YET ANOTHER SCALING AXIS WITH SOME FREE LUNCH: ENLARGING TOKEN-INDEXED PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

The scaling laws of large language models have driven remarkable progress, yet scaling these parameters is compute-intensive and the performance gains gradually diminish. In this paper, we show a new scaling axis, which involves our proposed token-indexed parameters, can effectively provide new space for boosting the model capacity with negligible computational cost especially at inference time. We then propose our new architecture: ReToken, and its MoE version, Mixture of ReToken (MoRT), whereby the transformer layers are augmented with token-specific modulation vectors retrieved from learned embedding tables. These vectors are applied through table lookups and element-wise operations, adding negligible FLOPs compared to the backbone's $O(d^2)$ complexity per token. Across dense and MoE backbones (190M–3B parameters), our method consistently reduces training loss and substantially improves downstream task accuracy with virtually unchanged inference FLOPs and latency. Our scaling law analysis reveals that MoRT introduces a multiplicative improvement factor to the loss function, fundamentally shifting the quality-compute Pareto frontier, achieving equivalent model quality with 20-30% less compute compared to baseline MoE models. These results verify the token-indexed parameters as a new scaling axis.

1 INTRODUCTION

The scaling laws of large language models (LLMs), or specifically of the transformer architectures [37] have significantly advanced the fast development of LLMs, whereby the increase of model parameters can boost the performance. However simply adding parameters can cause near-linearly [21; 14] computing cost growth, which can be economically and operationally unaffordable even to giant companies, especially considering its performance gains are diminishing.

Departing from the notion of existing model parameters, we show an orthogonal scaling axis by introducing a new set of parameters: the so-called token-indexed parameters with the advantage of inuring ignorable computational cost. By design, these parameters increase model expressivity without adding GEMM FLOPs. Unlike MoE (which adds routed expert computation) [32; 8], retrieval/memory layers (which add similarity search and lookups into large tables) [23; 17], or adapter/LoRA paths (which add extra linear ops) [27; 15], token-indexed parameters are retrieved via table lookups and applied through element-wise operations—adding only $O(d)$ operations compared to the backbone's $O(d^2)$ complexity per token. This property enables models to grow substantially in parameter count while maintaining almost unchanged inference FLOPs and latency.

We develop our dense architectures: ReToken and its MoE version: Mixture of ReToken (MoRT). ReToken augments each transformer layer with a token-specific modulation vector retrieved from learned embedding tables, scaling the MLP residuals [11] via Hadamard products. MoRT extends this by maintaining multiple modulators per token and using a lightweight router to select context-appropriate mixtures. Both architectures avoid introducing new GEMMs—they rely entirely on lookups, small routers, and element-wise operations readily fused with existing kernel epilogues.

Extensive experiments empirically validate the effectiveness and scalability of the proposed token-indexed parameter scaling law. Specifically, across dense models (190M–1.5B) and MoE architectures (1.3B–3B total parameters), ReToken and MoRT consistently reduce training loss and improve downstream task performance — with average gains of 4 points on benchmarks — while keeping inference latency virtually unchanged. Most significantly, our scaling law analysis reveals that

MoRT fundamentally shifts the quality-compute Pareto frontier: at fixed compute budgets, MoRT models achieve the same loss as baseline MoE models with 20-30% less compute, establishing token-indexed parameters as a highly efficient scaling dimension. **We make the following contributions:**

- For the first time to our best knowledge, we introduce the so-called token-indexed parameters. These parameters technically decouples model capacity from FLOPs, and can be seamlessly integrated into existing transformer architectures.
- We propose ReToken and MoRT, practical architectures that leverage token-indexed modulation to enhance both dense and MoE models with negligible computational overhead through careful engineering of lookups and kernel fusion.
- We demonstrate consistent improvements across scales: 50%+ training efficiency gains in dense models, substantial downstream task improvements (4% absolute accuracy gain on average), and 20-30% compute save in scaling law analysis.
- We provide comprehensive implementation strategies for efficient training (embedding parallel) and inference (CPU offloading, cross-layer connections) that make billion-scale token-indexed parameters practical on commodity hardware.

2 Related Works

Scaling Law. The Kaplan scaling law [21] shows that language model performance follows smooth power-law trends with parameters, data, and compute, enabling extrapolation across orders of magnitude. Chinchilla [14] revises the compute-optimal recipe: under a fixed FLOPs budget, parameters and training tokens should be scaled proportionally, producing smaller-but-better-trained models that outperform much larger undertrained ones and reduce downstream costs. Extensions that account for inference demand [31] suggest that at high request volumes it is optimal to train smaller models for longer, and caution that naïvely fitted laws may overestimate token benefits outside typical regimes. For sparse architectures, fine-grained MoE scaling [22] introduces expert granularity and reports that MoE outperforms dense models at matched compute across studied scales, with the gap widening as scale grows, while FFN-sized experts are rarely optimal compared to many smaller experts. Overall, these results motivate jointly optimizing parameters, tokens, inference load, sparsity, and vocabulary—rather than treating scaling as one dimensional.

Vocabulary Scaling. Beyond scaling parameters and data, it emerges as another factor for quality and efficiency. [36] provides a compute-optimal rule indicating that larger models benefit from larger vocabularies and show fixed-FLOPs gains. in continual-training scenarios, [35] replacing the old vocabulary with a better-matched one outperforms keeping the original tokenizer. [16] decouples input/output vocabularies to enlarge only the input side with no extra inference cost; [10] upgrades the tokenizer and retrains only the input embedding and LM head, yielding shorter sequences and faster decoding at comparable quality. **See Appendix A for more related works.**

Distinction from our approach: Specifically, let the vocabulary embedding matrix have shape $[V, d]$, the prior works primarily scale along the V dimension [16; 40]. By contrast, we scale along a d dimension. While n-gram expansion faces combinatorial limits and tends to capture local and fixed patterns, enlarging d provides a high-dimensional, context-interactive space in which tokens can acquire richer semantics through attention-mediated interactions during training.

3 Methodology

In this section, we first introduce ReToken (Sec. 3.1), which augments transformer layers with token-specific modulation vectors. We then describe Mixture of ReToken (MoRT) (Sec. 3.2), which uses a lightweight router to select context-aware mixtures. Subsequently, we detail the efficiency optimizations enabling scaling with negligible overhead (Sec. 3.3) and provide a scaling law analysis (Sec. 3.4) to formalize the benefits of our new scaling axis.

3.1 ReToken: Enlarging the Embedding along d

Mechanism. ReToken scales along the embedding channel dimension d. Concretely, every transformer layer ℓ keeps a learned table $\mathbf{E}^\ell \in \mathbb{R}^{V\times d}$, so that each token id $x \in [V]$ retrieves a vector $\mathbf{E}^\ell[x]$ that gates the backbone's residual updates via light Hadamard products. This enlarges token-wise parameters without any new matrix multiplications.

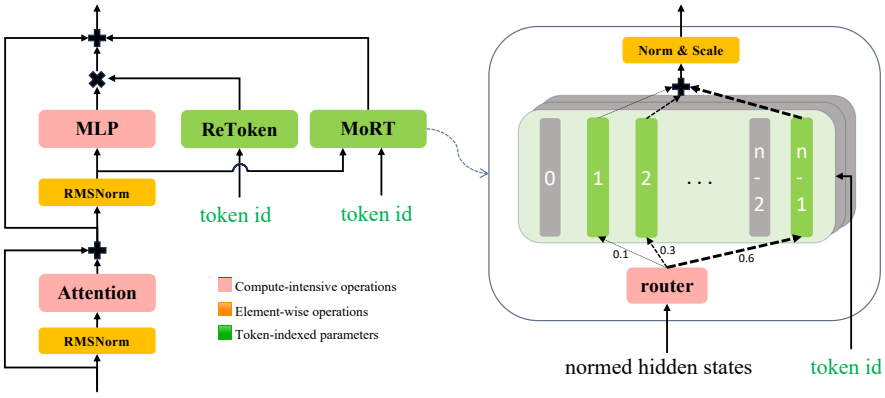


Figure 1: **Architecture overview.** (a) ReToken augments each transformer layer with token-specific modulation vectors retrieved from learned embedding tables and applied via element-wise multiplication. (b) MoRT extends this design with lightweight routing to select context-appropriate mixtures of modulators. Both architectures decouple parameter scaling from computational cost by relying on table lookups and element-wise operations rather than additional GEMMs.

