcal{H}_t^1$ (in Middle-Cycle strategy, $\mathcal{H}_t^1 = h_t^1$), the router computes a non-linear function (softmax or sigmoid) over experts: $g_t = \mathcal{G}(\theta_r^\top \mathcal{H}_t^1)$, where g_t^j denotes the routing score for expert $j \in \{1, \dots, N_r\}$. The token is assigned to expert $i = \arg \max_j g_t^j$ (top-1 gating), which corresponds to sequentially applying the recursion i times. The hidden state is then updated recursively as:

$$\mathcal{H}_t^{r+1} = \begin{cases} g_t^r f(\mathcal{H}_t^r, \Phi') + \mathcal{H}_t^r, & \text{if } r = i \\ g_t^r f(\mathcal{H}_t^r, \Phi'), & \text{otherwise} \end{cases} \quad (2)$$

Table 2: Comparison of routing strategies and key-value caching strategies. (Left) Summary of two routing strategies: expert-choice and token-choice, highlighting their pros, cons, and mitigating solutions from previous works [79, 93, 111]. (Right) Relative cost efficiency of caching strategies against a vanilla Transformer (normalized to 1). Here, N_r denotes the number of recursions, and k ($k < N_{\text{ctx}}$) denotes the number of selected tokens per layer. We compare only the recursion blocks, excluding the non-shared layers. KV cache memory and IO are measured across the entire model, whereas attention FLOPs are reported per layer.

	Expert-choice	Token-choice		Recursion-wise Caching	Recursive Sharing
Pros	Static compute budget	No leakage	KV Memory	$(N_r + 1)/2N_r$	$1/N_r$
Cons	Causality violation	Load imbalance	KV Cache IO	$(N_r + 1)/2N_r$	1
⊥ Sol	Aux Rout, Aux Loss	Bal Loss, Loss-free	Attn FLOPs	k^2/N_{ctx}^2	k/N_{ctx}

To compare routing strategies under equal compute, we align the token allocation budgets of expert-choice with that of token-choice. Specifically, we calibrate token capacity (i.e., top- k) of expert-choice to match the expected token distribution of token-choice routing with perfect load balancing. In perfectly balanced token-choice, each token is assigned to recursion depth $i \in \{1, \dots, N_r\}$ with equal probability $1/N_r$. Thus, recursion step j processes a fraction $(N_r - j + 1)/N_r$ of the tokens. For example, when $N_r = 3$, recursion steps 1, 2, and 3 handle $\{3/3, 2/3, 1/3\}$ of tokens, respectively. Therefore, we apply this same fractional allocation in the top- k selection of the expert-choice routing (i.e., k is sequenced like $N_r/N_r, \dots, 1/N_r$ over N_r recursion steps).

Strengths and limitations. (Table 2–Left) Although expert-choice routing guarantees perfect load balancing with static top- k selection, it suffers from information leakage [79, 93, 110]. This violation of causality during training forces to exploit an auxiliary router or a regularization loss [79, 110], aiming to precisely detect top- k tokens at inference without access to future token information. Meanwhile, token-choice is free from such leakage, but typically requires a balancing loss or loss-free algorithms [24, 93, 111] due to its inherent load balancing challenges. We explore each of these components for MoR in further detail (§4.2).

2.2.2 KV Caching Strategies: Recursion-wise Caching vs. Recursive Sharing

Dynamic-depth models often struggle with KV cache consistency during autoregressive decoding. When a token exits early, its corresponding keys and values in deeper layers will be missing, which can be crucial information for subsequent tokens. Some methods attempt to reuse stale entries [81] or run parallel decoding [5], but these solutions still introduce overhead and complexity. To this end, we design and explore two KV cache strategies tailored to MoR models: *recursion-wise caching* and *recursive sharing*.

Recursion-wise KV caching. (Figure 2c–Top) Inspired by Raposo et al. [79], we cache KV pairs selectively: only tokens routed to a given recursion step store their key-value entries at that level. Thereby, the KV cache size at each recursion depth is determined exactly by the capacity factor in expert-choice, or according to actual balancing ratios in token-choice. Attention is then restricted to those locally cached tokens. This design promotes block-local computation, which improves memory efficiency and reduces IO demands.

Recursive KV sharing. (Figure 2c–Bottom) A key design choice for our MoR model is that all tokens traverse at least the first recursion block². We leverage this by caching KV pairs exclusively at this initial step and reusing them across all subsequent recursions. Therefore, the query length might get shorter at each recursion depth based on the selection capacity, but the key and value lengths will consistently maintain the full sequence. This ensures that all tokens can access to past context without recomputation, despite any distribution mismatch.

Strengths and limitations. (Table 2–Right) Recursion-wise caching cuts KV memory and IO to approximately $(N_r + 1)/2N_r$ times across the entire model (when assuming capacity factors follow a sequence like $N_r/N_r, \dots, 1/N_r$ over N_r recursion steps). It also reduces per-layer attention FLOPs to a factor of $(k/N_{\text{ctx}})^2$ of those in vanilla models, resulting in substantially improved efficiency for both training and inference phases. Meanwhile, recursive sharing can yield maximal memory savings by globally reusing context. Speedups can be further achieved by skipping KV projection and prefill at shared depths (compatible only with Cycle strategy [85]). However, attention FLOPs only decrease by k/N_{ctx} , and high volume of KV IO still leads to a decoding bottleneck.

²Though this is not a strict requirement of the MoR framework itself.

Table 3: Comparison of MoR, Recursive, and Vanilla Transformers under both fixed FLOPs (16.5e18) and token (20B) settings. All models are trained on FineWeb-Edu and evaluated by validation negative log-likelihood (NLL) and few-shot accuracy. For the isoFLOP rows, the number of training tokens (N_{tok}) varies by model efficiency. For the fixed-token rows, we report the effective FLOPs consumed. For the model sizes, we report non-embedding parameter counts. For the KV mechanisms, we distinguish between Cache (recursion-wise caching) and Share (recursive sharing). [†]In recursive models, all tokens go through fixed recursion depths (N_r), instead of adaptive depths.

	MoR		Recursion		Pretrain			NLL ↓	Few-shot Accuracy ↑						
Models	Type	KV	Share	N_r	Param	FLOPs	N_{tok}	FineWeb	LD	HS	PQ	WG	ARC	MLLU	Avg
Vanilla	-	-	-	-	315M	16.5	20B	2.7824	32.0	37.8	65.6	50.5	39.6	28.0	42.3
Recursive [†]	-	-	M-Cyc	2	167M	16.5	20B	2.8079	31.0	37.1	66.7	52.3	40.8	27.5	42.6
	-	-	M-Cyc	3	118M	16.5	20B	2.8466	29.8	35.9	65.0	52.3	39.0	27.2	41.5
	-	-	M-Cyc	4	98M	16.5	19B	2.8781	28.2	35.4	65.5	52.5	38.0	26.8	41.0
MoR (ours)	Expert	Cache	M-Cyc	2	167M	16.5	27B	2.7511	34.4	39.3	65.7	51.2	39.6	28.1	43.1
	Expert	Cache	M-Cyc	3	118M	16.5	30B	2.7925	33.1	37.9	66.9	52.1	38.3	27.4	42.6
	Expert	Cache	M-Cyc	4	98M	16.5	30B	2.8204	30.1	37.3	65.0	51.1	38.9	27.4	41.6
	Expert	Cache	M-Cyc	2	167M	12.3	20B	2.7749	33.2	38.3	65.2	52.6	40.1	28.1	42.9
	Expert	Cache	M-Cyc	3	118M	11.0	20B	2.8246	31.9	37.0	65.7	50.5	38.3	27.4	41.8
	Expert	Cache	M-Cyc	4	98M	11.0	20B	2.8519	30.2	36.5	64.3	52.3	38.6	27.2	41.5
	Token	Cache	M-Cyc	3	118M	16.5	30B	2.9163	27.6	34.1	63.8	50.6	37.4	26.8	40.0
	Expert	Share	M-Cyc	3	118M	16.5	31B	2.7983	31.7	37.2	65.1	51.0	39.0	27.1	41.9

3 Experiments

We pretrain our models from scratch using a Llama-based Transformer architecture³ [65], referring to the configurations of SmolLM open-source models [4], on a deduplicated subset of the FineWeb-Edu dataset [75] in SmolLM-Corpus [7]. We evaluate the models on validation set of FineWeb-edu and six few-shot benchmarks [26]. Detailed training and evaluation procedures, as well as throughput measurement protocols, are described in Appendix D.

3.1 Main Results

MoR outperforms baselines with fewer parameters under equal train compute. Under an equal training budget of 16.5e18 FLOPs, we compared our Mixture-of-Recurions (MoR) model against both Vanilla and Recursive Transformers. As shown in Table 3, the MoR model, using an expert-choice router and two recursions, achieves a lower validation loss and surpasses the vanilla baseline in average few-shot accuracy (43.1% vs. 42.3%). Remarkably, this superior performance is achieved despite using nearly 50% fewer parameters. This is attributed to MoR’s higher computational efficiency, which allows it to process more training tokens within the same FLOPs budget. Furthermore, as N_r increases to 3 or 4, MoR maintains its competitive accuracy, consistently outperforming the recursive baselines while remaining within a tight margin of the full-capacity vanilla model.

MoR outperforms baselines with less compute at equal data. To isolate architectural differences, we analyze performance under a fixed number of training tokens (20B). Specifically, our MoR model with $N_r = 2$ outperforms both vanilla and recursive baselines—achieving lower validation loss and higher accuracy—despite using 25% fewer training FLOPs. This theoretical efficiency translates into significant practical gains: compared to the vanilla baseline, our model reduces training time by 19% and cuts peak memory usage by 25%. These improvements stem from our hierarchical filtering and recursion-wise attention mechanism, which shortens sequence lengths to achieve a superior compute-accuracy trade-off, even during pretraining.

MoR performance varies with routing and caching strategies. We also evaluate a few design variants within MoR framework, specifically with $N_r = 3$ that is lightweight and still comparable with Vanilla. In this case, using token-choice routing yields lower performance (40.0%) compared to expert-choice routing (42.6%), indicating that routing granularity plays a pivotal role in model performance. Additionally, applying KV cache sharing slightly reduces performance compared to independent caching, while providing improved memory efficiency. This trade-off remains favorable for practical deployment when memory usage is a key concern.

³Experiments on Llama are conducted without direction or involvement from Google advisors.

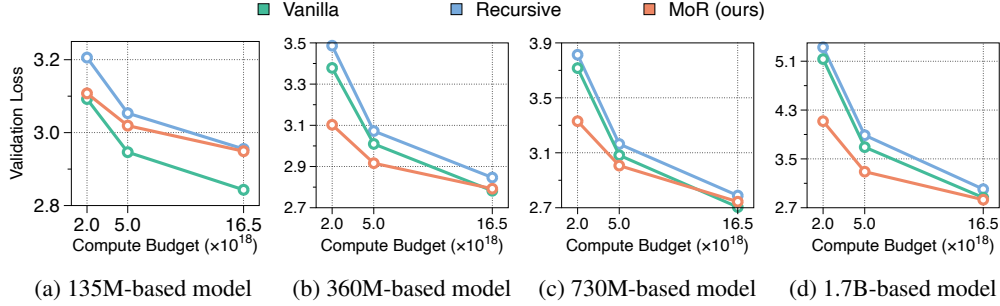


Figure 3: Validation loss across different compute budgets across four model sizes: 135M, 360M, 730M, and 1.7B parameters. For MoR models, we use expert-choice routing and recursion-wise caching. MoR consistently outperforms recursive baselines and matches or exceeds the standard Transformers at larger scales, despite using significantly fewer parameters (approximately one-third due to layer tying with $N_R = 3$).

3.2 IsoFLOP Analysis

A core criterion for evaluating a new model architectural design is whether performance continues to improve as model and compute scales grow [49]. Therefore, we evaluate MoR against both Vanilla and Recursive Transformers across a wide range of model sizes and computational budgets to show that it maintains competitive or superior predictive performance as the scale increases.

Experimental Setup. We experiment with four scales—135M, 360M, 730M, and 1.7B parameters—fixing the number of recursions to three for both Recursive and MoR configurations, resulting in roughly one-third the number of unique parameters. Each model is pretrained under three FLOPs budgets: 2e18, 5e18, and 16.5e18.

MoR is a scalable and parameter-efficient architecture. As shown in Figure 3, MoR consistently outperforms recursive baselines across all model sizes and compute budgets. While it underperforms the vanilla model at the smallest model size (135M)—likely due to a recursive capacity bottleneck—this gap closes rapidly at scale. For >360M parameters, MoR not only matches but often exceeds the Vanilla Transformer, particularly under low and mid-range budgets. Overall, these results highlight that MoR is a scalable and efficient alternative to standard Transformers. It achieves strong validation performance with significantly lower parameter counts, making it a strong candidate for both pretraining and large-scale deployment. Further details are presented in Appendix F.

3.3 Inference Throughput Evaluation

As a parameter-shared architecture, MoR can leverage continuous depth-wise batching [6] to dramatically boost inference throughput compared to Vanilla Transformers. This maintains high and consistent GPU utilization by immediately replacing completed sequences with incoming tokens during decoding. The early-exiting mechanism in MoR further eliminates bubbles in the computational batch.

Experimental Setup. We measure throughput for 360M scale-based MoR models with recursion depths of 2, 3, and 4, trained under a 16.5e18 FLOPs budget. Throughput (tokens/second) is measured based on the generation time for tokens per sample, where the number of tokens is sampled from a normal distribution with a mean of 256, starting without any input prefix. We examine two batching configurations: a fixed batch size of 32 and a (relative) maximum batch size derived by multiplying 32 by the ratio of the maximum batch sizes of vanilla and MoR models. Further details on the experimental setup are provided in Appendix G.

MoR boosts inference throughput with continuous depth-wise batching. In Figure 4a, across both batch settings, all MoR variants outperform the vanilla baseline, which even leverages continuous sequence-wise batching [53, 101]. Increasing recursion depth leads to more tokens exiting early and a further reduction in KV cache usage. This, in turn, boosts throughput significantly (e.g., MoR-4 achieves up to a $2.06\times$ speedup with $B = \text{Max}$). While there’s a slight performance degradation, it can be a favorable trade-off given the substantial throughput gain. These results support that the integration of the depth-wise batching paradigm with early-exiting can significantly accelerate MoR’s actual deployment throughput.