Formally, Let x be the token id, $\Delta\mathbf{m}_x^\ell$ be the MLP module increments (before adding to the residual stream) of token x in layer ℓ. ReToken forms multiplicative gate $\mathbf{p}_x^\ell \in \mathbb{R}^d$:

$$\mathbf{p}_x^\ell = \mathbf{1} + \mathbf{s}^\ell \odot \mathrm{Norm}_\varepsilon(E^\ell[x]), \tag{1}$$

where $s^\ell \in \mathbb{R}^d$ is a learned per-dimension scaler and $\mathrm{Norm}_\varepsilon(u) = \frac{u}{\|u\|_2+\varepsilon}$($\varepsilon$ is a small constant to avoid division by zero) [42]. The gated MLP increments is:

$$\widetilde{\Delta\mathbf{m}_x^\ell} = \Delta\mathbf{m}_x^\ell \odot \mathbf{p}_x^\ell \tag{2}$$

FLOPs Analysis. For each token and transformer layer, ReToken adds approximately $4d$ extra FLOPs. Since the backbone model's dominant cost is GEMMs with $O(d^2)$ complexity, the $O(d)$ epilogue work is negligible; after epilogue fusion many profilers dot not even count it separately.

3.2 Mixture of ReToken (MoRT)

While ReToken already offers a capacity boost by attaching a single token-indexed modulator per layer, MoRT generalizes this idea to capture additional headroom and context variability without introducing non-trivial GEMM FLOPs (see Figure 1(right) for the complete architecture).

The key insight of MoRT is that token representation can further benefit from contexual features. MoRT thus equips each token with a small pool of modulators and selects a tiny, context-relevant subset at runtime, keeping the data path bandwidth-light and inference latency essentially unchanged, while retaining ReToken's spirit of decoupling parameter growth from activation compute.

Mechanism. Each token is equipped with a pool of n_e modulators per layer and uses a lightweight router to pick top-K of them given the current hidden state. Crucially, selection and application remain table lookups plus element-wise ops.

Formally, for token x, let $\mathbf{h}_x^\ell \in \mathbb{R}^d$ be the pre-MLP hidden state (after RMSNorm) in layer ℓ. Each layer maintains a pool $\{\mathbf{E}_i^\ell \in \mathbb{R}^{V\times d}\}_{i=1}^{n_e}$. A linear router computes logits

$$\mathbf{g}_x^\ell = (\mathbf{h}_x^\ell)^\top \mathbf{R}^\ell \in \mathbb{R}^{n_e}, \quad \mathbf{R}^\ell \in \mathbb{R}^{d\times n_e}, \tag{3}$$

selects $G_x^\ell = \mathrm{TopK}(\mathbf{g}_x^\ell, K)$, and forms normalized weights $w_i^\ell = \frac{\sigma(g_i^\ell)}{\sum_{j\in G_x^\ell}\sigma(g_j^\ell)}$ for $i \in G_x^\ell$ (with a sigmoid σ instead of softmax to avoid probability-simplex competition and mitigate expert collapse [26]). The mixed token-indexed vector is:

$$\widehat{\mathbf{e}}_x^\ell = \sum_{i\in G_x^\ell} w_i^\ell \, \mathbf{E}_i^\ell[x] \in \mathbb{R}^d. \tag{4}$$

We normalize and apply a learned per-dimension scale $\mathbf{s}_{\mathrm{MoRT}}^\ell \in \mathbb{R}^d$, producing the additive residual injection. To ensure the variance of the hidden states controllable [28], we scale down with an extra

factor $\frac{1}{\sqrt{2N_l}}$, where N_l is the number of layers in the backbone model. The ablation study of this down-scaling factor is provided in Appendix I.

$$\Delta \mathbf{r}^{\ell}_{x,\mathrm{MoRT}} \;=\; \frac{1}{\sqrt{2N_l}} \cdot \mathbf{s}^{\ell}_{\mathrm{MoRT}} \odot \mathrm{Norm}_{\varepsilon}\big(\widehat{\mathbf{e}}^{\ell}_{x}\big), \tag{5}$$

Finally, $\Delta \mathbf{r}^{\ell}_{t,\mathrm{MoRT}}$ is fused into the layer write-back together with MLP output $\Delta \mathbf{m}^{\ell}_{x}$:

$$\mathbf{h}^{\ell}_{x} \;\leftarrow\; \mathbf{h}^{\ell}_{x} \;+\; \Delta \mathbf{m}^{\ell}_{x} \;+\; \Delta \mathbf{r}^{\ell}_{x,\mathrm{MoRT}} \tag{6}$$

MoRT generalizes ReToken while keeping the "no new large matmuls" property: it introduces context-conditioned, top-K additive residual modulation using only table lookups, a small router, and element-wise ops. Unlike MoE, MoRT does not invoke new FFN experts; it remains orthogonal to MLP scheduling and is compatible with both dense and MoE backbones.

Training with Load Balancing. To encourage all the embedding experts are adequately utilized and trained, we incorporate an auxiliary load-balancing loss during training, similar to standard practice in MoE models. Details of the load-balancing formulation are provided in Appendix E.

FLOPs analysis. Per token and per layer, MoRT adds three terms: (i) router logits $O(dn_e)$ from a single $d \times n_e$ matrix multiplication; (ii) top-K mixture $O(dK)$; (iii) normalization+scaling+residual add $O(d)$ (the L2 norm is $O(d)$ with a small constant). In total:

$$\mathrm{FLOPs}_{\mathrm{MoRT}} \;\approx\; c_1\, dn_e \;+\; c_2\, dK \;+\; c_3\, d, \tag{7}$$

where c_1, c_2, c_3 are small constants. Compared to the backbone MLP's dominant cost $\Theta(d^2)$ per token , MoRT's overhead is negligible.

3.3 Implementation for Practical Efficiency

The large number of token-indexed parameters in ReToken and MoRT necessitates efficient implementation strategies to maintain negligible overhead.

3.3.1 Efficient Inference for ReToken

Precomputation and Kernel Fusion. At inference, for each token in each layer, we precompute

$$\mathbf{P}^{\ell}[x] \triangleq \mathbf{1} + \mathbf{s}^{\ell} \odot \mathrm{Norm}_{\varepsilon}\big(\mathbf{E}^{\ell}[x]\big) \tag{8}$$

and cache $\mathbf{P}^{\ell} \in \mathbb{R}^{V \times d}$ as a look-up table on device. This removes online normalization and scaling, reducing ReToken to a single table lookup. We then fuse this element-wise modulation with the epilogues of MLP kernels (the same stage that applies bias/dropout/addition). This avoids extra memory round-trips and kernel launches, hiding the lookup latency behind the GEMM pipeline.

Decoupling and Asynchronous Prefetch. The ReToken module is entirely decoupled from the backbone topology, as its modulation vector $\mathbf{p}^{\ell}_{x}$ depends solely on the token id x. This allows vectors to be fetched ahead of time. To reduce HBM usage, we store $\{\mathbf{P}^{\ell}\}$ in CPU memory and asynchronously prefetch only the rows required by the current batch while the GPU executes GEMMs.

3.3.2 Efficient Inference for MoRT

Offload MoRT to CPU. The massive parameter count of MoRT's embedding pools presents a VRAM challenge. However, since these parameters are not involved in GEMM computations, they are ideal candidates for offloading to CPU memory. This strategy makes it possible to scale token-indexed parameters to sizes far exceeding the backbone model without consuming precious on-device VRAM. The inference process is orchestrated as follows:

1. Initially, MoRT embedding pools $\{\mathbf{E}^{\ell}_{i}\}_{i=1}^{n_e}$ for all layers are loaded into CPU memory.
2. For each token in a given layer ℓ, the router computes the Top-K indices G^{ℓ}_{x} on GPU.
3. The GPU sends a compact request containing only these indices to the CPU.
4. The CPU fetches the corresponding K embedding vectors $\{\mathbf{E}^{\ell}_{i}[x]\}_{i \in G^{\ell}_{x}}$ from its main memory and transfers only this small subset of data to the GPU's VRAM.
5. Once the vectors arrive on the GPU, the remaining computations (weighted sum, normalization, scaling, and residual addition) are executed.

Additional Optimizations. Beyond the aforementioned inference optimizations, we have developed two additional techniques to enhance the practical deployment of our approach. First, we design an *Embedding Parallel* strategy that significantly improves training efficiency by distributing the massive token-indexed parameters across multiple devices during the training phase. Second, we introduce *cross-layer residual connections* that enable more flexible deployment across diverse hardware configurations. Due to space limitations, we defer the detailed descriptions of these techniques to Appendix B and Appendix C, respectively.

3.4 Scaling Law Framework for Token-Indexed Parameters

Neural scaling laws provide an empirical-yet-principled lens to reason about the quality-compute trade-off of LLM training. While the Chinchilla scaling law [14] is widely used, We find it does not explicitly model the interaction between data and model capacity. As under a fixed data increase, the loss reduction can be more pronounced for larger models than for smaller ones. We therefore adopt the Kaplan scaling law [21] as the basis of our analysis.

Kaplan scaling law. Kaplan scaling law model the test loss as

$$\mathcal{L}(N_c, D) = \left[\left(\frac{A}{N_c}\right)^{\frac{\alpha}{\beta}} + \frac{B}{D}\right]^{\beta}, \tag{9}$$

where N_c denotes the compute-intensive parameters (in MoE cases, it refers to activated parameters), D is the number of training tokens, and A, B, α, β are empirical constants.

Compute constraint and the optimal efficient frontier. Training a model with N_c parameters on D tokens requires about $6N_cD$ FLOPs. Define the compute-optimal loss at budget C as

$$\mathcal{L}^*(C) = \min_{N_cD=\frac{C}{6}} \mathcal{L}(N_c, D). \tag{10}$$

Plugging $D = \frac{C}{6N_c}$ into Eq. 9 and minimizing w.r.t. N_c yields a closed form:

$$\mathcal{L}^*(C) = \left(1 + \frac{\alpha}{\beta}\right)^{\beta} \left(\frac{6AB\beta}{\alpha}\right)^{\frac{\alpha\beta}{\alpha+\beta}} \cdot C^{-\frac{\alpha\beta}{\alpha+\beta}}. \tag{11}$$

Token-indexed parameters as a new scaling dimension. MoRT introduces token-indexed parameters N_n in addition to the compute-intensive backbone N_c. Let $\eta = N_n/N_c$ be the parameter expansion ratio. We model the contribution of N_n via a discount factor $\gamma(\rho)$ that depends on $\rho = K/n_e$ which captures the MoRT activation sparsity. We define an effective parameter count

$$N_{\text{eff}} = N_c + \gamma(\rho)N_n = N_c\big(1 + \gamma(\rho)\eta\big). \tag{12}$$

We then substitute N_{eff} for N_c in Eq. 9:

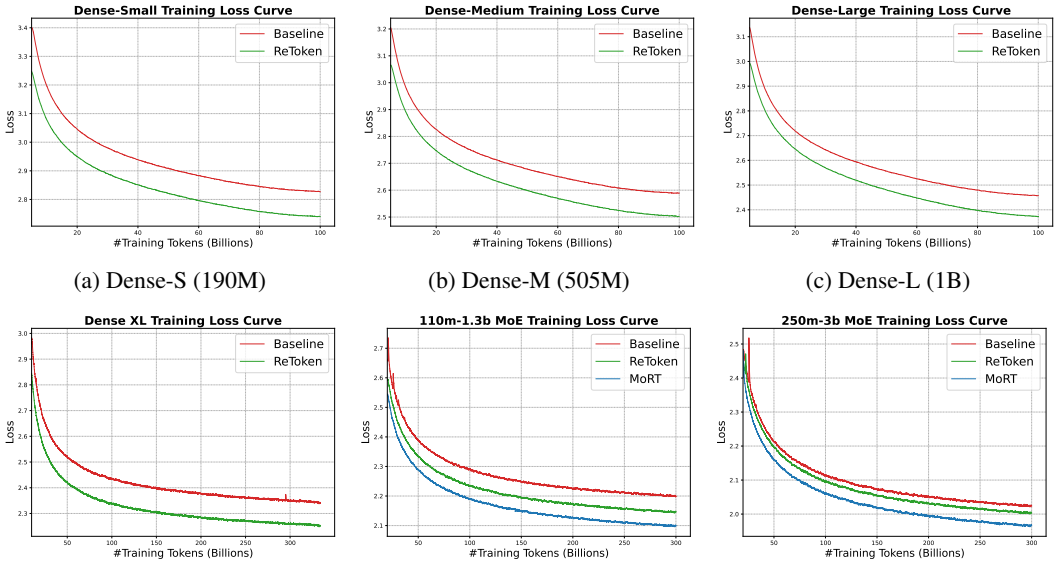
$$\mathcal{L}_{\text{MoRT}}(N_c, D; \eta, \rho) = \left[\left(\frac{A}{N_{\text{eff}}}\right)^{\frac{\alpha}{\beta}} + \frac{B}{D}\right]^{\beta} = \left[\left(\frac{A_{\text{MoRT}}}{N_c}\right)^{\frac{\alpha}{\beta}} + \frac{B}{D}\right]^{\beta},$$

where the effect is equivalent to rescaling the constant $A_{\text{MoRT}} = \frac{A}{1+\gamma(\rho)\eta}$. This formalizes the intuition that architecture changes from token-indexed parameters primarily manifest as a downward shift in the model-size term (via A), while leaving the data term (B) unchanged.

Implication on the efficient frontier: multiplicative downward shift. Under the same compute constraint C (note the budget depends on N_c and D, not on N_n), the MoRT compute-optimal frontier becomes

$$\mathcal{L}^*_{\text{MoRT}}(C; \eta, \rho) = \left(\frac{A_{\text{MoRT}}}{A}\right)^{\frac{\alpha\beta}{\alpha+\beta}} \mathcal{L}^*(C) = \underbrace{\big(1 + \gamma(\rho)\eta\big)^{-\frac{\alpha\beta}{\alpha+\beta}}}_{g(\eta,\rho)<1} \cdot \mathcal{L}^*(C). \tag{13}$$

Therefore, token-indexed parameters induce a multiplicative improvement on the efficient frontier: the slope $-\frac{\alpha\beta}{\alpha+\beta}$ remains unchanged, while the intercept shifts downward by $g(\eta, \rho)$, which appears as a parallel shift in log space.



(a) Dense-S (190M) (b) Dense-M (505M) (c) Dense-L (1B)

(d) Dense-XL (1.5B) (e) MoE (A110m-T1.3b) (f) MoE (A250m-T3b)

Figure 2: Training loss curves for dense and MoE models. The top row shows ReToken's performance on Dense-S, M, and L models. The bottom row shows results for Dense-XL, A110m-T1.3b MoE, and A250m-T3b MoE. In all settings, ReToken (and MoRT for MoE models) achieves a consistently lower training loss compared to the corresponding baseline model.