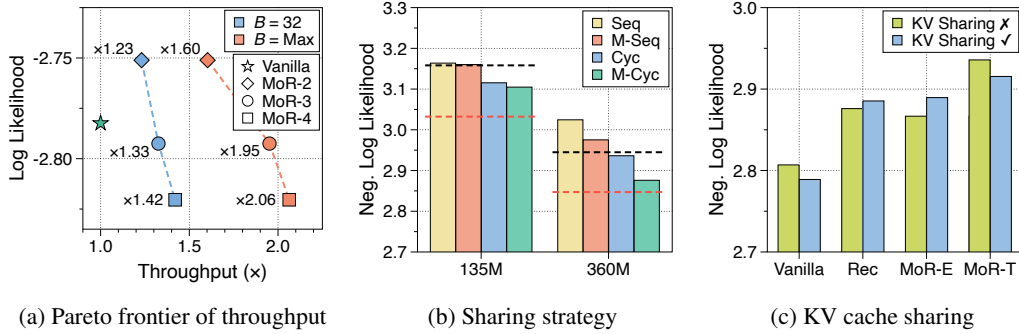


Figure 4: (a) Pareto frontier of inference throughput and log-likelihood for MoR and Vanilla Transformer under fixed and maximum batching scenarios. Setting details are in [Appendix G](#). (b) Negative log-likelihood (NLL) of Recursive Transformers with $N_r = 3$ across four different parameter-sharing strategies. We pretrained the models on 10 billion tokens. The dashed red and black lines denote the full-size Vanilla Transformer and parameter-matched vanilla models (approximately one-third scales), respectively. (c) NLL performance comparison across four different architectures with KV sharing. For MoR, green (disabled) and blue (enabled) refer to recursion-wise KV caching and recursive KV sharing strategies. MoR-E and MoR-T denotes expert-choice and token-choice MoR, respectively. All models are based on 360M scale and trained on 10 billion tokens.

4 Ablation Studies

4.1 Parameter Sharing Strategies

Middle-Cycle is the most effective parameter sharing strategy. As discussed in [§2.1](#), parameter sharing is a key component of Recursive Transformers and MoR. To identify the most effective sharing configuration, we empirically compare four aforementioned strategies: Cycle, Sequence, Middle-Cycle, and Middle-Sequence. We evaluate each strategy on Recursive Transformers based on 135M and 360M model sizes. As shown in [Figure 4b](#), “Middle-Cycle” consistently achieves the lowest validation loss, and its superiority is further confirmed by the detailed results in [Appendix H](#). Based on these findings, we adopt the “Middle-Cycle” configuration for all subsequent MoR and Recursive Transformers presented in this paper.

4.2 Routing Strategies

We conduct an extensive ablation study to understand the impact of various design choices for routing schemes in our MoR framework. Detailed results are summarized in [Appendix I](#).

In expert-choice routing, auxiliary loss and linear router yield the best performance. For the *expert-choice* routing setup (*Left* of [Table 4](#)), we evaluate several design aspects: solution to mitigate causality violation (auxiliary router vs. auxiliary loss), normalization functions (sigmoid vs. tanh), router architectures (MLP, Linear, or Wide-MLP), and the impact of an auxiliary z-loss [111]. To assess how well the router performs dynamic allocation, we measure the proportion of “dead” tokens—those never selected by the final recursion in a batch—on the validation dataset. Our key findings are as follows: First, using an auxiliary loss is more effective for inference-time behavior than training a separate auxiliary router. Second, a sigmoid normalization function and a simple linear router architecture yield the best performance. Finally, the auxiliary z-loss has a negligible impact on accuracy, though it does slightly reduce the proportion of dead tokens.

In token-choice routing, balancing loss yields stable and accurate routing. For *token-choice* routing (*Right* of [Table 4](#)), we follow common MoE practices and enable z-loss by default. We compare two balancing strategies: using a balancing loss and training in a loss-free manner using router bias. While both approaches achieve similar log-probability and few-shot accuracy, the explicit balancing loss yields a significantly lower MaxVio [93] in our MoR architectures, making it the preferable choice for stable routing. However, despite this, the model often struggles to balance loads among its heterogeneous experts, even for nearly half of the training steps. Softmax activation with an MLP router performs best, and removing z-loss—though we add back with a very small coefficient in the final design—results in higher performance and routing stability.

Table 4: Ablation results for expert-choice (*Left*) and token-choice (*Right*) routers with various design choices. We use MoR models that apply three recursions to a 360M model with recursion-wise caching. Model performance is measured by NLL and average few-shot accuracy. We evaluate router metrics—dead token ratio (for expert-choice) and MaxVio (for token-choice)—on the validation set. The dead token ratio denotes the proportion of tokens that remain unselected during the last recursion step, measured on 500 samples, each with 2K sequence length. The selected design choice is highlighted in gray.

Expert-choice Router				Performance (↓ / ↓ / ↑)		
Sampling	Func	Arch	z-loss	Dead	NLL	Few-shot
Aux Rout	σ	MLP	✗	0.0	2.8893	39.4
Aux Rout	$\tanh$	MLP	✗	66.7	2.8720	36.2
Aux Loss	σ	MLP	✗	0.0	2.8816	40.0
Aux Loss	$\tanh$	MLP	✗	0.0	2.9933	38.8
Aux Loss	σ	Linear	✗	0.1	2.8667	40.1
Aux Loss	σ	W-MLP	✗	0.4	2.8716	39.4
Aux Loss	σ	Linear	✓	0.0	2.8824	40.0

Token-choice Router				Performance (↓ / ↓ / ↑)		
Balancing	Func	Arch	z-loss	M-Vio	NLL	Few-shot
Loss (0.1)	soft	MLP	✓	0.200	3.0239	38.5
Loss (0.01)	soft	MLP	✓	0.682	2.9118	39.4
Loss-free	soft	MLP	✓	0.852	2.9081	39.4
Loss-free	σ	MLP	✓	1.281	3.0188	37.6
Loss (0.1)	soft	Linear	✓	0.492	2.9974	38.4
Loss (0.1)	soft	W-MLP	✓	0.384	3.0293	38.8
Loss (0.1)	soft	Linear	✗	0.266	2.9358	39.1

4.3 KV Caching Strategies

KV sharing robustly works even in parameter-shared architectures. In Figure 4c, we first investigate the effect of KV sharing in Vanilla and Recursive Transformers. As consistent with prior works [9, 85, 94], if we pretrain models from the scratch, KV sharing does not often compromise performance due to the greater parameter flexibility. Surprisingly, the Recursive Transformer remains relatively robust to KV sharing, despite its reduced degrees of freedom. We found evidence for this by decomposing the KV pairs at each recursion depth into their magnitude and direction. As detailed in Appendix J, depths that share parameters exhibit highly consistent magnitude patterns and high cosine similarity, providing a clear justification for why KV sharing results in only a slight performance drop.

KV sharing degrades expert-choice but benefits token-choice routing in MoR. We compare recursion-wise KV caching and recursive KV sharing mechanisms in our MoR framework. We observe that while recursive KV sharing offers the advantages of reduced memory footprint and overall FLOPs⁴, it leads to quite large performance degradation in expert-choice routing under a fixed token setting. This suggest that exclusively updating and attending to the tokens active in that recursion depth may be more beneficial. Conversely, MoR with token-choice routing could benefit from KV sharing, where its weaker, inaccurate routing decisions can be complemented by the additional contextual information provided by shared KV pairs.

5 Analysis

5.1 Compute-optimal Scaling Analysis

MoR scaling favors model size over training length under isoFLOPs. As illustrated in Figure 5a, MoR exhibits a distinct compute-optimal scaling behavior compared to baselines under isoFLOPs constraints. The flatter slope of MoR’s optimal path (a line connecting stars) indicates that it benefits more significantly from increases in parameter count (i.e., less data-hungry). This is likely because the performance of the shared parameter block itself becomes important, even more than feeding in additional data. Therefore, the optimal scaling policy for MoR models favors allocating resources to increasing model capacity by using larger models trained for shorter steps.

5.2 Routing Analysis

The allocation of recursion depth reflects contextual predictability of the subsequent token. In the *Right* of Figure 1, we illustrate each token’s recursion depth, which directly indicates how easily the next token can be predicted within the given context. For example, the second subword

⁴Although attention FLOPs increase by N_{ctx}/k than recursion-wise KV caching, reduced KV projection FLOPs lead to an overall reduction.

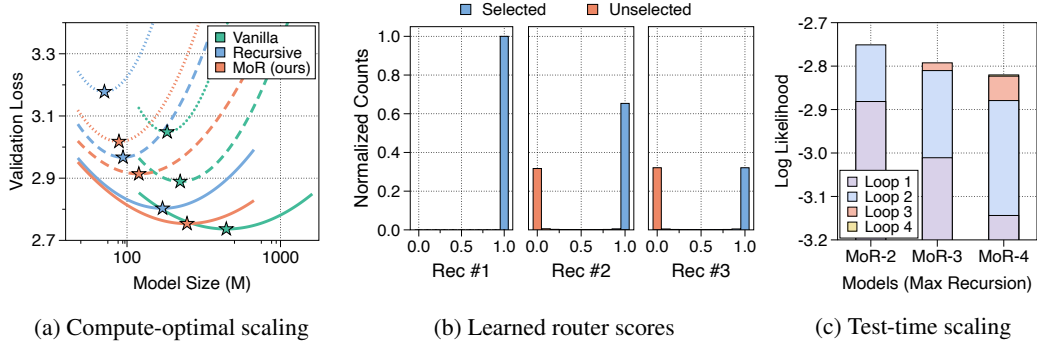


Figure 5: (a) Compute-optimal scaling analysis for three model architectures. Each star indicates the optimal model size for a given compute budget. We visualize the results in §3.2 by fitting polynomial functions for each architecture and FLOPs budget, and derive the optimal points from these fits. (b) Distribution of router outputs for selected and unselected tokens at each recursion step. As an example, a 360M size-based MoR model with $N_r = 3$, expert-choice router with auxiliary loss, and recursion-wise caching, is used. (c) Test-time scaling analysis illustrating the cumulative log-likelihood improvement with increasing recursion depth, measured over 500 samples. As we increase N_r based on a 360M model size, the number of unique parameters in MoR decreases, resulting in a gradual decline in overall performance (i.e., a decrease in log-likelihood). All models are trained by an expert-choice router with auxiliary loss and a recursion-wise caching mechanism.

part (e.g., “-ensively”) of a word is often straightforward to predict, thus needing fewer steps. As also shown in Table 17, while the initial generation (opening) of function words like “---”, “(”, “,”, and “,” appears easy for models, predicting their closing parts or the first token immediately following their opening might be more challenging.

Expert-choice router with auxiliary loss perfectly separates selected from unselected tokens.

Figure 5b visualizes one example of the expert-choice router output distribution at each recursion step in a MoR model with $N_r = 3$. For each recursion step, the normalized counts of routing scores are plotted, distinguishing between tokens selected by the expert (blue) and those not selected (orange). In all steps, auxiliary loss achieves a perfect separation in router outputs, with selected tokens sharply concentrated near a routing score of 1.0 and unselected tokens clustering near 0.0. Router output distributions for the two routing strategies and their associated design choices are detailed in Appendix K.

5.3 Test-time Scaling Analysis

MoR enables test-time scaling via deeper recursion. We visualize how log-likelihood evolves across recursion steps in MoR models with $N_r = \{2, 3, 4\}$ in Figure 5c. The overlaid bars illustrate the performance of each model when the maximum thinking (recursion) depth of tokens gradually increases. This suggests that deeper recursion not only provides additional compute but also enables each subsequent step to specialize further in refining the token representation or “thought process” at its particular depth, leading to better performance. Thereby, these results support the view that MoR enables test-time scaling: allocating more recursion steps at inference can improve generation quality.

6 Conclusion

Mixture-of-RecurSIONs (MoR) presents a unified Transformer architecture that simultaneously leverages parameter sharing, adaptive recursion depth, and efficient KV caching without compromising model quality. By dynamically assigning recursion depth to tokens via lightweight routers and selectively caching key-value states for selected tokens, MoR reduces both quadratic attention computation and redundant memory access costs. Extensive empirical evaluations show that MoR lowers validation perplexity and improves average few-shot accuracy compared to both vanilla and previous recursive baselines, even with higher inference throughput. These results demonstrate that MoR offers an effective path towards achieving large-model capabilities with significantly reduced computational and memory overhead.

Acknowledgements

We thank Jacob Eisenstein for valuable feedback on an earlier version of the paper. We also acknowledge the support and helpful conversations from Seungyeon Kim, Mostafa Elhoushi, Pascal Vincent, Irina Rish, Sangdoo Yun, and Baeseong Park. Finally, we thank the Google Cloud Platform for awarding Google Cloud credits for this project, and Google’s research grant project for its support.

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) ([RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST), 5%], [No. RS-2024-00457882, AI Research Hub Project, 5%], and [No. 2022-0-00871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration, 90%]). This research was also enabled in part by computational resources, software, and technical assistance provided by [Mila](#), [ULaval](#), [Calcul Québec](#), and [Digital Research Alliance of Canada](#).

References

- [1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.298.
- [2] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.
- [3] Preslav Aleksandrov, Meghdad Kurmanji, Fernando Garcia Redondo, David O’Shea, William Shen, Alex Iacob, Lorenzo Sani, Xinchu Qiu, Nicola Cancedda, and Nicholas D Lane. Abbie: Autoregressive block-based iterative encoder for efficient sequence modeling. *arXiv preprint arXiv:2507.08567*, 2025.
- [4] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. Smollm - blazingly fast and remarkably powerful, 2024.
- [5] Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5910–5924, Singapore, dec 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.362. URL <https://aclanthology.org/2023.emnlp-main.362>.
- [6] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. *arXiv preprint arXiv:2410.20672*, 2024.
- [7] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm-corpus, 2024. URL <https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus>.
- [8] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.
- [9] William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. *Neural Information Processing Systems*, 2024. doi: 10.48550/arXiv.2405.12981.
- [10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

- [11] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv: 2302.01318*, 2023.
- [12] Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuo-huan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. *arXiv preprint arXiv:2502.13842*, 2025.
- [13] Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.
- [14] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023.
- [15] Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7085–7095, 2022.
- [16] Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *arXiv preprint arXiv:2505.07686*, 2025.
- [17] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.
- [18] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.
- [19] DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv: 2412.19437*, 2024.
- [20] Mostafa Dehghani, Stephan Gouws, O. Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. *International Conference on Learning Representations*, 2018.
- [21] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=SJg7KhVKPH>.
- [22] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.
- [23] Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.
- [24] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.
- [25] Advait Gadhikar, Souptik Kumar Majumdar, Niclas Popp, Piyapat Saranrittichai, Martin Rapp, and Lukas Schott. Attention is all you need for mixture-of-depths routing. *arXiv preprint arXiv:2412.20875*, 2024.
- [26] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL <https://zenodo.org/records/12608602>.