Compute saving interpretation. Eq. 13 further implies an isoperformance compute saving: to match a baseline frontier loss at compute C, MoRT needs

$$\mathcal{L}^*_{\text{MoRT}}(C_{\text{MoRT}}; \eta, \rho) = \mathcal{L}^*(C) \quad \Longrightarrow \quad C_{\text{MoRT}} = \frac{C}{1 + \gamma(\rho)\eta}. \tag{14}$$

Thus, the attainable compute reduction is controlled by the effective expansion factor $1 + \gamma(\rho)\eta$.

Isocompute validation protocol. Empirically, we validate Eq. 13 using standard isocompute methodology: (i) fit a baseline frontier $\mathcal{L}^*(C)$ by training multiple (N_c, D) pairs on fixed budgets C; (ii) at the same budgets, augment the optimal baseline with token-indexed parameters (controlled η and ρ) while keeping N_c and D unchanged; (iii) test whether the measured log-loss curves exhibit an approximately constant downward shift consistent with $g(\eta, \rho)$.

4 Experiments

4.1 Overall Effectiveness on Dense and MoE Models

We first try to validate that token-indexed parameters in general consistently improve performance across diverse architectures and scales, before further examining its scaling properties.

4.1.1 Experimental Setup

Backbone Models. We establish baselines using both dense and MoE LLM architectures. For dense models, we use four Qwen-style [39] models of varying sizes: 190M(S), 0.5B(M), 1B(L), and 1.5B(XL) parameters. For MoE models, we use two highly-sparse configurations: one with 110M activated parameters and 1.3B total parameters (A110m-T1.3b), and another with 250M activated parameters and 3B total parameters (A250m-T3b)[1]. Each MoE layer contains 145 experts, with one shared expert and 144 routed experts, from which the top-8 are activated per token. For more details about the model configurations, please refer to the appendix D.

ReToken/MoRT Configuration. We evaluate two methods: ReToken and MoRT. *Both are implemented as attached modules that augment the baseline architectures without modifying their backbone structure.* ReToken is benchmarked on all dense (S, M, L, XL) and both MoE models (A110m-T1.3b, A250m-T3b). MoRT is evaluated on the two MoE backbones. For MoRT, we set the number of modulator experts to $n_e = 5$ and activate the top $K = 3$.

[1]Both excludes tokenizer vocabulary embedding and prediction head parameters.

Table 1: Downstream task accuracy for Small, Medium, and Large dense models, comparing the baseline architecture against our ReToken-enhanced variant. All models were pre-trained on 100B tokens from the Fineweb-edu dataset. ReToken provides a consistent performance uplift across all evaluated tasks and model sizes. The best result of each task is highlighted in bold.

Model		Downstream Task Performance (Accuracy %)								
		arc_e	arc_c	openbookqa	boolq	winogrande	sciq	hellaswag	piqa	Avg.
Small (Dense)	Baseline	59.34	28.50	30.20	53.58	52.17	78.70	37.75	66.54	50.85
	ReToken	**60.48**	**31.06**	**31.4**	**61.53**	**52.17**	**79.40**	**39.96**	**66.81**	**52.85**
medium (Dense)	Baseline	66.79	33.79	35.40	55.07	53.67	88.00	47.47	69.15	56.17
	ReToken	**69.61**	**36.52**	**37.80**	**57.58**	**53.99**	**90.30**	**49.67**	**71.82**	**58.41**
Large (Dense)	Baseline	71.63	38.31	37.2	59.27	53.83	91.10	53.28	72.25	59.90
	ReToken	**73.74**	**41.72**	**38.6**	**60.85**	**56.11**	**91.70**	**54.91**	**73.88**	**61.44**

Datasets and Training. The S, M, and L dense models are pretrained for 100B tokens on the Fineweb-edu [25], an open-source collection of text dataset. The XL dense model and all MoE models are trained on a diverse, high-quality dataset curated from online corpora, which includes general text, code, math, and multilingual content after rigorous filtering. Specifically, the XL dense model is trained for 330B tokens, the A110m-T1.3b MoE model for 300B tokens, and the A250m-T3b MoE model for 500B tokens. To ensure a fair comparison, all training configurations for the ReToken and MoRT models were kept identical to those of their corresponding baselines. For more details about the data and training hyperparameters, see Appendix D.

Evaluation. Downstream performance is evaluated by two benchmark suites. For the dense models (S, M, L), we use 8 foundational language understanding tasks. For the larger XL dense and MoE models, we employ a framework with 4 areas: **Knowledge** (MMLU [12], TriviaQA [20], ARC [4], GPQA [29]), **Reasoning** (Hellaswag [41], C3 [33], BBH [34], SocialIQA [30]), **Code** (MBPP [1], HumanEval [3], LiveCodeBench [18]) and **Math** (MATH [13], GSM8K [5], DROP [7]).

4.1.2 Results and Analysis

Training Loss. As shown in Fig, 2, our proposed methods achieve a consistent and significant reduction in training loss across all model scales and architectures. For the dense models, ReToken consistently maintains a lower loss trajectory compared to the baseline models from 190M to 1.5B parameters. Similarly, for the MoE models, both ReToken and MoRT demonstrate a clear advantage over the strong baselines, reducing the training loss for both the 1.3B and 3B MoE configurations. This demonstrates that scaling token-indexed parameters is a robust method for improving model optimization and data fitting capabilities, without increasing computational complexity.

Downstream Performance. The benefits observed during training translate to robust improvements on downstream tasks. As shown in Table 1, for the S, M, and L dense models, ReToken consistently outperforms the baseline on every benchmark. This trend continues with the larger models, as detailed in Table 2 and Table 3. The ReToken-XL model achieves substantial improvements over its baseline, with notable gains across all four capability domains.

For MoE models, the improvements are even more pronounced with the addition of MoRT as shown in Table 2 and Table 3. On the A110m-T1.3b model, ReToken delivers substantial gains while MoRT achieves even stronger performance, with particularly impressive improvements in Knowledge and Code domains. The larger A250m-T3b model shows a similar tiered pattern: baseline performance is surpassed by ReToken, which is in turn outperformed by MoRT. MoRT's gains are especially notable in Knowledge, Reasoning, and Math tasks, demonstrating that the contextualized routing of multiple modulators provides additional benefits over single-modulator approaches. These consistent improvements across all model scales and architectures validate that token-indexed parameters offer a robust path to enhanced language model capabilities without computational overhead. **We note that the magnitude of these improvements varies across model architectures; a detailed discussion on this topic is provided in Appendix F.**

4.2 Overall Efficiency on Dense and MoE models

In this section, we discuss the efficiency of the ReToken and MoRT methods in term of latency, especially considering the CPU-offloading approach for MoRT.

Table 2: Performance evaluation on comprehensive downstream tasks (Part 1: Knowledge and Code). The table compares the Baseline, ReToken, and MoRT variants for XL Dense, A110m-T1.3b MoE, and A250m-T3b MoE models. The best result of each task is highlighted in bold.

Model		Knowledge					Code			
		MMLU	TriviaQA	ARC_C	GPQA	**Avg.**	MBPP	HumanEval	LCB	**Avg.**
XL (Dense)	Baseline	32.74	25.72	35.74	25.60	29.95	9.53	9.15	0.87	6.52
	ReToken	**37.29**	**35.22**	**41.58**	**29.80**	**35.97**	**12.20**	**9.76**	**2.61**	**8.19**
A110m-T1.3b (MoE)	Baseline	27.91	22.81	29.21	25.59	26.38	7.00	**10.57**	1.57	6.38
	ReToken	30.98	27.31	30.24	28.28	29.20	10.60	9.96	1.45	7.34
	MoRT	**34.07**	**36.53**	**34.72**	**30.03**	**33.84**	**12.50**	10.55	**1.64**	**8.23**
A250m-T3b (MoE)	Baseline	36.39	39.57	39.76	29.47	36.30	19.20	16.46	2.52	12.73
	ReToken	40.05	41.13	43.87	28.79	38.46	21.80	18.90	4.70	15.13
	MoRT	**43.47**	**46.29**	**47.01**	**31.70**	**42.12**	**24.51**	**19.89**	**5.96**	**16.79**

Table 3: Performance evaluation on comprehensive downstream tasks (Part 2: Reasoning and Math). The table compares the Baseline, ReToken, and MoRT variants for XL Dense, A110m-T1.3b MoE, and A250m-T3b MoE models. The best result of each task is highlighted in bold.