- [27] Khashayar Gatmiry, Nikunj Saunshi, Sashank J Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? *arXiv preprint arXiv:2410.08292*, 2024.
- [28] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *International Conference on Learning Representations*, 2023. doi: 10.48550/arXiv.2310.01801.
- [29] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.
- [30] Google Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv: 2403.05530*, 2024.
- [31] Google Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL <https://arxiv.org/abs/2507.06261>.
- [32] Sia Gholami and Marwan Omar. Do generative large language models need billions of parameters? *arXiv preprint arXiv: 2309.06589*, 2023.
- [33] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.
- [34] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.
- [35] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.
- [36] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [37] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [38] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.
- [39] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Sophia Shao. Speed: Speculative pipelined execution for efficient decoding. *arXiv preprint arXiv:2310.12072*, 2023.
- [40] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv: 2401.18079*, 2024.
- [41] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33: 9782–9793, 2020.
- [42] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1 (2):3, 2022.

- [43] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2016. doi: 10.1007/978-3-319-46493-0_39. URL https://doi.org/10.1007/978-3-319-46493-0_39.
- [44] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.
- [45] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.
- [46] 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, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- [47] Guochao Jiang, Guofeng Quan, Zepeng Ding, Ziqin Luo, Dixuan Wang, and Zheng Hu. Flashthink: An early exit method for efficient reasoning. *arXiv preprint arXiv:2505.13949*, 2025.
- [48] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv: 2403.05527*, 2024.
- [49] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [50] Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, et al. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166*, 2023.
- [51] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36:39236–39256, 2023.
- [52] Deqian Kong, Minglu Zhao, Dehong Xu, Bo Pang, Shu Wang, Edouardo Honig, Zhangzhang Si, Chuan Li, Jianwen Xie, Sirui Xie, et al. Latent thought models with variational bayes inference-time computation. In *Forty-second International Conference on Machine Learning*.
- [53] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.
- [54] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *International Conference on Learning Representations*, 2019.
- [55] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [56] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.
- [57] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23. JMLR.org*, 2023.

- [58] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. *arXiv preprint arXiv:2210.06313*, 2022.
- [59] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Reza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.
- [60] Luyang Liu, Jonas Pfeiffer, Jiaxing Wu, Jun Xie, and Arthur Szlam. Deliberation in latent space via differentiable cache augmentation. *arXiv preprint arXiv:2412.17747*, 2024.
- [61] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.
- [62] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.
- [63] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.
- [64] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.
- [65] AI @ Meta Llama Team. The llama 3 herd of models. *arXiv preprint arXiv: 2407.21783*, 2024.
- [66] Yaxin Luo, Gen Luo, Jiayi Ji, Yiyi Zhou, Xiaoshuai Sun, Zhiqiang Shen, and Rongrong Ji. γ -mod: Exploring mixture-of-depth adaptation for multimodal large language models. *arXiv preprint arXiv:2410.13859*, 2024.
- [67] Mehrnaz Mofakhami, Reza Bayat, Ioannis Mitliagkas, Joao Monteiro, and Valentina Zantedeschi. Performance control in early exiting to deploy large models at the same cost of smaller ones. *arXiv preprint arXiv:2412.19325*, 2024.
- [68] Ali Momeni, Babak Rahmani, Benjamin Scellier, Logan G. Wright, Peter L. McMahon, Clara C. Wanjura, Yuhang Li, Anas Skalli, Natalia G. Berloff, Tatsuhiro Onodera, Ilker Oguz, Francesco Morichetti, Philipp del Hougne, Manuel Le Gallo, Abu Sebastian, Azalia Mirhoseini, Cheng Zhang, Danijela Marković, Daniel Brunner, Christophe Moser, Sylvain Gigan, Florian Marquardt, Aydogan Ozcan, Julie Grollier, Andrea J. Liu, Demetri Psaltis, Andrea Alù, and Romain Fleury. Training of physical neural networks. *arXiv preprint arXiv: 2406.03372*, 2024.
- [69] Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.
- [70] Kei-Sing Ng and Qingchen Wang. Loop neural networks for parameter sharing. *arXiv preprint arXiv:2409.14199*, 2024.
- [71] OpenAI. Gpt-4 technical report. *PREPRINT*, 2023.
- [72] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 design, automation & test in europe conference & exhibition (DATE)*, pages 475–480. IEEE, 2016.
- [73] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

- [74] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv: 2104.10350*, 2021.
- [75] 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. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL <https://openreview.net/forum?id=n6SCkn2QaG>.
- [76] Jacob Pfau, William Merrill, and Samuel R Bowman. Let’s think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.
- [77] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *arXiv preprint arXiv: 2211.05102*, 2022.
- [78] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR, 2022.
- [79] 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. *arXiv preprint arXiv:2404.02258*, 2024.
- [80] Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.
- [81] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv: 2207.07061*, 2022.
- [82] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- [83] Noam M. Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*, 2017.
- [84] Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.
- [85] Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.
- [86] Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, and Xian Li. Llm pretraining with continuous concepts. *arXiv preprint arXiv:2502.08524*, 2025.
- [87] Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*, 2021.
- [88] Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. *arXiv preprint arXiv:2310.07096*, 2023.
- [89] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.
- [90] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

- [91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [92] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *arXiv preprint arXiv: 2312.03863*, 2023.
- [93] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.
- [94] Haoyi Wu and Kewei Tu. Layer-condensed kv cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11175–11188, 2024.
- [95] Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. Tied transformers: Neural machine translation with shared encoder and decoder. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5466–5473, 2019.
- [96] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv: 2309.17453*, 2023.
- [97] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A walk with sgd. *arXiv preprint arXiv:1802.08770*, 2018.
- [98] Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Zheng Lin, Li Cao, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.
- [99] Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramid-infer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024.
- [100] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.
- [101] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.
- [102] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.
- [103] Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining language models to ponder in continuous space. *arXiv preprint arXiv:2505.20674*, 2025.
- [104] Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. Learning to skip for language modeling. *arXiv preprint arXiv:2311.15436*, 2023.
- [105] Jun Zhang, Desen Meng, Ji Qi, Zhenpeng Huang, Tao Wu, and Limin Wang. p-mod: Building mixture-of-depths mllms via progressive ratio decay. *arXiv preprint arXiv:2412.04449*, 2024.
- [106] Tianyi Zhang, Jonah Yi, Zhaozhao Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Advances in Neural Information Processing Systems*, 37:3304–3331, 2024.
- [107] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

- [108] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.
- [109] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.
- [110] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.
- [111] 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. *arXiv preprint arXiv:2202.08906*, 2022.

Contents

1	Introduction	1
2	Method	3
2.1	Preliminary	3
2.2	Mixture-of-Recurions	4
2.2.1	Routing Strategies: Expert-choice vs. Token-choice	4
2.2.2	KV Caching Strategies: Recursion-wise Caching vs. Recursive Sharing	5
3	Experiments	6
3.1	Main Results	6
3.2	IsoFLOP Analysis	7
3.3	Inference Throughput Evaluation	7
4	Ablation Studies	8
4.1	Parameter Sharing Strategies	8
4.2	Routing Strategies	8
4.3	KV Caching Strategies	9
5	Analysis	9
5.1	Compute-optimal Scaling Analysis	9
5.2	Routing Analysis	9
5.3	Test-time Scaling Analysis	10
6	Conclusion	10
A	Related Work	21
B	Limitations and Future Works	22
C	Details of Design Choices for Mixture-of-Recurions	23
C.1	Parameter-sharing Strategy	23
C.2	Routing Strategy	24
C.3	KV Caching Strategy	25
D	Experimental Setup	25
E	Expanded Results of Main Experiments	26
E.1	Performance Validation at 1.7B Scale	26
E.2	Increasing Recursion Depth under Fixed Parameters	26
F	Expanded Results of IsoFLOP Analysis	27
G	Details of Experimental Settings for Throughput Measurement	29
H	Expanded Results of Parameter Sharing Strategy	30

I	Expanded Results of Design Choices for Router	33
I.1	Details of Design Configurations	33
I.2	Router Performance Evaluation Metrics	33
I.3	Extended Evaluation Results of Router Designs	33
J	Expanded Results of KV Cache Sharing Mechanism	35
J.1	Key Value Representation Trends in Recursive Transformers	35
J.2	Performance Comparison of KV Sharing Strategy	36
K	Expanded Qualitative Results	37
K.1	Analysis on Adaptive Computation Paths	37
K.2	Analysis on Router Weights	39

A Related Work

Recursive Transformers. Parameter sharing provides an orthogonal path to efficiency [6, 20, 45, 54, 70, 87, 88, 95]. The *Universal Transformer* first showed that repeatedly applying a single block can match the representational power of deep, non-shared stacks [20]. *Looped Transformers* shown to be effective, can act as programmable computers [33], learn iterative data-fitting algorithms [100], generalize to much longer inputs on algorithmic tasks [23], and illuminate few-shot learning by mimicking multi-step optimizers [27]. Furthermore, Bae et al. [6] mitigate the accuracy loss often associated with weight tying by adding low-rank adaptation (LoRA) adapters [42] in each loop, yielding *Relaxed Recursive Transformers*. Recent work further demonstrates that Recursive Transformers excel at latent reasoning via recurrent depth [29]. While most prior studies focus on the efficiency gains from weight tying, the *recursive* architecture itself offers a second level: inspired by early-exiting [81] and compute routing [79], one can vary the number of recursions per input (e.g., per token), allocating compute only where it is most beneficial during both training and inference.

Adaptive computation. Many works have shown that *dynamic* compute allocation can markedly reduce the cost of training and inference, from traditional neural networks [8, 35, 43, 72, 90] to large language models [5, 21, 22, 24, 41, 79]. Early exiting methods learn to halt processing for “easy” samples (e.g., tokens or sequences in language modeling) by skipping the remaining layers [20, 21, 67, 81]. Alternatively, early exits can be combined with speculative decoding techniques [11, 57] during inference by leveraging lower layers for fast drafting [5, 22]. Recently, *Mixture-of-Depths* (MoD) [79] reframed adaptive depth as a routing problem: a router at each layer selects a *subset* of tokens to receive the full computation, while the rest bypass the layer, yielding finer-grained conditional compute. This new form of adaptive allocation is well suited to Transformer architectures and has already been extended to other modalities [66, 105], highlighting a promising paradigm of dynamic compute at token-level granularity. MoR applies this routing idea to *recursive* Transformers: tokens are dynamically sent through repeated calls of a single, weight-tied block instead of through distinct layers. This shift keeps parameter count constant, allows arbitrarily deep (adaptive) compute beyond the model’s physical depth.

Routing mechanism. LLMs have increasingly employed routers to enable adaptive computation, primarily in sparse Mixture-of-Experts (MoE) frameworks [15, 56, 83, 111], i.e., each token is processed by a subset of expert networks chosen by a learned router, dramatically increasing model capacity without a computational overhead. Early MoE architectures [24, 46, 56] adopted a *token-choice* routing strategy, wherein the router selects the top-k experts for each token based on its hidden state. While effective, this approach often leads to load imbalance across experts, necessitating auxiliary balancing losses. To address this, *expert-choice* routing [36, 110] has been proposed, wherein each expert selects the tokens to serve, ensuring perfect load balancing and improved efficiency. Building on this, a few works employed trainable routers to determine which layers to skip [25, 79, 104]. Unlike traditional early-exit methods, these expert-choice routing mechanisms enforce a static compute budget by capping the number of tokens processed per layer (or depth).

Key-value caching. Key-value (KV) caching stores the per-token key and value tensors produced at each layer during autoregressive decoding; reusing them eliminates quadratic-time recomputation and boosts throughput [9, 28, 48, 59, 77, 82, 96]. Unfortunately, retaining these tensors quickly saturates GPU memory, especially for long contexts and large batches [9, 14]. Prior work tackles this issue by quantizing KV activations to lower precision [40, 106], discarding entries that contribute little to the final output [63, 107], and sharing keys and values across attention heads [2, 82]. Brandon et al. [9] push this idea further, allowing adjacent layers to share the same key and value tensors and achieving additional memory savings with negligible quality loss. Our Mixture-of-Recursions offer a complementary avenue: KV caches generated in early recursions can be reused in later ones, potentially reducing memory consumption even further. This also provides the advantage of only needing to run the first recursion during prefill phase [85] (only with Cycle strategy), promising significant speedups for prompt settings over 1 million tokens. Two caching strategies in MoR can be optimized based on their distinct benefits to suit various deployment settings.

Latent reasoning. An emerging line of work enables LLMs to perform reasoning internally within hidden states rather than through explicit verbalization [13, 34, 52, 76, 86]. Many approaches adopt a *fixed* latent reasoning depth: they insert special tokens or structured prompts (e.g., a learnable “pause” token [34] or filler punctuation [76]) that allow the model to execute a predetermined number of hidden reasoning passes before producing an answer. Others reuse the model’s hidden states in a closed loop for a fixed number of iterations by feeding final hidden states back as input to simulate chain-of-thought [38, 80, 84]. Another line of research enhances latent reasoning by augmenting hidden states with intermediate semantic signals [86, 102]. However, these methods lack the flexibility to allocate computation where it is most needed, leading to unnecessary overhead on easy inputs and insufficient reasoning on complex ones. This motivates leveraging looping mechanisms for more adaptive latent reasoning [12, 29, 80, 103].

B Limitations and Future Works

Reasoning MoR models. Recent studies have highlighted the redundancy within reasoning chains and address it by applying token-level adaptive computation, like early-exit mechanisms [16, 47, 98]. Our MoR framework inherently enables latent reasoning by adaptively determining the necessary recursion depth for individual tokens. Therefore, a crucial future work involves exploring how the router can dynamically learn to adjust to the necessity of chain-of-thought (CoT) chains when post-trained on actual reasoning datasets. Developing advanced routing strategies that explicitly align recursion depth with reasoning complexity may enhance reasoning accuracy, computational efficiency, and even interpretability for deliberative reasoning process.

Further scaling model family. Due to current compute constraints, our experiments have been limited to models of up to 1.7 billion parameters. The next step is to train Mixture-of-Recursions (MoR) models at a larger scale (over 3 billion parameters) on substantially larger corpora. To enhance the scalability of the MoR architecture, we can first increase the size of the non-shared block. A potential follow-up work could introduce depth-specific LoRA [6] or experts (i.e., incorporating Mixture-of-Experts principles) and leverage expert parallelism [78] to efficiently compute the input at each depth in parallel, which is expected to improve model quality without huge speed bottlenecks. Furthermore, to reduce overall pre-training costs, we could also explore continued pre-training (uptraining), starting from existing pre-trained vanilla LLM checkpoints. As future work, we plan to investigate MoR performance using various initialization strategies for recursive models, as explored in prior work [6].