Model		Reasoning					Math			
		Hellaswag	C3	BBH	SocialIQA	**Avg.**	MATH	GSM8K	DROP	**Avg.**
XL (Dense)	Baseline	51.19	39.34	21.04	41.64	38.30	2.17	5.76	10.55	6.16
	ReToken	**55.07**	**47.83**	**24.12**	**45.97**	**43.25**	**3.36**	**9.07**	**17.71**	**10.05**
A110m-T1.3b (MoE)	Baseline	45.70	27.95	18.96	34.44	31.76	1.96	3.87	6.60	4.14
	ReToken	48.01	29.86	18.82	36.39	33.27	3.04	**4.17**	7.99	5.07
	MoRT	**52.06**	**30.22**	**20.66**	**38.51**	**35.36**	**4.37**	4.15	**8.90**	**5.81**
A250m-T3b (MoE)	Baseline	57.05	36.84	21.87	42.63	39.60	7.16	13.65	11.86	10.63
	ReToken	58.52	44.05	24.74	44.83	43.04	8.30	16.38	13.27	12.65
	MoRT	**60.37**	**49.01**	**28.58**	**47.15**	**46.28**	**10.42**	**19.96**	**18.44**	**16.27**

Setup. We conducted a series of benchmarks using the SGLang [43] on a single NVIDIA H800, comparing the following configs: (1) Dense 1.5B models with and without ReToken; (2) MoE A250m-T3B models with the baseline, ReToken, and MoRT (top $K = 2$, $n_e = 4$). To address the load-balancing requirements inherent to MoE models, we utilized the real dataset Pile [9] instead of random tokens. Each set of experiments was conducted for 100 runs per configuration to compute the average latency and variance. We focus on two key latency metrics: (1) TTFT (time to first token) which measures the time cost of prefill phase and (2) TPOT (time per output token) which measures the time taken for each token generated during the decoding phase.

Results. The quantitative latency results are shown in Table 4 and Table 5. As for TTFT, the prefill phase is compute-bound, and the extra memory accesses for ReToken/MoRT are largely hidden behind the backbone GEMMs. Across batch sizes 8/16 and sequence lengths 1k–4k, the overhead for ReToken is negligible, while MoRT introduces up to $\sim$10-11% overhead in the worst case. On the other hand, for TPOT, during the decoding phase, where memory bandwidth becomes more critical, CPU-offloaded token-indexed parameters introduce a moderate per-token latency. Specifically, ReToken increases TPOT by $\sim$12%, while MoRT increases TPOT by $\sim$18-19%.

Taken together, these results demonstrate that the overhead introduced by ReToken/MoRT is well-bounded during both the prefill and decoding phases. Given that ReToken/MoRT consistently improve training loss and downstream task accuracy while GPU memory usage remains essentially unchanged when token-indexed tables are CPU-offloaded, we consider this latency-efficiency trade-off reasonable for many deployment scenarios.

4.3 Studies for Scaling Law of MoRT

This section empirically validates our central hypothesis: scaling token-indexed via MoRT establishes a new, efficient scaling dimension that shifts the quality-compute Pareto frontier. We employ a rigorous isocompute methodology to quantify these gains.

4.3.1 Experimental Setup

We conduct a four-stage evaluation and analyze MoRT's scaling behavior.

Stage 1: Establishing the Baseline Efficient Frontier. We establish the efficient frontier for our baseline MoE. Following standard scaling law methodology, we select five compute budgets span-

Table 4: Comparison of TTFT (Time to First Token) on different batch size and sequence length for dense and MoE Models with ReToken and MoRT.

TTFT (ms)		Dense (1.5B)		MoE (A250m-T3B)		
		Baseline	ReToken	Baseline	ReToken	MoRT(2 of 4)
bs=8	seqlen=1k	37.75 ± 3.87	36.55 ± 5.19	36.39 ± 1.81	37.14 ± 1.78	40.17 ± 4.68
	seqlen=2k	55.21 ± 2.42	56.77 ± 2.48	45.39 ± 1.39	46.44 ± 1.63	47.70 ± 2.86
	seqlen=4k	114.09 ± 3.37	118.06 ± 4.16	89.82 ± 5.60	90.53 ± 7.67	95.90 ± 9.50
bs=16	seqlen=1k	60.70 ± 7.93	62.97 ± 7.52	62.73 ± 10.46	63.19 ± 8.49	66.60 ± 4.21
	seqlen=2k	114.34 ± 3.40	117.27 ± 2.82	89.28 ± 6.48	90.74 ± 6.55	98.15 ± 4.45
	seqlen=4k	220.99 ± 8.22	223.33 ± 8.82	180.19 ± 10.94	181.15 ± 8.58	187.14 ± 10.20

Table 5: Comparison of TPOT (Time Per Output Token) on different batch size and sequence length for dense and MoE Models with ReToken and MoRT.

TPOT (ms/token)		Dense (1.5B)		MoE (A250m-T3B)		
		Baseline	ReToken	Baseline	ReToken	MoRT(2 of 4)
bs=8	seqlen=1k	2.12 ± 0.02	2.25 ± 0.02	2.83 ± 0.03	3.06 ± 0.02	3.31 ± 0.05
	seqlen=2k	2.18 ± 0.02	2.31 ± 0.02	2.90 ± 0.02	3.13 ± 0.02	3.35 ± 0.03
	seqlen=4k	2.28 ± 0.02	2.41 ± 0.02	3.04 ± 0.04	3.22 ± 0.04	3.41 ± 0.03
bs=16	seqlen=1k	2.15 ± 0.05	2.45 ± 0.08	3.19 ± 0.03	3.42 ± 0.04	3.72 ± 0.08
	seqlen=2k	2.28 ± 0.03	2.56 ± 0.02	3.32 ± 0.08	3.69 ± 0.05	3.95 ± 0.06
	seqlen=4k	2.45 ± 0.02	2.75 ± 0.03	3.56 ± 0.16	3.95 ± 0.07	4.20 ± 0.09

ning an order of magnitude: 3.89e18, 6.11e18, 1.11e19, 1.89e19, 3.33e19 FLOPs. For each budget C_i, we conduct grid search over model size N_c and dataset size D pairs, training multiple configurations to identify the optimal (N_c^*, D^*) that minimizes test loss. Figure 3 illustrates these isocompute curves, where each nadir represents the compute-optimal configuration.

Stage 2: MoRT Scaling Validation. Using the five optimal baseline configurations (N_c^*, D^*), we train the corresponding MoRT models while maintaining identical compute budgets. For this primary evaluation, we set the parameter ratio $\eta = N_n/N_c = 30$ and activation sparsity $\rho = 1.0$. This controlled comparison isolates the effect of token-indexed parameters on the efficient frontier.

Stage 3: Parameter Ratio Sensitivity(η): To study how MoRT's efficient frontier shifts as the amount of token-indexed parameters changes, we fix the activation sparsity at $\rho = 1.0$ and, for each baseline-optimal configuration (N_c^*, D^*) on a given compute budget, train MoRT models with parameter ratios $\eta \in \{30, 50, 80\}$.

Stage 4: Activation Sparsity Sensitivity(ρ): To isolate the effect of activation sparsity on the efficient frontier, we fix $\eta = 80$ and, again at each baseline-optimal configuration (N_c^*, D^*), train MoRT models with activation ratios $\rho \in \{0.3, 0.6, 1.0\}$.

All models are trained on identical dataset and evaluated using held-out test loss.

4.3.2 Frontier Shift and Scaling-Law Validation

Our experiments show that MoRT fundamentally improves the scaling efficiency of LLMs.

Isocompute Analysis. Fig. 3 confirms that baseline MoE models exhibit the expected U-shaped isocompute curves, with clear optimal points forming the baseline efficient frontier.

Frontier Shift. Fig. 4 gives our primary finding: MoRT models (blue, $\eta = 30$, $\rho = 1.0$) establish a new efficient frontier below the baseline (red). This downward shift shows that MoRT achieves lower test loss at every compute budget—a fundamental improvement in scaling efficiency. Quantitatively, MoRT achieves the same loss as baselines with a 20% speedup.

A central prediction of Eq. 13 is that

$$\log \mathcal{L}^*_{\text{MoRT}}(C; \eta, \rho) = \log \mathcal{L}^*(C) + \log g(\eta, \rho) \tag{15}$$

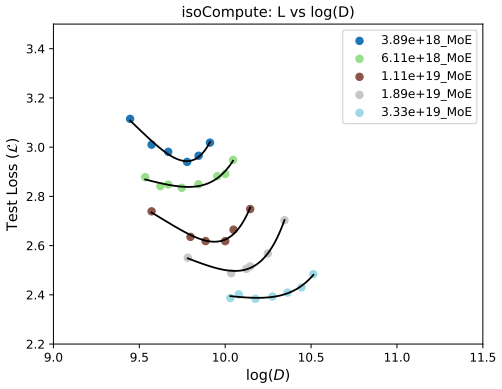


Figure 3: Isocompute for baseline MoE models with compute budgets. Each curve shows test loss vs. data size at fixed FLOPs, with optimal points (minima) defining the efficient frontier.

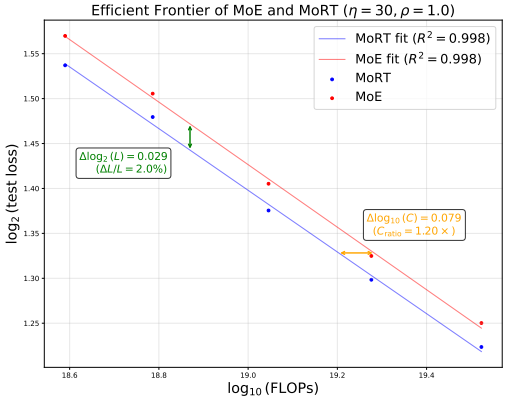


Figure 4: Efficient frontiers: MoRT (blue) vs. baseline MoE (red). It achieves 2.0% lower loss (vertical shift) at each compute budget, equivalent to 1.20× efficiency (horizontal shift).

Table 6: Effect of η on MoRT performance. Higher ratios of token-indexed to compute-intensive parameters yield better loss reduction and compute efficiency (fixed $\rho = 1.0$).

η	30	50	80
Loss Reduction	2.00%	2.42%	2.85%
Compute Ratio	1.20	1.25	1.29

Table 7: Effect of ρ on MoRT performance. Full activation provides best results but with diminishing returns (fixed $\eta = 80$).

ρ	0.3	0.6	1.0
Loss Reduction	2.70%	2.81%	2.85%
Compute Ratio	1.27	1.28	1.29

The baseline and MoRT frontiers in Fig. 4 are well-approximated by straight lines with near-identical slopes on log-log axes, and the gap between them is approximately constant across the tested budgets. Together, these observations empirically validate Eq. 15.

Furthermore, it's indicated that MoRT's performance gain is determined by its internal configuration (η and ρ), not the scale of the backbone model. This scale-invariant improvement confirms that token-indexed parameters constitute an orthogonal scaling axis. Its benefits compound with traditional scaling, offering a path to enhance model performance across compute budget.

4.3.3 SENSITIVITY ANALYSIS

Parameter Ratio (η). Table 6 shows that increasing the ratio of token-indexed to compute-intensive parameters yields consistent improvements. As η grows from 30 to 80, the relative loss reduction increases from 2.00% to 2.85%, with a corresponding improvement in compute efficiency from 1.20× to 1.29×. These gains show no signs of saturation within our tested range, suggesting that further scaling of token-indexed parameters remains promising.

Activation Sparsity (ρ). Table 7 reveals the impact of activation sparsity. While performance improves monotonically with ρ, we observe diminishing returns: the gain from $\rho = 0.3$ to $\rho = 0.6$ (0.11% absolute) exceeds that from $\rho = 0.6$ to $\rho = 1.0$ (0.04% absolute). This suggests an efficiency sweet spot where partial activation balances performance with memory bandwidth requirements.

5 CONCLUSION AND OUTLOOK

We have introduced token-indexed parameters as a new and complementary scaling axis. Our methods, ReToken and MoRT, consistently improve dense and MoE models with negligible inference overhead. A scaling-law analysis confirmed MoRT shifts the quality-compute Pareto frontier, achieving baseline performance with 20-30% compute save.

Limitations and Future Work. Due to resource limitations, our evaluation is currently limited to 3B model, which does not reach the frontier-scales. Moreover, since ReToken and MoRT introduce more embedding parameters, the VRAM during training will inevitably cause an overhead. Future work will consider 1) exploring additional probabilities in the design choices, and 2) extending to multimodal models with modality-specific parameter pools.

REFERENCES

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

[2] Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

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

[4] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

[6] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

[7] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

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

[9] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[10] Shuhao Gu, Mengdi Zhao, Bowen Zhang, Liangdong Wang, Jijie Li, and Guang Liu. Retok: Replacing tokenizer to enhance representation efficiency in large language model. *arXiv preprint arXiv:2410.04335*, 2024.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[12] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

[14] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models (2022). *arXiv preprint arXiv:2203.15556*, 2022.

[15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1 (2):3, 2022.

[16] Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling. *arXiv preprint arXiv:2501.16975*, 2025.

[17] Zihao Huang, Qiyang Min, Hongzhi Huang, Defa Zhu, Yutao Zeng, Ran Guo, and Xun Zhou. Ultra-sparse memory network. *arXiv preprint arXiv:2411.12364*, 2024.

[18] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[19] Shibo Jie, Yehui Tang, Kai Han, Yitong Li, Duyu Tang, Zhi-Hong Deng, and Yunhe Wang. Mixture of lookup experts. *arXiv preprint arXiv:2503.15798*, 2025.

[20] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[21] 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.

[22] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

[23] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

[24] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

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

[26] Huy Nguyen, Nhat Ho, and Alessandro Rinaldo. Sigmoid gating is more sample efficient than softmax gating in mixture of experts. *Advances in Neural Information Processing Systems*, 37: 118357–118388, 2024.

[27] Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. Adapters: A unified library for parameter-efficient and modular transfer learning. *arXiv preprint arXiv:2311.11077*, 2023.

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

[29] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[30] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[31] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws, 2024. *URL https://arxiv.org/abs/2401.00448*.

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

[33] Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. Investigating prior knowledge for challenging chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*, 8:141–155, 2020.

[34] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[35] Sho Takase, Ryokan Ri, Shun Kiyono, and Takuya Kato. Large vocabulary size improves large language models. *arXiv preprint arXiv:2406.16508*, 2024.

[36] Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. *Advances in Neural Information Processing Systems*, 37:114147–114179, 2024.

[37] 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.

[38] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024.

[39] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[40] Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling embedding layers in language models. *arXiv preprint arXiv:2502.01637*, 2025.