Adaptive capacity control. Expert-choice routing offers the significant advantage of guaranteeing perfect load balancing through pre-determined capacity factors [79, 110]. However, a limitation arises when we want to allocate different capacities during inference. Specifically, in our MoR models, we observe that when using an auxiliary loss, the router outputs for selected and unselected tokens are almost perfectly separated. This makes it challenging to adjust top-k values after training. Therefore, a more adaptive model design, which can leverage different capacities during both training and inference phases, is needed to address this limitation.

Compatibility with sparse algorithms. Given MoR’s token-level adaptive recursion, we can further optimize computation by integrating structured sparsity. This approach allows for the selective activation of subnetworks or parameters [64], dynamically pruning unnecessary computations at both the token and layer levels [22, 79]. This investigation into sparse model designs promises significant efficiency improvements. We believe many sparsity-based techniques, such as pruning [37] or quantization [44], are highly complementary to our MoR framework. This will provide deeper insights into effective sparse architectures within recursive models, offering promising directions for future research.

Expansion to multimodal and non-text domains. MoR’s recursion block is inherently modality-agnostic, allowing its adaptive depth mechanism to extend beyond text processing. This crucial property enables MoR to readily integrate into vision, speech, and unified multimodal transformer architectures. Applying token-adaptive recursion to long-context video or audio streams holds the potential for even greater memory efficiencies and substantial throughput gains, crucial for real-world applications. By dynamically adjusting the processing depth for each token or segment, MoR could unlock these significant benefits.

C Details of Design Choices for Mixture-of-Recursions

In this section, we provide detailed descriptions of the design choices employed in Mixture-of-Recursions, expanding upon the summary provided in the main pages.

C.1 Parameter-sharing Strategy

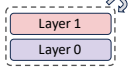
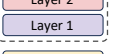
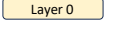
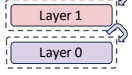
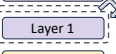
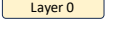
Table 5 shows formulation and visualization of four parameter-sharing strategies: Cycle, Middle-Cycle, Sequence, and Middle-Sequence. These strategies determine how a shared pool of blocks Φ' are reused across a total of L unrolled layers. The optimal strategy for parameter sharing in recursive models remains an open question.

In the **Cycle** strategy, a fixed set of parameters is reused cyclically across all recursion steps. By forcing the model to re-engage with the input through the same shared block, it encourages a deeper, iterative refinement process, akin to “rethinking” the problem from the ground up at every stage. However, because the same transformations are applied repeatedly regardless of input variation, it may limit the model’s capacity to learn diverse or highly specialized features.

On the other hand, the **Sequence** strategy assigns distinct parameters to each recursion block in sequential order. A potential drawback is that simply applying similar transformations twice in a row may lead to redundant features with diminishing returns. Nevertheless, the use of a fixed, sequential order of layers may provide a stable and predictable structure.

Building upon these strategies, the **Middle** sharing variant further refines parameter reuse by preserving unique parameters at the first and last layers while sharing weights only among the intermediate layers. This approach aims to balance the trade-off between parameter efficiency and representational flexibility, maintaining distinct entry and exit transformations while benefiting from reduced parameter redundancy in the middle layers. In line with recent findings [29, 50], Middle sharing can capture important input and output nuances more effectively than pure Cycle or Sequence sharing, without significantly increasing model size.

Table 5: Parameter-sharing strategies in Recursive Transformers. This table shows *Cycle*, *Middle-Cycle*, *Sequence*, and *Middle-Sequence* schemes with layer reuse, where Middle-* retains unique first and last layers.

Layers	Cycle Strategy		Middle-Cycle Strategy	
	Equation	Figure	Equation	Figure
Last	–		$f(\mathbf{h}_t^{L-1}; \Phi_{L-1})$	
Recursion	$f(\mathbf{h}_t^\ell; \Phi'_{\ell \bmod (L/N_r)})$		$f(\mathbf{h}_t^\ell; \Phi'_{(\ell-1 \bmod ((L-2)/N_r))+1})$	
First	–		$f(\mathbf{h}_t^0; \Phi_0)$	
Layers	Sequence Strategy		Middle-Sequence Strategy	
	Equation	Figure	Equation	Figure
Last	–		$f(\mathbf{h}_t^{L-1}; \Phi_{L-1})$	
Recursion	$f(\mathbf{h}_t^\ell; \Phi'_{\lfloor \ell/N_r \rfloor})$		$f(\mathbf{h}_t^\ell; \Phi'_{\lfloor (\ell-1)/N_r \rfloor + 1})$	
First	–		$f(\mathbf{h}_t^0; \Phi_0)$	

C.2 Routing Strategy

In this section, we provide an in-depth explanation of the two routing strategies employed in Mixture-of-RecurSIONs: *Expert-choice* and *Token-choice* routers. Each approach has distinct advantages and inherent limitations, which we first outline before discussing the mitigation techniques we utilized.

Expert-choice routing. The expert-choice router offers several advantages, including a fixed compute budget that simplifies resource management. However, it suffers from a key issue: the top- k selection operation, which requires information of tokens that appear later in the sequence, violates causality in autoregressive inference. This non-causal dependency (i.e., information leakage) can cause unexpected behavior during inference, potentially reducing model reliability.

To address these challenges, we explore two approaches: the auxiliary router and the auxiliary loss [79]. The **auxiliary router** is an additional lightweight network trained jointly but used only during inference; it predicts whether a token will be among the top- k selection. This additional router is trained with a binary cross-entropy loss, where the top- k selections from the main router are defined as the targets. Importantly, its training is isolated from the main objective through gradient blocking, so it does not affect the primary model training. Meanwhile, the **auxiliary loss** applies the binary cross-entropy loss to the main router itself, enabling it to simultaneously learn to push top- k tokens towards one and others towards zero during training. This ensures the router can reliably predict which tokens will be selected as top- k during inference.

Token-choice routing. In contrast, the token-choice router assigns recursion depths on a per-token basis without enforcing a fixed compute budget, thus avoiding leakage of information across tokens and preserving autoregressive properties. However, this introduces load imbalance across experts, which results in uneven token distribution across experts (or recursion depths), potentially causing inefficient compute allocation and unbalanced training.

To mitigate load imbalance, we employ two solutions from existing literature. **Balancing Loss** [24, 56] regularizes for a more uniform distribution of tokens across experts. For a sequence of length T , a balancing loss for MoR is calculated as follows:

$$\begin{aligned}\mathcal{L}_{\text{Balance}} &= \alpha \sum_{i=1}^{N_r} f_i P_i, \\ f_i &= \frac{N_r}{T} \sum_{t=1}^T \mathbb{I}(\text{Token } t \text{ selects Expert } i), \\ P_i &= \frac{1}{T} \sum_{t=1}^T g_t^i,\end{aligned}$$

where N_r is the total number of experts (which is also the number of recursion), g_t^i is the routing score of expert i for token t , f_i represents the fraction of tokens routed to expert i , P_i denotes the average routing scores of expert i , and λ is a hyperparameter controlling the strength of the auxiliary loss.

Loss-free [93] utilizes router biasing without explicit regularization loss. Specifically, this method adjusts per-expert bias terms b_i to balance token assignments across experts. During each training batch, routing scores are computed, and the number of tokens assigned to each expert (c_i) is counted. The load violation error is calculated as $e_i = \bar{c}_i - c_i$ where $\bar{c}_i$ is the average token count for expert i . Biases are then updated via $b_i \leftarrow b_i + u \times \text{sign}(e_i)$, where u is a bias update rate. The biased routing scores for selecting top- k expert are calculated as

$$g_i^t = \begin{cases} g_i^t, & \text{if } g_i^t + b_i \in \text{topk}(\{g_j^t + b_j \mid 1 \leq j \leq N\}, k) \\ 0, & \text{otherwise} \end{cases}$$

Note that the expert bias term is only utilized to adjust the routing strategy by influencing the top- k selection.

C.3 KV Caching Strategy

This work investigates two principal strategies for key-value (KV) caching to optimize memory usage during Recursive Transformer computations: *recursion-wise caching* and *recursive KV sharing*.

Recursion-wise caching. This keeps separate KV caches for each recursion step, ensuring tokens attend only to the KV pairs generated in their current recursion block. This prevents distribution mismatches between recursion steps, helping to maintain model accuracy while reducing memory and computational costs.

Recursive KV sharing. In contrast, recursive sharing reuses KV pairs computed in the first recursion step for all subsequent steps. Although this approach further lowers memory usage and eliminates the need to compute deeper recursion during the prefill phase, it introduces potential mismatches as later recursion steps receive KV representations originally intended for earlier steps. Such mismatch can negatively impact model performance when token routing is precise. Therefore, recursion-wise caching is generally preferred in settings with selective token routing to avoid performance degradation, while recursive KV sharing may be considered when memory efficiency is prioritized and prefill time is main bottleneck in the system.

D Experimental Setup

Training settings. We utilized a Llama-based Transformer architecture [65], referring to the configurations of the open-source SmolLM models [4]. All models were pretrained on a deduplicated subset of the FineWeb-Edu dataset [75] in SmolLM-Corpus [7], which comprises 220 billion tokens sourced from educational materials. Pretraining was conducted using four H100 or A100 GPUs. In our main and isoFLOPs analysis experiments, we utilized a Trapezoid learning rate scheduler, which consists of warmup (about 5%), stable, and cooldown (20%) phases. This approach allows us to efficiently continue pretraining for scaling laws from intermediate checkpoints, eliminating the need to train all models independently. In contrast, for all other experiments, we used a simple cosine annealing scheduler.

Evaluation settings. To assess model performance, we evaluated few-shot accuracy on six benchmarks using the Language Model Evaluation Harness: LAMBADA (LD), HellaSwag (HS), PIQA (PQ), WinoGrande (WG), ARC (Easy and Challenge), and MMLU. For all few-shot datasets, excluding LAMBADA, WinoGrande, and MMLU, we normalized accuracy by the byte length of the target string. We adhered to the standard number of shots for each dataset, and used the continuation task specifically for MMLU for simplicity. All evaluation performance measurements were conducted using a single H100 or A100 GPU.

Model architecture details. Table 6 summarizes the architectural specifications of the four Vanilla Transformer models used as the base for our recursive models. Each model variant differs in scale, ranging from 135M to 1.7B total parameters (including both non-embedding and embedding components). For consistency and comparability, all models are trained using a vocabulary size of 49K and a maximum input sequence length of 2K tokens.

Table 6: Key parameters of four model size variants. A model’s size is defined by the total number of its non-embedding and embedding parameters. Three small models utilize Grouped-Query Attention [1], reducing the number of key-value heads. We refer to the base configurations of the open-sourced SmolLM models [4].

Models	Base Configuration				Attention & Feed-Forward				Input	
	N-emb	Emb	N_L	d_{model}	N_{head}	N_{KV}	d_{head}	d_{inter}	Vocab	L_{ctx}
Vanilla 135M	106M	28M	30	576	9	3	64	1536	49K	2K
Vanilla 360M	315M	47M	32	960	15	5	64	2560	49K	2K
Vanilla 730M	654M	75M	26	1536	24	8	64	4096	49K	2K
Vanilla 1.7B	1.61B	101M	24	2048	32	32	64	8192	49K	2K

E Expanded Results of Main Experiments

E.1 Performance Validation at 1.7B Scale

We validate our models by scaling up the size to 1.7B, all under the same FLOPs budget. As shown in Table 7, the MoR model with expert-choice routing shows strong performance, but the vanilla model performs slightly better. While this could be due to the differences in the optimal number of tokens associated with each model’s optimal scaling policy, it might also signal that the current architecture design of MoR is not suitable for scaling. Nevertheless, we may be able to further improve performance by increasing the number of non-shared blocks (despite reduced efficiency), applying depth-specific LoRA or MoE to each recursion [6], or uptraining with initialization from a well-pretrained checkpoint. Further validation with larger sizes or training on more tokens is required.

Table 7: Comparison of MoR, Recursive, and Vanilla Transformers at 1.7B scale under both fixed FLOPs (68.5e18) settings. All models are trained on FineWeb-Edu and evaluated by few-shot accuracy. For the isoFLOP rows, the number of training tokens (N_{tok}) varies by model efficiency. For the model sizes, we report non-embedding parameter counts. [†]In recursive models, all tokens go through fixed recursion depths (N_r), instead of adaptive depths.

Models	MoR		Recursion		Pretrain			Few-shot Accuracy $\uparrow$						
	Type	KV	Share	N_r	Param	FLOPs	N_{tok}	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla	-	-	-	-	1.61B	68.5	20B	40.8	49.4	70.6	54.8	47.4	30.2	48.9
Recursive [†]	-	-	M-Cyc	2	0.87B	68.5	18B	37.3	46.5	68.9	52.6	44.2	29.6	46.5
	-	-	M-Cyc	3	0.67B	68.5	20B	36.4	45.3	69.5	52.7	43.9	29.1	46.2
MoR (ours)	Expert	Cache	M-Cyc	2	0.87B	68.5	26B	41.1	47.5	70.0	55.6	46.0	30.3	48.4
	Expert	Cache	M-Cyc	3	0.67B	68.5	27B	37.2	46.6	69.1	53.7	44.1	29.7	46.7
	Token	Cache	M-Cyc	3	0.67B	68.5	30B	35.6	43.2	68.1	53.0	43.4	29.0	45.4

E.2 Increasing Recursion Depth under Fixed Parameters

Table 8 illustrates the performance change when the number of recursions (N_r) is increased while the number of unique parameters is fixed at 118M. As the number of recursions rises from 1 (Vanilla) to 2 and 3 (MoR), the negative log-likelihood decreases on both the train and validation sets, and the average few-shot accuracy improves. This demonstrates that the MoR model can effectively scale performance by merely increasing the recursion steps, even with a fixed model size.

Table 8: Comparison of MoR with different numbers of recursions under the same number of parameters. Vanilla equals MoR with recursion 1. All models are trained on FineWeb-Edu with 10B tokens, and we apply the Middle-Cycle parameter sharing for MoR models. We report negative log-likelihood (NLL) on the train and validation sets and few-shot accuracy across six benchmarks.