[41] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

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

[43] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A More Related Works

Mixture of Experts. MoE architectures have advanced toward sparser routing, finer-grained experts, and greater deployability. The sparsely-gated MoE of [32] activates only a small subset of FFN experts per example via a trainable gate with an auxiliary load-balancing loss, scaling capacity to tens of billions of parameters with limited extra compute and validating conditional computation for language modeling and translation. Switch Transformer [8] further simplifies routing by selecting a single expert per token, yielding a favorable speed–quality trade-off under fixed compute and enabling trillion-parameter scaling. [6; 24] pushes extreme expert specialization by partitioning experts more finely and isolating shared experts for generic knowledge, improving utilization and reducing redundancy, and reporting competitive or superior performance to larger dense/MoE baselines at comparable cost. To ease deployment and accelerate inference, [19] reparameterizes trained FFN experts into lookup tables for inference, retrieving retrieving precomputed expert outputs on demand to slash VRAM usage and avoid real-time expert compute, which yields dense-like latency in their setting.

Large Memory Layer. Large memory layers expand model capacity by decoupling parameters from compute. Product Key Memory [23] inserts a very large, sparsely accessed key–value table into the FFN, adding minimal extra FLOPs while improving language modeling at scale. Ultra-Sparse Memory [17] makes access even sparser and lighter—activating only a few slots per token—and reports favorable scaling behavior and lower inference latency under matched compute. Memory Layers at Scale [2] trains per-layer key–value lookups within Transformers, dedicating capacity for knowledge storage; under fixed compute budgets these memory-augmented models surpass dense baselines that spend substantially more compute and, at matched parameter counts, often compete with or outperform MoE. In contrast to retrieval-based memory, our modulators are token-indexed and act directly on the residual stream without similarity search or additional matmuls.

B Efficient Training: Embedding Parallel (EmP)

MoRT's parameters are large but sparsely accessed: in each layer and step we only touch K rows per token across the n_e tables. We exploit this with an Embedding Parallel (EmP) scheme that shards token-indexed tables and communicates only the touched rows.

Sharding layouts. Let each layer hold $\{\mathbf{E}_i^\ell \in \mathbb{R}^{V\times d}\}_{i=1}^{n_e}$. We consider:

- **V-sharding (vocab-parallel):** split the V dimension into S_V shards; rank (s) owns rows $\mathcal{V}_s$. Request for (x, i) is sent to the rank that owns x.
- **E-sharding (expert-parallel over modulators):** split the n_e dimension into S_E shards; rank (s) owns a subset $\mathcal{E}_s$. Request for (x, i) is sent to the rank owning i.
- **2D sharding (V×E):** partition by both vocab and modulator index, giving $S_V \times S_E$ shards; request is routed to the unique owner of (x, i).

All three avoid replicating the full tables on every rank.

EPa step (per layer).

1. **Local routing:** Each compute rank forms $G_x^\ell = \mathrm{TopK}(\mathbf{g}_x^\ell, K)$ locally.
2. **Row requests (all-to-allv):** Build per-destination lists of $(x, i, w_i^\ell, \text{token\_slot})$ and *deduplicate* repeated (x, i) within the step/batch.
3. **Remote gather & optional mixing:** Owner ranks fetch $E_i^\ell[x]$ for the requested pairs. Two options: (A) Return raw rows (K vectors of length d), or (B) Return pre-mixed$\widehat{\mathbf{e}}^\ell[x] = \sum_{i\in G_x^\ell} w_i^\ell \mathbf{E}_i^\ell[x]$ (one vector of length d). Option B halves bandwidth (from Kd to d) and moves a tiny Kd dot-product to the owner.
4. **All-to-allv return & residual add:** Source ranks receive rows (or the pre-mix), apply normalization+scale, and add $\Delta\mathbf{r}_{t,\mathrm{MoRT}}^\ell$ into the residual epilogue.

C Efficient inference: Cross-Layer Residual Connections.

Considering the complexities of heterogeneous hardware and real-world deployments, there may be cases where CPU-offloaded MoRT vectors fail to synchronize with the GPU's computation. In such cases, the backbone may stall while waiting for MoRT output, leading to non-trivial latency.

Mechanism. To make inference robust against such stalls, we introduce *cross-layer residual connection*. Instead of injecting the MoRT residual $\Delta\mathbf{r}^{\ell}_{t,\mathrm{MoRT}}$ immediately at the end of layer ℓ, we delay its application to a future layer, $\ell + l_d$:

$$\mathbf{h}_t^{\ell+l_d} \leftarrow \mathbf{h}_t^{\ell+l_d} + \Delta\mathbf{r}^{\ell}_{t,\mathrm{MoRT}} \tag{16}$$

This delay opens a window where the data transfer for layer-ℓ's vectors can occur in parallel with the GPU's execution of GEMMs for layers $\ell + 1, \ldots, \ell + l_d$. By choosing an appropriate delay l_d based on the deployment hardware, the communication latency can be largely hidden, making the capacity expansion from MoRT virtually free in terms of inference latency.

Effect on model performance. Conceptually, delaying the injection of the MoRT residual from layer ℓ to $\ell + l_d$ does not reduce the representational capacity of MoRT. Formally, let F denote the backbone mapping from layer $\ell + 1$ to $\ell + l_d$. Without cross-layer delay, the hidden states at layer $\ell + l_d$ is

$$\mathbf{h}^{\ell+l_d} = F(\mathbf{h}^{\ell} + \Delta\mathbf{r}^{\ell}) \tag{17}$$

where $\Delta\mathbf{r}^{\ell}$ denotes MoRT output of layer ℓ. With cross-layer injection, it becomes

$$\mathbf{h}^{\ell+l_d}_{\text{cross}} = F(\mathbf{h}^{\ell}) + \Delta\mathbf{r}^{\ell} \tag{18}$$

A first-order Taylor expansion around h^{ℓ} gives

$$F(\mathbf{h}^{\ell} + \Delta\mathbf{r}^{\ell}) \approx F(\mathbf{h}^{\ell}) + J_F(\mathbf{h}^{\ell})\Delta\mathbf{r}^{\ell} \tag{19}$$

so that

$$\mathbf{h}^{\ell+l_d} - \mathbf{h}^{\ell+l_d}_{\text{cross}} \approx (J_F(\mathbf{h}^{\ell}) - I)\Delta\mathbf{r}^{\ell} \tag{20}$$

In pre-norm Transformers, F is itself close to the identity mapping over a short depth l_d, making J_F numerically close to I. Together with the fact that MoRT output $\Delta\mathbf{r}^{\ell}$ is small, this implies that the discrepancy between same-layer and cross-layer injection is expected to be minor.

To validate this intuition, we conduct an ablation on the 3B–250M MoE backbone with 18 layers. We apply MoRT ($n_e = 4$, topK=2) to layers 1–16 and keep layers 17–18 unchanged to match the parameter count. The router in all variants takes the post-attention layer-norm output as input. We compare: (1) **no delay**: layer-ℓ MoRT is added at layer ℓ; (2) **cross 1 layer**: layer-ℓ MoRT is added at layer $\ell + 1$; (3) **cross 2 layers**: layer-ℓ MoRT is added at layer $\ell + 2$.

All models are trained for 100B tokens. The training loss curves nearly overlap (as shown in Fig. 5), and the final loss gap between no-delay and cross-2-layer is less than 0.002, showing that cross-layer residual connections do not introduce any non-trivial degradation in model performance.

D Detailed Training and Model Hyperparameters

Table 8 provides the pretraining hyperparameters for dense models, covering four model sizes (small, medium, large, and XL). Table 9 shows the pretraining hyperparameters for MoE (Mixture of Experts) models, specifically two configurations (A110m-T1.3b and A250m-T3b).