Models	Pretrain			Recursion		NLL $\downarrow$		Few-shot Accuracy $\uparrow$						
	N-Emb	N_L	N_{tok}	N_r	Share	Train	Valid	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla	118M	12	10B	1	-	2.9182	2.9678	25.81	33.08	61.81	51.46	38.10	26.64	39.48
MoR	118M	1+10+1	10B	2	M-Cyc	2.8767	2.9205	27.09	33.84	62.13	53.04	36.77	26.88	39.96
MoR	118M	1+10+1	10B	3	M-Cyc	2.8667	2.9111	27.38	34.60	63.17	51.46	37.15	26.83	40.10

F Expanded Results of IsoFLOP Analysis

In the main paper (§3.2), we compared Vanilla, Recursive and our Mixture-of-Recursions (MoR) models under matched *training compute*. Four base model capacities were studied—135M, 360M, 730M and 1.7B parameters. For recursive and MoR models, we fix the recursion count to $N_r = 3$, so the number of *unique* parameters is roughly one-third of the vanilla counterpart. Each architecture is trained once for the *largest* compute budget (16.5EB)⁵ and the resulting checkpoint is re-used to obtain the 5EB and 2EB variants, as detailed below.

FLOPs approximated calculation of Transformers. We follow the approximation for calculating FLOPs as detailed in Kaplan et al. [49]. Our analysis solely focuses on forward pass FLOPs, since the FLOPs involved in the backward pass are typically just double those of the forward pass. For most operations within Transformers, which primarily consist of linear projections, the forward pass FLOPs are calculated as two times the number of parameters, excluding the attention mechanism.

Regarding attention, we specifically account for the operations from the dot product between queries and keys and the scaling of values with softmax values. We only calculate FLOPs that contribute to the actual loss, excluding redundant computations in the upper triangular portion due to causality masking. Furthermore, we omit any additional computational costs associated with FlashAttention [18], normalization, and non-linearity operations from our overall FLOPs calculation.

As a result, Vanilla and Recursive Transformers have the same FLOPs. For MoR, the FLOPs calculation varies based on the routing and KV caching strategy. Especially, we calculated FLOPs based on the sequence length at each recursion depth, which is determined by the capacity factor and caching mechanism. In the case of token-choice routing, since the actual token allocation changes at every step, we approximated the FLOPs by assuming perfect balancing. Furthermore, we add extra layers to a few MoR models to ensure their effective depth is divisible by the recursion number. For example, in a 135M model with 30 layers, setting a base depth of 10 and applying recursion three times (as in the Middle-Cycle strategy) results in a total of 32 layers. These additional layers introduce extra FLOPs, so we reduce the number of training steps accordingly to maintain our predefined FLOP budget.

Trapezoid learning-rate schedule with checkpoint reuse. To avoid retraining every model from scratch for each FLOPs budget, we employ the *trapezoid* schedule [97]. The rule of this scheduler is as follows:

$$\eta(t) = \begin{cases} \frac{t}{w} \eta_{\max}, & 0 \leq t < w \quad (\text{warm-up}), \\ \eta_{\max}, & w \leq t < p \quad (\text{plateau}), \\ \eta_{\max} \left(1 - \frac{t-p}{d}\right), & p \leq t < p + d \quad (\text{cool-down}), \end{cases}$$

where w denotes the warm-up interval, $p - w$ is the constant-LR plateau, and d represents the cool-down segment. This stable phase allows us to efficiently manage experiments by saving intermediate checkpoints and then running additional cool-down steps from those points, according to each budget. For the warmup, we allocate 5 % of the total training steps of the smallest budget (2EB), and we set the cool-down steps to 20 % of the total training steps for each *corresponding* budget.

⁵1EB = 10^{18} floating-point operations.

Results at a glance. Table 9 reports NLL on FineWeb-Edu validation set and few-shot accuracy on six benchmarks. Our findings reveal a clear trend: more compute consistently leads to better models, evidenced by lower NLL and improved accuracy with higher FLOPs. However, weight-sharing alone in recursive models degraded performance compared to Vanilla, a clear trade-off for the reduced parameters. Crucially, token-routed MoR models overcome this shortcoming, catching up to and then surpassing Vanilla models from 360M parameters upward, all while utilizing only one-third of the parameters. This performance advantage persists at 730M and 1.7B parameter scales.

Table 9: Detailed results of isoFLOP analysis across three compute budgets. We evaluate negative log-likelihood (NLL) on the FineWeb-Edu validation set and few-shot accuracy on six downstream tasks for four base model sizes (135M, 360M, 730M, 1.7B). Each model was initially trained up to 16.5EB and sliced back to 5EB and 2EB via mid-training checkpoints using a trapezoid learning-rate schedule. All models used three recursion steps. For MoR models, we use expert-choice routing and recursion-wise caching mechanisms. We highlight the best-performing model in each setting in gray.

		Pretrain				Recursion		NLL ↓	Few-shot Accuracy ↑						
Models	Base	N-Emb	N _L	FLOPs	N _{tok}	Share	Loop	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla	135M	106M	30	2.0e+18	6.5B	-	-	3.0922	22.80	30.93	62.35	51.14	36.28	26.29	38.30
Recursive	135M	42M	1+10+1	2.0e+18	6.1B	M-Cyc	3	3.2058	19.79	29.32	60.17	50.59	34.83	25.40	36.68
MoR	135M	42M	1+10+1	2.0e+18	9.2B	M-Cyc	3	3.1077	21.13	31.00	59.79	49.09	34.87	25.63	36.92
Vanilla	135M	106M	30	5.0e+18	16.1B	-	-	2.9464	26.88	33.69	63.98	51.46	37.08	27.07	40.03
Recursive	135M	42M	1+10+1	5.0e+18	15.1B	M-Cyc	3	3.0534	24.51	31.57	62.40	50.83	35.78	25.94	38.51
MoR	135M	42M	1+10+1	5.0e+18	23.1B	M-Cyc	3	3.0192	22.01	32.53	61.75	49.88	35.39	26.19	37.96
Vanilla	135M	106M	30	16.5e+18	53.3B	-	-	2.8432	30.16	36.51	64.80	53.43	40.17	27.82	42.15
Recursive	135M	42M	1+10+1	16.5e+18	50.0B	M-Cyc	3	2.9552	25.98	33.36	63.98	51.78	36.96	26.68	39.79
MoR	135M	42M	1+10+1	16.5e+18	76.2B	M-Cyc	3	2.9490	22.61	33.99	61.92	47.83	35.95	26.36	38.11
Vanilla	360M	315M	32	2.0e+18	2.4B	-	-	3.3785	17.27	27.90	59.36	51.38	32.10	25.49	35.58
Recursive	360M	118M	1+10+1	2.0e+18	2.4B	M-Cyc	3	3.4864	10.34	26.66	58.00	51.54	30.94	24.94	33.74
MoR	360M	118M	1+10+1	2.0e+18	3.6B	M-Cyc	3	3.1026	24.14	30.53	61.86	50.99	34.74	25.50	37.96
Vanilla	360M	315M	32	5.0e+18	6.0B	-	-	3.0097	25.17	32.10	63.22	48.62	36.01	26.69	38.63
Recursive	360M	118M	1+10+1	5.0e+18	6.0B	M-Cyc	3	3.0722	23.29	31.19	62.62	51.30	35.85	25.99	38.37
MoR	360M	118M	1+10+1	5.0e+18	9.0B	M-Cyc	3	2.9161	28.33	34.53	63.22	51.07	36.70	26.98	40.14
Vanilla	360M	315M	32	16.5e+18	19.8B	-	-	2.7824	31.94	37.92	66.10	51.30	39.70	27.95	42.49
Recursive	360M	118M	1+10+1	16.5e+18	19.8B	M-Cyc	3	2.8466	29.75	35.92	64.91	51.46	39.12	27.18	41.39
MoR	360M	118M	1+10+1	16.5e+18	29.7B	M-Cyc	3	2.7924	33.15	37.94	66.97	52.09	38.46	27.49	42.68
Vanilla	730M	654M	26	2.0e+18	1.2B	-	-	3.7164	07.74	26.58	57.62	51.14	29.74	24.46	32.88
Recursive	730M	252M	1+8+1	2.0e+18	1.2B	M-Cyc	3	3.8136	05.53	26.25	55.77	50.59	29.88	24.63	32.11
MoR	730M	252M	1+8+1	2.0e+18	1.8B	M-Cyc	3	3.3300	17.93	28.74	59.30	51.46	33.14	25.37	35.99
Vanilla	730M	654M	26	5.0e+18	3.1B	-	-	3.0821	22.05	31.99	62.68	50.67	35.88	26.12	38.23
Recursive	730M	252M	1+8+1	5.0e+18	3.1B	M-Cyc	3	3.1640	18.51	30.72	62.13	47.83	35.97	25.84	36.83
MoR	730M	252M	1+8+1	5.0e+18	4.5B	M-Cyc	3	3.0067	26.18	32.76	62.46	50.91	36.93	26.37	39.27
Vanilla	730M	654M	26	16.5e+18	10.1B	-	-	2.7048	34.50	40.29	66.81	49.49	40.82	28.66	43.43
Recursive	730M	252M	1+8+1	16.5e+18	10.1B	M-Cyc	3	2.7886	30.76	37.84	65.51	52.41	39.26	27.51	42.21
MoR	730M	252M	1+8+1	16.5e+18	14.9B	M-Cyc	3	2.7438	32.93	39.55	66.32	54.38	40.00	28.09	43.55
Vanilla	1.7B	1.61B	24	2.0e+18	0.6B	-	-	5.1349	00.00	24.96	51.03	51.38	25.75	23.07	29.37
Recursive	1.7B	0.67B	1+8+1	2.0e+18	0.5B	M-Cyc	3	5.3277	00.00	25.27	51.36	48.62	26.52	22.98	29.13
MoR	1.7B	0.67B	1+8+1	2.0e+18	0.8B	M-Cyc	3	4.1175	01.44	25.80	53.97	49.64	27.56	24.08	30.42
Vanilla	1.7B	1.61B	24	5.0e+18	1.5B	-	-	3.6926	08.33	26.84	57.29	51.30	29.72	24.51	33.00
Recursive	1.7B	0.67B	1+8+1	5.0e+18	1.3B	M-Cyc	3	3.8876	03.14	26.57	54.73	49.17	29.01	24.49	31.19
MoR	1.7B	0.67B	1+8+1	5.0e+18	2.0B	M-Cyc	3	3.2905	17.62	28.32	59.03	49.80	32.14	25.28	35.37
Vanilla	1.7B	1.61B	24	16.5e+18	4.8B	-	-	2.8658	26.94	35.61	64.74	50.59	38.55	26.81	40.54
Recursive	1.7B	0.67B	1+8+1	16.5e+18	4.5B	M-Cyc	3	3.0042	23.25	32.09	62.95	50.75	37.64	26.53	38.87
MoR	1.7B	0.67B	1+8+1	16.5e+18	6.5B	M-Cyc	3	2.8316	28.10	36.18	64.64	50.99	38.68	27.25	40.97

G Details of Experimental Settings for Throughput Measurement

We implement a continuous depth-wise batching inference system [6, 39] to evaluate decoding throughput of MoR models. Queries are enqueued and scheduled dynamically during decoding using 1K samples from the FineWeb-Edu validation set. In particular, for MoR, when some queries exit early, the vacant slots in the batch are immediately filled with new queries waiting in the queue, maintaining a fully utilized batch at all times.

We compare the throughput of Vanilla and MoR models (at a 360M parameter scale) for generating a certain length of tokens per sample, where the number is sampled from a normal distribution with a mean of 256, starting without any input prefix. The speeds of the MoR models are normalized against the speed of the Vanilla Transformer. For pretraining MoR-4 models, we add two additional layers (34 layers in total) before applying recursion. This ensures the total effective depth is divisible by the recursion number (specifically for the Middle-Cycle strategy). Consequently, the speed comparison for MoR-4 is made against a modified vanilla model that includes these two extra layers, resulting in a total of 34 layers (32 original layers + 2 added layers).

We use two batching settings: (1) a *fixed batch size of 32* and (2) a *relative maximum batch size*, derived by multiplying 32 by the ratio of the maximum batch sizes of vanilla and MoR models. Specifically, based on the H100 GPU’s VRAM size, we calculated the maximum batch sizes by considering model parameters and their KV cache memory. For simplicity, we omit the memory size from the hidden states at the current position. Under these adaptive conditions, MoR-2 supports a batch size of 42, MoR-3 supports 48, and MoR-4 supports up to 51. By employing recursion-wise KV caching, MoR allows a substantial increase in batch size stemming from its reduced parameter and KV cache memory footprint.

For implementation, we use a queue to enable continuous depth-wise batching and employ FlashAttention 2 [17] to support variable-length KV caches within a batch. We adopt a *static*-sized cache where each position is updated over time, since this is compatible with `torch.compile` [73] to further optimize inference speeds. Furthermore, mimicking real-world deployment scenarios [53, 109], we decouple the transformer block phase from the rest of the computation by pre-processing the input embeddings or the first non-shared layer before passing into the transformer blocks. Then, we measured the actual time taken during the forward pass. Note that we include a warmup stage by running the model for 100 iterations before actual measurement, in order to obtain stable timing results. For further optimization, tokens that exited early were accumulated up to the maximum batch size before being processed by the last non-shared layer, classifier, and embedding layers (including the non-shared first layer in the case of MoR models). After this, we queue them for sequential batching by following a FIFO (First-In, First-Out) strategy. For implementation convenience, we exclude the time spent on caching and updating for KV pairs, as these aspects can be significantly optimized through various engineering techniques [53]. We leave a more precise speed comparison, which accounts for these considerations, as future work.

H Expanded Results of Parameter Sharing Strategy

This section complements the ablation in §4.1 by providing the full quantitative panorama behind Figure 4b. We revisit the four weight-tying schemes—*Cycle*, *Sequence*, *Middle-Cycle*, and *Middle-Sequence*—on two base model scales (135M and 360M non-embedding parameters) and two different recursion depths ($N_r = 2$ and 3). All models were trained from scratch for 10B tokens under identical optimization hyperparameters. Validation NLL on FineWeb-Edu and averaged few-shot accuracy over six benchmarks are summarized in Table 10.

Middle-Cycle is consistently the safest choice. For the 360M models, Middle-Cycle achieves the lowest NLL at both depths ($N_r = 2$ and $N_r = 3$) and also shows the largest improvement in average accuracy compared to vanilla reduced models. For the 135M models, while Cycle is slightly ahead at two recursion setting (3.0071 vs. 3.0330), Middle-Cycle overtakes when recursion depth rises (3.1048 vs. 3.1154) and shows a steadier accuracy profile. Meanwhile, pure Sequence sharing records the worst NLL in all four settings, and its accuracy gap widens with recursion depth. The Middle strategy slightly improves the performance of the Sequence, but it still performs worse than the Cycle-based methodology. We visualized the results in Figure 6.

Table 10: Comparison of parameter-sharing strategies (Cycle, Sequence, Middle-Cycle, Middle-Sequence) across two model scales (135M and 360M) and two recursion depths ($N_R = 2$ and $N_R = 3$). All models are pretrained from scratch on 10B tokens. We report validation negative log-likelihood (NLL) on FineWeb-Edu and few-shot accuracy across six tasks. Middle-Cycle consistently outperforms other strategies in both NLL and average task accuracy, especially at higher recursion depth. We highlight the optimal strategy for each setting in gray.

	Pretrain			Recursion		NLL↓	Few-shot Accuracy↑						
Base Model	N-Emb	N_L	N_{tok}	Share	Loop	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla 135M	106M	30	10B	-	-	3.0323	24.14	31.12	61.15	52.01	34.74	25.95	38.19
Vanilla 135M	53M	15	10B	-	-	3.0818	23.64	30.10	60.94	50.99	35.38	25.93	37.83
Vanilla 135M	35M	10	10B	-	-	3.1582	21.46	29.30	60.01	52.01	34.40	25.53	37.12
Vanilla 135M	53M	15	10B	Cyc	2	3.0071	25.52	31.25	61.10	50.99	36.08	26.11	38.51
Vanilla 135M	53M	15	10B	Seq	2	3.1093	22.39	29.60	61.10	50.12	34.46	25.72	37.23
Vanilla 135M	57M	1+14+1	10B	M-Cyc	2	3.0330	23.40	31.20	61.59	50.59	35.44	25.54	37.96
Vanilla 135M	57M	1+14+1	10B	M-Seq	2	3.0991	21.70	30.06	60.45	49.41	35.20	25.74	37.09
Vanilla 135M	35M	10	10B	Cyc	3	3.1154	21.42	30.14	60.61	49.72	34.15	25.57	36.94
Vanilla 135M	35M	10	10B	Seq	3	3.1637	19.99	29.39	59.25	51.62	33.79	25.32	36.56
Vanilla 135M	39M	1+9+1	10B	M-Cyc	3	3.1048	22.41	30.35	61.04	49.01	34.80	25.91	37.26
Vanilla 135M	39M	1+9+1	10B	M-Seq	3	3.1602	20.69	29.35	61.43	51.30	34.40	25.51	37.11
Vanilla 360M	315M	32	10B	-	-	2.8471	27.27	34.78	64.20	52.80	38.29	26.72	40.68
Vanilla 360M	157M	16	10B	-	-	2.8908	27.01	33.49	64.42	52.09	37.40	26.54	40.16
Vanilla 360M	98M	10	10B	-	-	2.9449	26.41	32.93	63.38	50.36	37.15	26.48	39.45
Vanilla 360M	157M	16	10B	Cyc	2	2.8487	28.47	34.79	63.06	49.96	37.38	26.81	40.08
Vanilla 360M	157M	16	10B	Seq	2	2.9467	26.33	32.49	62.89	52.41	36.37	26.24	39.46
Vanilla 360M	167M	1+15+1	10B	M-Cyc	2	2.8295	28.59	34.98	64.53	50.51	39.68	27.20	40.91
Vanilla 360M	167M	1+15+1	10B	M-Seq	2	2.9303	26.14	32.71	62.79	51.38	36.31	25.73	39.18
Vanilla 360M	98M	10	10B	Cyc	3	2.9363	25.87	32.98	62.89	50.28	36.35	26.54	39.15
Vanilla 360M	98M	10	10B	Seq	3	3.0245	24.55	31.48	63.11	49.25	35.65	25.73	38.30
Vanilla 360M	118M	1+10+1	10B	M-Cyc	3	2.8760	28.51	34.89	64.31	50.51	39.51	27.20	40.82
Vanilla 360M	118M	1+10+1	10B	M-Seq	3	2.9753	24.18	31.89	62.08	49.72	36.47	26.27	38.44

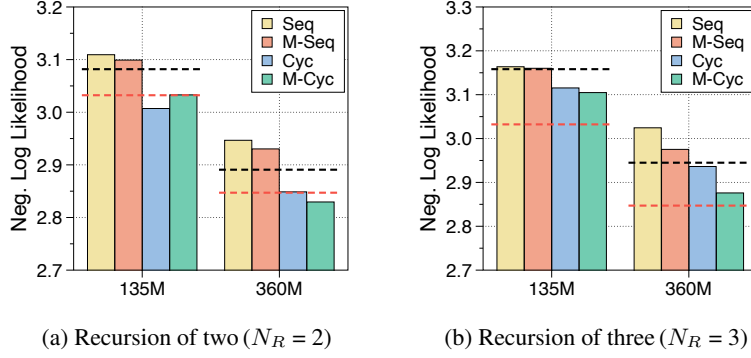


Figure 6: Validation negative log-likelihood (lower is better) on *FineWeb-Edu* for four parameter-sharing strategies. Bars are grouped by model capacity (135M vs. 360M parameters). Middle-Cycle consistently attains the lowest NLL, with its margin widening as either model size or depth increases. The horizontal dashed lines mark the *untied* (non-sharing) baselines: the lower red line represents the full capacity model, while the upper black line represents a parameter-matched reduced model with a footprint equal to the unique trainable parameter sizes of the recursive model.

For fairer comparison, we also compare the Middle-Cycle and Cycle sharing strategies, fixing the number of unique parameters for both models. For 730M model comparison, both strategies use 3 recursion steps. Especially, Middle-Cycle uses 8 layers for the recursion block and 2 unshared layers ($1 + 8 \times 3 + 1$), resulting in 10 unique layers and 26 effective layers. We compare this to the Cycle strategy with 10 layers per recursion block (10×3), which also has 10 unique layers but a total of 30 effective layers. Despite having fewer effective layers, as shown in Table 11, the Middle-Cycle model achieves better performance, with a lower negative log-likelihood (2.7552 vs. 2.7573) and higher average few-shot accuracy (42.32 vs. 41.90). While these results are promising, we leave further validation across more scales as future work.

Table 11: Comparison of Middle-Cycle and Cycle sharing strategies under the same number of unique parameters. All models use 730M scales (non-embedding parameters) as the base model with 3 recursion steps. The models are trained from scratch on 10B tokens using the Fineweb-Edu dataset.

Base Model	Recursion		Pretrain			NLL ↓	Few-shot Accuracy ↑							
	Share	Loop	N-Emb	N_L	N_{tok}		Fineweb	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla 730M	Cycle	3	252M	10	10B	2.7573	28.72	37.31	65.56	52.01	39.94	27.85	41.90	
Vanilla 730M	M-Cycle	3	252M	1+8+1	10B	2.7552	29.75	37.73	66.00	52.49	40.49	27.50	42.32	

Behavior under continued pre-training (up-training). Table 12 extends the study by “up-training” models—continuing from open-sourced SmolLM [4] checkpoints for an additional 5B tokens. Both Middle strategies demonstrate superior performance across all settings, and notably, they significantly outperform the reduced baseline models that are initialized in the same manner but without recursion. The other strategies reach a performance plateau earlier, suggesting that they have limited room for further improvement in capacity.

Table 12: Uptraining results across four parameter sharing strategies. Models are trained on 5B tokens from FineWeb-Edu and evaluated by train NLL and few-shot accuracy across six benchmarks. ARC denotes average of ARC-Easy and ARC-Challenge tasks, MMLU denotes the MMLU-Cont task. We highlight the optimal strategy for each setting in gray.

Base Model	Pretrain			Recursion			NLL ↓	Few-shot Accuracy ↑						
	N-Emb	N_L	N_{tok}	Share	Init	Loop	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg
Vanilla 360M	315M	32	5B	-	-	-	2.4825	41.67	50.63	70.35	55.09	46.99	30.82	49.26
Vanilla 360M	157M	16	5B	-	Step	1	2.7168	31.85	37.59	64.74	53.20	41.06	27.34	42.63
Vanilla 360M	157M	16	5B	Cyc	Avg	1	2.8603	22.14	30.36	60.07	48.22	34.99	25.56	36.89
Vanilla 360M	157M	16	5B	Seq	Avg	1	2.7919	25.40	32.35	62.30	50.12	35.88	26.19	38.71
Vanilla 360M	98M	10	5B	-	Step	1	2.8915	26.63	35.03	64.42	52.09	38.75	26.86	40.63
Vanilla 360M	98M	10	5B	Cyc	Avg	1	3.0512	23.19	31.27	62.30	51.22	36.71	26.29	38.50
Vanilla 360M	98M	10	5B	Seq	Avg	1	2.9915	25.67	32.21	62.30	51.38	36.68	26.56	39.14
Vanilla 360M	157M	16	5B	Cyc	Step	2	2.7165	31.30	37.68	64.91	52.17	39.29	27.53	42.15
Vanilla 360M	157M	16	5B	Cyc	Avg	2	2.8263	23.21	30.52	60.55	50.28	36.01	25.50	37.68
Vanilla 360M	157M	16	5B	Cyc	Lower	2	2.8024	27.67	34.71	63.49	49.88	38.12	26.87	40.13
Vanilla 360M	157M	16	5B	Cyc	Upper	2	2.7915	18.26	34.88	63.06	51.85	39.27	26.88	39.03
Vanilla 360M	157M	16	5B	Cyc	Rand	2	2.7575	25.29	34.78	61.64	52.01	38.09	26.62	39.74
Vanilla 360M	157M	16	5B	Seq	Step	2	2.6862	34.14	42.49	67.90	53.35	43.24	28.80	44.99
Vanilla 360M	157M	16	5B	Seq	Avg	2	2.7508	29.01	34.13	63.60	52.09	36.31	26.58	40.29
Vanilla 360M	157M	16	5B	Seq	Lower	2	2.8300	27.50	33.28	63.38	51.38	37.44	26.32	39.88
Vanilla 360M	157M	16	5B	Seq	Upper	2	2.7498	30.49	40.07	65.61	52.25	40.28	28.17	42.81
Vanilla 360M	157M	16	5B	Seq	Rand	2	2.7153	32.00	41.31	66.10	53.35	42.13	28.52	43.90
Vanilla 360M	167M	1+15+1	5B	M-Cyc	Step	2	2.6800	35.47	42.39	67.19	50.99	42.54	28.79	44.56
Vanilla 360M	167M	1+15+1	5B	M-Cyc	Avg	2	2.7314	33.81	40.42	66.87	51.78	41.68	28.17	43.79
Vanilla 360M	167M	1+15+1	5B	M-Cyc	Lower	2	2.7449	30.76	39.50	66.16	50.99	41.12	28.07	42.77
Vanilla 360M	167M	1+15+1	5B	M-Cyc	Upper	2	2.6605	34.41	43.74	67.46	53.20	43.75	28.93	45.25
Vanilla 360M	167M	1+15+1	5B	M-Cyc	Rand	2	2.6730	35.65	43.04	67.74	52.17	42.60	28.62	44.97
Vanilla 360M	167M	1+15+1	5B	M-Seq	Step	2	2.6627	35.09	43.34	67.57	51.22	43.66	28.91	44.97
Vanilla 360M	167M	1+15+1	5B	M-Seq	Avg	2	2.7143	33.92	40.93	66.49	51.70	40.72	28.24	43.66
Vanilla 360M	167M	1+15+1	5B	M-Seq	Lower	2	2.7696	30.74	38.36	65.94	51.78	41.27	27.73	42.64
Vanilla 360M	167M	1+15+1	5B	M-Seq	Upper	2	2.6931	32.66	42.35	67.14	52.80	42.83	28.49	44.38
Vanilla 360M	167M	1+15+1	5B	M-Seq	Rand	2	2.6908	35.07	42.03	66.32	53.75	43.11	28.42	44.78
Vanilla 360M	98M	10	5B	Cyc	Step	3	2.8901	27.46	35.26	63.82	51.54	39.35	27.44	40.81
Vanilla 360M	98M	10	5B	Seq	Step	3	2.8258	30.43	37.37	63.76	52.33	40.55	27.65	42.02
Vanilla 360M	118M	1+10+1	5B	M-Cyc	Step	3	2.7735	31.38	39.31	65.51	50.51	40.70	27.65	42.51
Vanilla 360M	118M	1+10+1	5B	M-Seq	Step	3	2.7678	31.67	39.23	65.89	52.09	40.65	27.90	42.91
Vanilla 360M	118M	1+10+1	5B	M-Cyc	Upper	3	2.8100	28.86	37.61	65.51	52.57	41.44	28.17	42.36

I Expanded Results of Design Choices for Router

I.1 Details of Design Configurations

We investigate various router design choices to optimize performance and stability. Specifically, we tune the coefficient values controlling the strength of auxiliary or balancing loss terms (*Coeff*), and adjust the scaling factor applied after the router function (α) to modulate routing weights. Moreover, we test different activation functions (*Func*), such as sigmoid or softmax, are evaluated, with architectural variations (*Arch*) of the router network, including linear layer, 2-layer MLP with GELU activation, Wide-MLP that expands the hidden layer size by a factor of four.

We also incorporate several techniques to stabilize training. To improve training stability, we utilize the *router z-loss* [111], which penalizes large logits produced by the gating network. Large logits can cause numerical instability and hinder effective training of the router. The z-loss is computed as follows:

$$L_z(x) = \frac{1}{B} \sum_{i=1}^B \left(\log \sum_{j=1}^{N_r} e^{x_j^{(i)}} \right)^2,$$

where B is the number of tokens in the batch, N_r is the number of experts, and $x \in \mathbb{R}^{B \times N_r}$ denotes the logits input to the router. This regularization encourages the gating network to produce smaller logits, promoting more stable and reliable routing decisions.

I.2 Router Performance Evaluation Metrics

Expert-choice routing. We evaluate dead token ratio and sampling accuracy to assess the router’s selection behavior. The *dead token ratio* measures the proportion of tokens at specific positions within the batch that are consistently unselected during the final recursion step, indicating a positional bias where certain token positions are systematically neglected by the router. The *sampling accuracy* how well the router used during inference predicts whether a token belongs to the top- k tokens identified during training, reflecting the router’s ability to consistently select the most relevant tokens. Ideally, high sampling accuracy with a low dead token ratio indicates a router that both identifies important tokens accurately and maintains diversity in token selection.

Token-choice routing. We evaluate the router’s ability to balance token assignments across experts using MaxVio (maximum violation) and entropy metrics. *MaxVio* [93] measures the load imbalance across experts:

$$\text{MaxVio} = \frac{\max_i \text{Load}_i - \overline{\text{Load}}_i}{\overline{\text{Load}}_i},$$

where Load_i denotes the actual number of tokens assigned to the i -th expert, and $\overline{\text{Load}}_i$ represents the expected load per expert assuming perfect balance.

To measure the diversity of token assignments across experts, we also compute the *Entropy* of the average selection probabilities for each expert:

$$H = - \sum_{i=1}^{N_r} \bar{p}_i \log \bar{p}_i,$$

where $\bar{p}_i$ is the average probability of selecting the i -th expert over all tokens in the evaluation batch, and N_r is the total number of experts. A higher entropy indicates a more uniform distribution of tokens among experts, reflecting balanced and diverse routing decisions.

I.3 Extended Evaluation Results of Router Designs

The results presented in Table 13 indicate that although both the auxiliary router and auxiliary loss methods enhance sampling accuracy, they are also associated with high dead token ratios. In particular, some auxiliary router variants exhibit dead token ratios as high as 66.7%, suggesting that the router always selects tokens from the same positions across inputs, reflecting a positional bias. Notably, employing a linear router architecture in conjunction with auxiliary loss effectively reduces the dead token ratio without compromising sampling accuracy.

Results from Table 14 reveal that applying an explicit balancing loss significantly reduces MaxVio and increases entropy, leading to improved load balance without sacrificing overall model performance. Loss-free approaches, while simpler, tend to show higher MaxVio and lower entropy, indicating less balanced token routing. Architectures such as MLP and Linear routers perform comparably under balancing loss, with z-loss often contributing to improved routing stability and model accuracy. Nevertheless, it still struggles to achieve balance during quite long initial stage. The heterogeneity among the experts, stemming from the use of computation blocks with varying recursion depths as experts, likely complicates load balancing.

Table 13: Ablation results on using expert-choice router with different routing configurations. We use the recursion-wise KV caching strategy by default. Coeff denotes coefficient values for auxiliary loss term, and α denotes scaling term after router function. Dead token ratio are measured within evaluation batch size of 500. Warmup refers to gradually decreasing the capacity from 1.0 to desired value over warmup steps. The last highlighted row represents a chosen final strategy, and intermediate best-performing designs (based on performance and routing metrics) are highlighted to illustrate how it was derived.

Expert-choice Configurations							Router Metrics		NLL↓	Few-shot Accuracy↑							
Sampling	Coeff	Func	α	Arch	Warmup	z-loss	Dead↓	Samp-Acc↑	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg	
-	-	rand	-	-	✗	✗	0.0	-	2.9335	26.0	33.1	61.6	52.3	35.8	26.2	39.1	
Aux Router	-	-	-	MLP	✗	✗	66.7	50.0	NaN	0.0	25.04	49.5	49.6	23.9	23.0	28.5	
Aux Router	-	σ	0.1	MLP	✗	✗	0.0	89.2	2.8893	26.1	33.8	62.0	51.5	36.6	26.4	39.4	
Aux Router	-	σ	1.0	MLP	✗	✗	66.7	50.0	2.8867	26.4	33.6	63.0	52.4	37.0	24.1	39.8	
Aux Router	-	tanh	0.1	MLP	✗	✗	66.7	98.6	2.8720	13.9	31.8	60.7	49.3	35.8	25.8	36.2	
Aux Router	-	tanh	1.0	MLP	✗	✗	66.7	97.0	3.0624	18.26	29.7	60.1	50.9	34.6	25.5	36.5	
Aux Loss	0.01	-	-	MLP	✗	✗	66.7	50.0	NaN	0.0	25.04	49.5	49.6	23.9	23.0	28.5	
Aux Loss	0.01	σ	0.1	MLP	✗	✗	0.0	99.6	2.8967	24.8	33.6	63.3	50.3	36.6	26.6	39.2	
Aux Loss	0.01	σ	1.0	MLP	✗	✗	65.9	100.0	2.9189	12.0	31.6	59.4	51.5	33.2	25.3	35.5	
Aux Loss	0.01	tanh	0.1	MLP	✗	✗	32.8	99.7	2.9426	23.5	32.4	62.4	49.8	35.6	26.0	38.3	
Aux Loss	0.01	tanh	1.0	MLP	✗	✗	0.0	98.8	3.2743	16.4	28.14	58.8	52.2	31.6	24.8	35.3	
Aux Loss	0.1	σ	0.1	MLP	✗	✗	0.0	99.8	3.0416	21.5	31.0	61.8	50.3	35.0	26.0	37.6	
Aux Loss	0.001	σ	0.1	MLP	✗	✗	0.0	99.1	2.8816	27.6	34.3	63.0	51.6	36.7	26.5	40.0	
Aux Loss	0.001	tanh	0.1	MLP	✗	✗	0.0	56.4	2.9933	25.0	32.3	61.5	51.5	36.6	26.0	38.8	
Aux Loss	0.001	σ	0.1	Linear	✗	✗	0.1	99.2	2.8667	27.4	34.6	63.2	51.5	37.2	26.5	40.1	
Aux Loss	0.001	σ	0.1	W-MLP	✗	✗	0.4	99.2	2.8716	27.8	33.9	62.4	49.9	36.3	26.3	39.4	
Aux Loss	0.001	σ	0.1	Linear	✓	✗	4.9	99.1	2.8744	26.0	33.9	62.0	51.2	36.1	26.1	39.2	
Aux Loss	0.001	σ	0.1	Linear	✗	✓	0.0	99.3	2.8824	26.9	34.0	63.8	52.3	36.8	26.4	40.0	

Table 14: Ablation results on token-choice router under different routing configurations. We use the recursion-wise KV caching strategy by default. Coeff denotes coefficient for balancing loss term, or updating coefficient (u) for loss-free algorithm. The last highlighted row represents a chosen final strategy, but we added z-loss back in with a small coefficient of 1e-3 since it often stabilizes load balancing. The intermediate best-performing designs (based on performance and routing metrics) are highlighted to illustrate how it was derived.

Token-choice Configurations						Router Metrics		NLL ↓	Few-shot Accuracy ↑						
Balancing	Coeff	Func	α	Arch	Z-loss	MaxVio ↓	Entropy ↑	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg
-	-	rand	-	-	✓	0.007	1.099	3.0268	24.8	32.0	61.4	52.2	35.5	26.1	38.7
Loss	0.1	soft	1.0	MLP	✓	0.200	1.076	3.0239	24.2	31.9	61.4	51.5	35.7	26.2	38.5
Loss	0.01	soft	1.0	MLP	✓	0.682	0.921	2.9118	28.0	33.3	62.8	49.7	36.4	26.2	39.4
Loss-free	0.01	soft	1.0	MLP	✓	1.788	0.297	2.9078	25.5	32.5	61.3	52.3	36.1	26.0	38.9
Loss-free	0.01	σ	0.1	MLP	✓	0.956	0.646	3.1144	21.8	29.8	60.3	51.6	34.0	25.7	37.2
Loss-free	0.01	σ	1.0	MLP	✓	0.918	0.749	3.0188	23.4	31.3	59.9	50.0	35.2	25.8	37.6
Loss-free	0.001	soft	1.0	MLP	✓	0.852	0.915	2.9081	25.8	33.6	62.8	50.6	37.5	26.7	39.5
Loss-free	0.001	σ	0.1	MLP	✓	1.281	0.551	2.9165	23.9	33.1	61.2	51.6	37.3	26.2	38.9
Loss-free	0.001	σ	1.0	MLP	✓	0.542	0.941	3.0188	24.9	32.0	61.9	51.4	35.5	25.9	38.6
Loss	0.1	soft	1.0	Linear	✓	0.492	0.960	2.9974	23.7	31.3	62.2	50.3	36.7	26.0	38.4
Loss	0.1	soft	1.0	W-MLP	✓	0.384	1.037	3.0293	25.3	31.5	62.2	51.2	36.4	26.3	38.8
Loss	0.1	soft	1.0	Linear	✗	0.266	1.056	2.9358	25.7	32.6	61.9	51.7	36.4	26.5	39.1

J Expanded Results of KV Cache Sharing Mechanism

J.1 Key Value Representation Trends in Recursive Transformers

Sharing KV caches across model depths has emerged as a promising approach to improve inference throughput in Vanilla Transformers [9]. This technique can reduce the memory footprint required for KV caches, enabling larger inference batch sizes. Significant speedups can be also achieved by skipping the KV projection and even prefill operations at shared depths, especially with Cycle strategy [85]. Due to the high degree of freedom in Vanilla models—where trainable parameters can be well optimized for shared caches—these models exhibit only marginal performance drops when KV caches are shared between adjacent layers. In contrast, Recursive Transformers have far fewer parameters available for being optimized to tied KV states. Nevertheless, we hypothesize that similar patterns may emerge between shared blocks. To investigate this, we decomposed the KV states from pretrained Recursive Transformers into magnitude and directional components.

As shown in Figure 7, the sharing of key and value projection layers across recursion depths leads to clear recursive patterns in the *magnitude* values. Although the magnitudes of hidden states tend to increase, the projection layers appear to be trained to produce similar signal sizes at corresponding depths within each recursion.

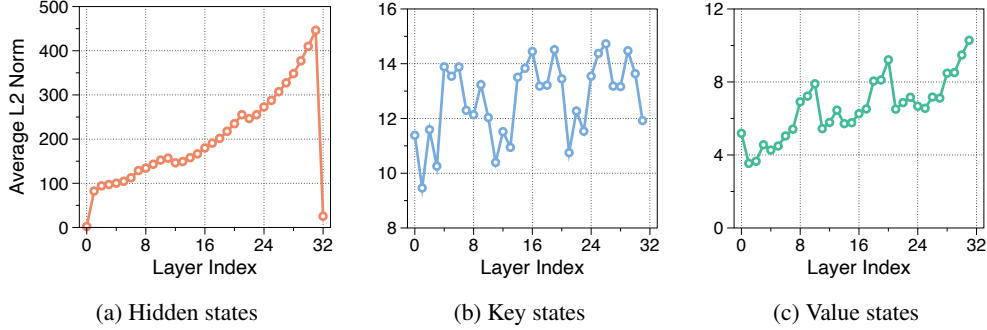


Figure 7: Average L2 norm magnitude of (a) hidden states, (b) key states, and (c) value states across layers in a Middle-Cycled Recursive Transformer 360M with 3 recursion steps. Note that the last hidden states correspond to the final hidden states after the last layer normalization.

When we measured the cosine similarity in Figure 8, distinct diagonal patterns emerge, suggesting that shared projection layers generate highly similar key and value representations. While sharing value states across recursions appears to be more challenging than sharing key states, these findings suggest that the performance drop from KV cache sharing can be marginal even in Recursive Transformers.

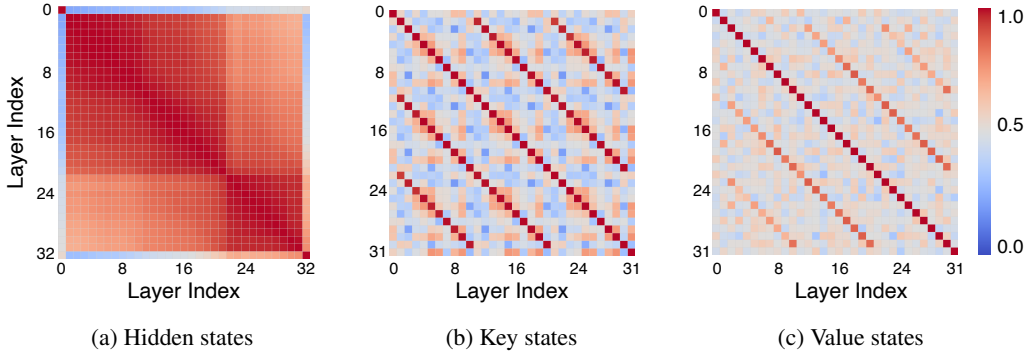


Figure 8: Cosine similarity matrices showing the layer-wise similarity of (a) hidden states, (b) key states, and (c) value states in Recursive Transformer with Middle-Cycle strategy and recursion depth 3. Results are from a 360M parameter model with 32 layers. The hidden states matrix includes the final hidden states after the last layer normalization.

J.2 Performance Comparison of KV Sharing Strategy

Experimental results of key-value cache sharing. In Table 15, we present the performance results when applying KV cache sharing to Vanilla, Recursive, and MoR models. Especially, we tested various strategies for KV caches, including Cycle or Sequence strategies that share the same concepts as parameter sharing (see §C.1 for details). Interestingly, KV cache sharing even improves the performance of vanilla models, where sharing acts as a regularization technique. In case of recursive models, we align the sharing strategy for parameters and KV caches. Despite some variations in the results after applying KV sharing, the Middle-Cycle strategy (the best parameter sharing strategy) showed a slight perplexity drop, albeit not substantial.

When moving to MoR models, they still introduced a small amount of degradation in our best settings (expert-choice router). However, considering the reduced parameter sizes and cache sizes, we believe this minor drop is acceptable. Furthermore, we explored an alternative sharing strategy (indicated by †) that utilized shared caches for inactive (unselected) positions while updating active positions through actual computation. This method is analogous to a recursive caching scheme but initializes inactive positions with key-value pairs from the first recursive iteration. Although it did not provide additional benefits, it is still worth exploring combinations of KV sharing and actual updates.

Table 15: Comparison of KV cache sharing strategies across Vanilla, Recursive, and MoR Transformers. Models are pretrained on 10B tokens of FineWeb-Edu, and evaluated using negative log-likelihood (NLL) on train set and few-shot accuracy across benchmarks. KV sharing denotes use of recursive KV sharing mechanism. If MoR is mentioned without further specification of KV sharing strategy, it implies the use of the recursion-wise caching strategy. †It indicates training with hybrid KV sharing that leverages shared caches for inactive positions while updating active ones through actual computation.

Models	Pretrain		Recursion		MoR	KV Sharing		NLL↓	Few-shot Accuracy↑						
	N-Emb	N_L	Share	Loop	Type	Share	Loop	FineWeb	LD	HS	PQ	WG	ARC	MLLU	Avg
Vanilla	315M	32	-	-	-	-	-	2.8471	27.3	34.8	64.2	52.8	38.3	26.7	40.7
Vanilla	315M	32	-	-	-	Seq	2	2.7848	30.0	36.5	64.6	50.7	39.4	26.9	41.3
Vanilla	315M	32	-	-	-	Cyc	2	2.7650	30.0	36.7	65.2	51.1	39.6	27.5	41.7
Vanilla	295M	30	-	-	-	-	-	2.8069	29.1	35.6	65.1	50.4	38.5	27.3	41.0
Vanilla	295M	30	-	-	-	Seq	3	2.7879	28.3	36.4	64.3	52.7	39.4	27.3	41.4
Vanilla	295M	30	-	-	-	Cyc	3	2.7890	28.9	36.5	64.6	51.4	39.0	27.6	41.3
Recursive	157M	16	Seq	2	-	-	-	2.9467	26.3	32.5	62.9	52.4	36.4	26.2	39.5
Recursive	157M	16	Seq	2	-	Seq	2	2.8904	26.4	33.4	64.0	51.0	37.0	26.9	39.8
Recursive	157M	16	Cyc	2	-	-	-	2.8487	28.5	34.8	63.1	50.0	37.4	28.8	40.1
Recursive	157M	16	Cyc	2	-	Cyc	2	2.8577	26.2	34.5	64.2	51.4	37.3	26.9	40.1
Recursive	167M	1+15+1	M-Cyc	2	-	-	-	2.8295	28.6	35.0	64.5	50.5	39.7	27.2	40.9
Recursive	167M	1+15+1	M-Cyc	2	-	M-Cyc	2	2.8451	27.3	34.7	63.7	50.5	37.8	27.0	40.2
Recursive	98M	10	Seq	3	-	-	-	3.0245	24.6	31.5	63.1	49.3	35.7	25.7	38.3
Recursive	98M	10	Seq	3	-	Seq	3	2.9554	24.2	32.3	62.5	52.7	36.6	26.2	39.1
Recursive	98M	10	Cyc	3	-	-	-	2.9363	25.9	33.0	62.9	50.3	36.4	26.5	39.2
Recursive	98M	10	Cyc	3	-	Cyc	3	2.9155	24.1	32.9	62.4	51.2	37.4	26.7	39.1
Recursive	118M	1+10+1	M-Cyc	3	-	-	-	2.8760	28.5	34.9	64.3	50.5	39.5	27.2	40.8
Recursive	118M	1+10+1	M-Cyc	3	-	M-Cyc	3	2.8854	27.3	33.8	63.3	52.3	37.5	26.8	40.2
MoR	118M	1+10+1	M-Cyc	3	Expert	-	-	2.8667	27.4	34.6	63.2	51.5	37.2	26.5	40.1
MoR	118M	1+10+1	M-Cyc	3	Expert	M-Cyc	3	2.8895	34.0	61.6	50.2	26.0	36.5	27.0	39.2
MoR	118M	1+10+1	M-Cyc	3	Expert	M-Cyc†	3	2.8653	24.8	34.3	62.0	50.1	36.7	26.7	39.1
MoR	118M	1+10+1	M-Cyc	3	Token	-	-	2.9358	25.7	32.6	61.9	51.7	36.4	26.5	39.1
MoR	118M	1+10+1	M-Cyc	3	Token	M-Cyc	3	2.9155	25.7	32.6	61.8	49.4	36.2	26.0	38.6

Relaxation for key-value sharing constraints. We also investigated relaxing the constraints on KV sharing in Table 16, similar to the relaxation approach in Bae et al. [6] for parameter sharing constraints. Specifically, we first re-examined four relaxation techniques for standard Recursive Transformers with very small ranks [42, 61] or prefix lengths [62]. We also experimented with the position encoding [12], where trainable embeddings are element-wise multiplied with the output of each recursion block.

Our results show that these techniques do not provide substantial performance improvements when pretraining relaxed models from scratch, consistent with prior studies, as they introduce only a limited number of additional parameters. Although we hypothesize that incorporating prefix-based approaches (such as adding trainable prefixes to attention) into KV sharing might lead to greater benefits, our experiments did not reveal substantial differences in this regard. Further exploration of more sophisticated techniques for efficiently relaxing KV cache sharing constraints remains an open direction for future research.

Table 16: Experimental results of relaxing parameter sharing and KV cache sharing constraints in Recursive Transformers. All models are trained on FineWeb-Edu with 10B tokens, and we apply the Middle-Cycle parameter sharing for 360M models with 3 recursion depths. We evaluate them based on training NLL and few-shot accuracy across six benchmarks. Relaxation types include encoding trainable embeddings on recursion outputs via element-wise multiplication (Enc), applying LoRA and DoRA to query and value weight matrices, and adaptation prompt tuning (Adapt-P).

	Pretrain		Relaxation			KV Sharing		NLL↓	Few-shot Accuracy↑						
Models	N-Emb	N_L	Type	Rank	Len	Share	Loop	FineWeb	LD	HS	PQ	WG	ARC	MMLU	Avg
Recursive	118M	1+10+1	-	-	-	-	-	2.8854	27.3	33.8	63.3	52.3	37.5	26.8	40.2
Recursive	118M	1+10+1	Enc	-	-	-	-	2.8604	27.3	34.6	63.9	53.4	38.6	26.7	40.2
Recursive	124M	1+10+1	LoRA	64	-	-	-	2.8599	27.3	34.6	64.3	50.9	38.0	26.9	39.7
Recursive	124M	1+10+1	DoRA	64	-	-	-	2.8945	26.4	33.6	64.4	50.6	37.4	26.5	39.2
Recursive	126M	1+10+1	Adapt-P	-	256	-	-	2.8626	27.1	34.7	64.0	51.9	37.6	26.8	39.7
Recursive	118M	1+10+1	-	-	-	M-Cyc	3	2.8854	27.3	33.8	63.3	52.3	37.5	26.8	40.2
Recursive	126M	1+10+1	Adapt-P	-	256	M-Cyc	3	2.9030	24.5	33.1	63.0	52.2	26.7	37.6	39.5

K Expanded Qualitative Results

K.1 Analysis on Adaptive Computation Paths

Table 17 illustrates a qualitative analysis of the recursion depth assigned to each subword token. This visualization provides a detailed insight into how tokens within each sample exhibit varying levels of recursive processing, showcasing the adaptive computation mechanism within the MoR framework. Notably, some tokens exit early (purple), while others require deeper processing (blue and red), reflecting the model’s ability to focus more compute on challenging parts of the input.

Table 17: Visualization of the recursion depth for each subword token, with colors representing the number of recursion steps: 1, 2, and 3. This depth specifically indicates how many times recursion is applied at that token’s position to predict the subsequent token. Each row corresponds to a single sample, offering a clear illustration of the token-level recursion distribution in practice. We use an MoR model with $N_r = 3$, auxiliary loss, and recursion-wise KV caching. This model is built on a 360M parameter base and trained on 30B tokens.

Sample	Text
Sample #1	People who feel comfortable defending their views — defensively confident — may also eventually change those views and corresponding behaviors. National Election Studies surveys showed that defensive confidence predicted defection in the 2006 U.S. House elections, above and beyond the impact of various demographic and political variables. Moreover, defensive confidence was also associated with political knowledge and attention to politics and government affairs, but not
Sample #2	2014, 7:57 AM Space X Falcon 9 - R Rocket Suffers Malfunction, Self-Destructs During Test Flight August 23, 2014, 9:36 AM Texas Chosen as Site for SpaceX’s First Commercial Launch pad August 5, 2014, 1:44 PM South Carolina Prison Finds Crashed Drone Carrying Drugs, Phones August 1, 2014, 2:49 PM NASA’s Mars 2020 Rover Gains Seven New Instruments for Exploration August 1, 2014, 1:30 PM NASA
Sample #3	9 PR Newswire. All rights reserved A report released Thursday on the Slide Fire in Oak Creek Canyon doesn’t hold back in laying out the danger facing the burned-out area. The U.S. Forest Service issued the report Thursday afternoon stating the biggest concern is the steeper slopes of Oak Creek Canyon is especially subject to debris flows, rockslides, flash flooding and erosion that could become concentrated flow or a landslide. The report said in smaller streams
Sample #4	Bilbo to accompany the dwarves to fight the enemy. He says, “Saruman believes it is only great power that can hold evil in check, but that is not what I have found. I found it is the small everyday deeds of ordinary folk that keep the darkness at bay. Small acts of kindness and love.” That’s what Jesus teaches us as well. Warning us that we would live in dark times, He reminded us that because of Him we are “the light of the world” (Matt. 5:14)
Sample #5	mitting Plants After four years of research, most of it in total darkness, a Stanford University plant biologist has discovered that some plants have a system of fiber optics that can transmit light up and down their tissues in a way similar to the method the telephone company uses to transmit many of its telephone calls. Dr. Dina Mandoli, seeking to find the light-transmitting properties of plants, had to rely on her sense of touch as she placed tissues of oat, mung
Sample #6	an impression of market power abuses or other market failures. In some cases, however, prices may spike to higher than usual levels and cause public concern and the need for more public information. To address such events, the proposed amendment includes an event trigger that would require the public release of entity-specific information on a much quicker timeframe. The proposed amendment requires that, when the trigger is exceeded, the portion of every
Sample #7	and spaceflight and spacecraft. covers elementary astronomy, Newtonian mechanics, the Sun and related physics and spaceflight. Also included are a Spanish translation, 46 lesson plans, a short but complete math course (algebra + trig), teachers’ guides, glossary, timelines, 345 questions (current tally) by users and their answers, over 100 problems to solve, and more. Learning Design Engines as Remote Control to Learning Support Environments. Context
Sample #8	a well established role in scientific computing, and a recent increased presence in desktop computing, it almost certain that contemporary information professionals will encounter Unix based systems in their work. This workshop is an intermediate level look at the Unix operating system as compared to the Unix introduction that our S401 students receive. We will have a short review but it will be essential to have the S401-UNIX material internalized as much as
Sample #9	code and will serve as a good introduction to the syntax necessary for creating shell programs. We will continue by discussing the importance of quoting now that we have used more powerful metacharacters (variable and command substitution) and need to control their expansion (interpretation). Finally, we will end our day revisiting the Unix find utility and looking at some advanced uses for this indispensable tool. Slides for Day 5 (PDF) A program running in Unix

K.2 Analysis on Router Weights

To gain insight into the router output distributions, we visualized the results in Figure 9. Our analysis reveals that various routing mechanisms are optimized to balance expert loads according to the desired capacity. Notably, expert-choice routers achieved nearly perfect load balancing with the auxiliary loss, resulting in almost binary values (1 or 0) for selected and unselected tokens, respectively. For the auxiliary router, it was able to distinguish between selected and unselected tokens to some extent, but still showed overlapping. Since this strategy allows for different capacity factors during training and inference⁶, further research into methodologies that can more distinctly separate inter-cluster variations seems necessary.

Other token-choice strategies also exhibited good balancing properties with reasonable router values, which are used to refine the outputs of the corresponding recursion computation blocks. However, most cases failed to converge to optimal load balancing (i.e., these were edge cases where they achieved their own optimal load balancing), highlighting the challenges of achieving consistent performance in heterogeneous expert settings.

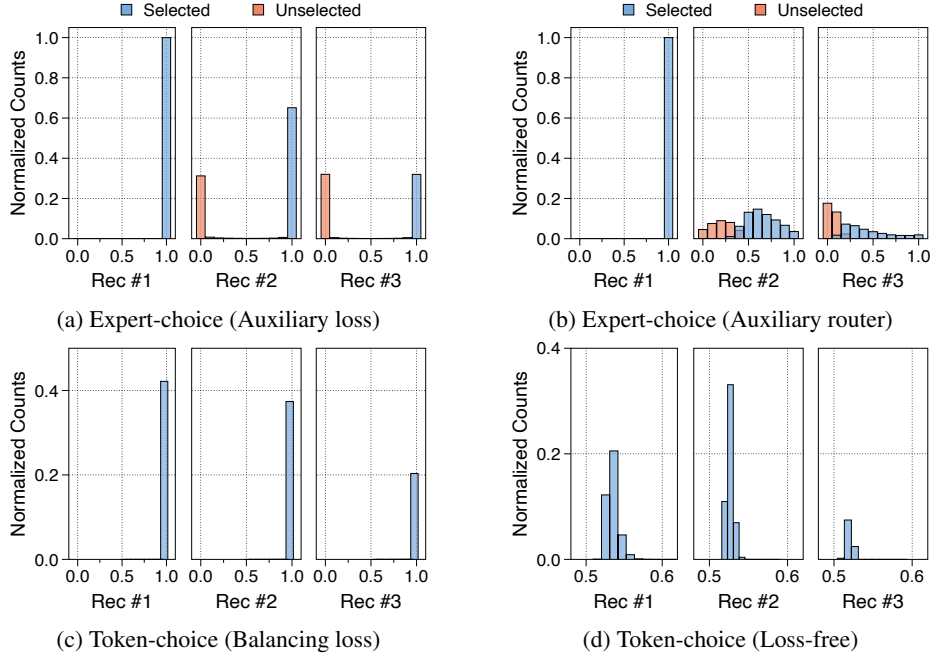


Figure 9: Distribution of router weights for selected and unselected tokens at each recursion step in expert-choice and token-choice MoR ($N_r=3$). All models use the recursion-wise KV caching strategy. The subplots show results for (a) expert-choice routing with auxiliary loss, (b) auxiliary router, (c) token-choice routing with balancing loss, and (d) loss-free algorithm. Each subplot uses the best hyperparameter settings identified in Table 4.

⁶The auxiliary router learns to capture intra-token differences within selected or unselected groups, rather than being biased towards extreme points.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: Abstract and Introduction sufficiently contain our main contributions: presenting Mixture-of-Recursions that unifies parameter sharing, token-level adaptive depth, and memory-efficient KV caching, within a single framework.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: In Appendix B, we provide a detailed discussion of the limitations of our work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: We do not include any theoretical results in this paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provided detailed instructions of our new Mixture-of-Recursion (MoR) architecture in the paper.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We rely on accessible, well-known, and publicly available model architecture and datasets. Code for both training and evaluation will be released upon publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Appendix D contain all the experimental settings and details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Our work involves pretraining models from scratch using a fixed data order, which minimizes sources of randomness. Therefore, there is little to no stochastic variation in our experiments that would necessitate reporting error bars or statistical significance measures.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: In Appendix D, we elaborate the details on compute resource we have used for the experiments and report the elapsed time.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: We have reviewed the NeurIPS Code of Ethics and couldn't identify any violation of the code of ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: This study does not invoke considerable societal concerns.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper does not contain such risks for data/model misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: We follow the conventional LLM evaluation protocols for every experiment and dataset, which we assume are properly cited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: We do not provide new assets in this research.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: We do not deal with crowdsourcing nor human subjects in this research.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: We do not deal with crowdsourcing nor human subjects in this research.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: We do not involve LLMs for the core method development in this research.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.