E Load-Balancing Auxiliary Loss for MoRT

Similar to MoE architectures that require balanced expert utilization for optimal throughput and learning, MoRT benefits from uniform routing across its n_e embedding experts $\{E_i\}_{i=1}^{n_e}$. To encourage all embeddings are adequately trained and contribute to model performance, we incorporate an auxiliary load-balancing loss that encourages balanced routing distributions.

Formulation. Consider a training batch with T tokens, where MoRT maintains n_e embedding experts and each token routes to top-K embeddings. Let $G_t \subseteq \{1, \ldots, n_e\}$ denote the top-K routing set for token t, and $p_t^{(i)} \in [0, 1]$ represent the routing probability of token t to embedding i.

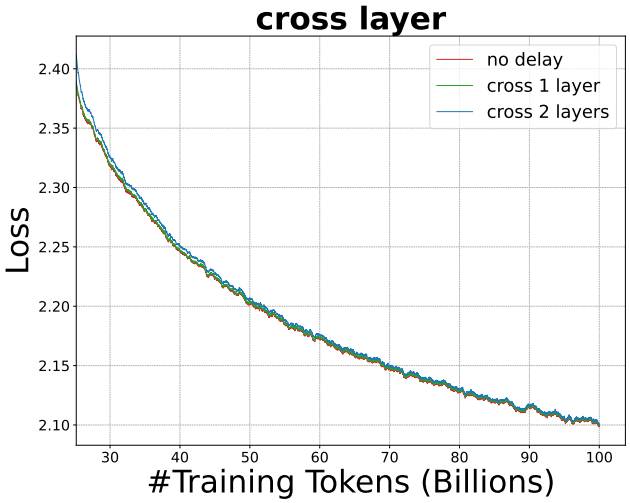


Figure 5: **Training loss with and without cross-layer residual connections.** Training curves for the 3B–250M MoE backbone (18 layers) under three variants: (1) no delay, (2) cross 1 layer, and (3) cross 2 layers. All models are trained for 100B tokens and the curves almost overlap, with the final loss gap between no-delay and cross-2-layer being less than 0.002.

Table 8: Hyper-parameters for dense model pretraining.

	Parameters	small	medium	large	XL
Training	lr-schedule	cosine	cosine	cosine	WS [38]
	max, min lr	(1e-3, 1e-4)	(8e-4, 8e-5)	(6e-4, 6e-5)	(6e-4, 0)
	warmup-ratio	0.05	0.05	0.05	0.05
	decay-ratio	0.95	0.95	0.95	0.00
	AdamW-β	(0.95, 0.9)	(0.95, 0.9)	(0.95, 0.9)	(0.95, 0.9)
	weight-decay	0.1	0.1	0.1	0.1
	grad_clip	1.0	1.0	1.0	1.0
Model	hidden dim.	768	1024	1280	1536
	#layers	12	24	36	28
	#q heads	12	16	20	12
	#kv heads	12	16	20	2
	context-length	1024	1024	1024	8192
	FFN size	3072	4096	5120	8960
Data	tokenzier	gpt2	gpt2	gpt2	qwen2
	#tokens(B)	100	100	100	330
	batch size	4096	4096	4096	512

We define the following quantities:

- **Aggregate routing probability:** $P_i = \sum_{t=1}^{T} p_t^{(i)}$ — the unnormalized sum of routing probabilities for embedding i
- **Actual token count:** $n_i = \sum_{t=1}^{T} \mathbf{1}\{i \in G_t\}$ — number of tokens actually routed to embedding i
- **Normalized routing probability:** $p_i = \frac{P_i}{T}$ — the average routing probability for embedding i
- **Normalized load fraction:** $f_i = \frac{n_i}{TK}$ — the fraction of total route capacity used by embedding i

The load-balancing auxiliary loss is formulated as:

$$\mathcal{L}_{\text{aux}} = \lambda \cdot n_e \sum_{i=1}^{n_e} p_i f_i = \lambda \cdot \frac{n_e}{T^2 K} \sum_{i=1}^{n_e} P_i n_i \tag{21}$$

Table 9: Hyper-parameters for MoE model pretraining.

	Parameters	A110m-T1.3b	A250m-T3b
Training	lr-schedule	WS	WS
	lr	4.2e-4	4.2e-4
	warmup-ratio	0.05	0.05
	decay-ratio	0.00	0.00
	AdamW-β	(0.95, 0.9)	(0.95, 0.9)
	weight-decay	0.1	0.1
	grad_clip	1.0	1.0
Model	hidden dim.	512	768
	#dense layers	1	1
	#moe layer	11	17
	#q heads	8	16
	#kv heads	4	4
	context-length	8192	8192
	#routed experts	144	144
	# shared experts	1	1
	topK route	8	8
	dense FFN size	10944	10944
	MoE FFN size	512	512
Data	tokenizer	qwen2	qwen2
	#tokens(B)	300	500
	batch size	1024	1024

where λ is a hyperparameter controlling the strength of the load-balancing constraint.

Intuition. The term p_i captures the expected routing distribution to embedding i, while f_i measures its actual utilization. The loss in Equation 21 penalizes the co-occurrence of high routing probability and high actual load, thereby discouraging the concentration of routing on a subset of embeddings. This encourages a more uniform distribution where ideally $p_i \approx f_i \approx 1/n_e$ for all i, ensuring that all embeddings receive sufficient training signal and contribute effectively to model capacity.

Implementation. In practice, we compute this auxiliary loss per layer and average across all MoRT layers. The hyperparameter λ is typically set to 10^{-4}, balancing load distribution without overwhelming the primary language modeling objective. This auxiliary loss is added to the main cross-entropy loss during training and is automatically handled by the backward pass without requiring special gradient computation.

F Further Analysis on ReToken's Efficacy

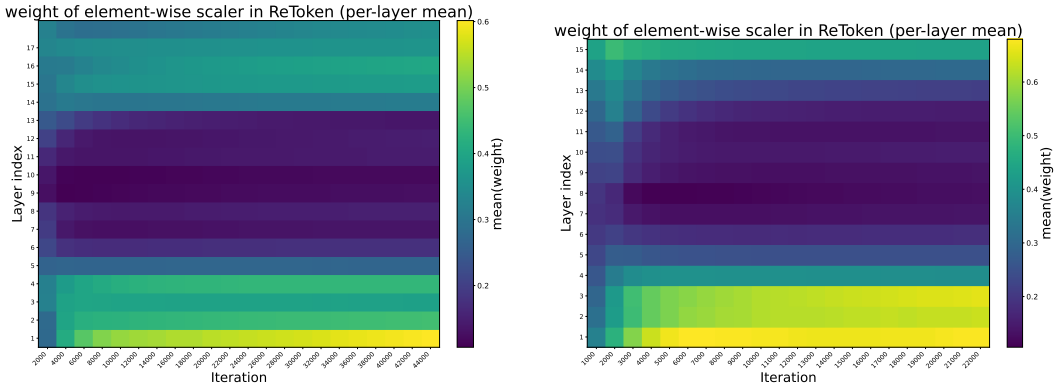
Greater Gains on Dense-XL vs. Smaller Dense Models. ReToken's larger performance uplift on the Dense-XL model is directly attributable to its vocabulary size. The Qwen2 tokenizer used by Dense-XL has a vocabulary of 152k, three times larger than the 50k GPT-2 vocabulary of the smaller models. This triples the number of token-indexed parameters, providing a much richer capacity to learn token-specific priors.

Performance on MoE vs. Dense Architectures. The gains from ReToken are more modest on MoE backbones because they constitute a much stronger, more parameter-efficient baseline than dense models. This observation aligns with prior work showing that highly efficient architectures can exhibit diminishing returns from certain augmentations [16]. Notably, our **MoRT** architecture overcomes this by further scaling token-indexed parameters, proving effective even on strong MoE backbones.

G Characterizing the Learned Sparse Embedding Parameters

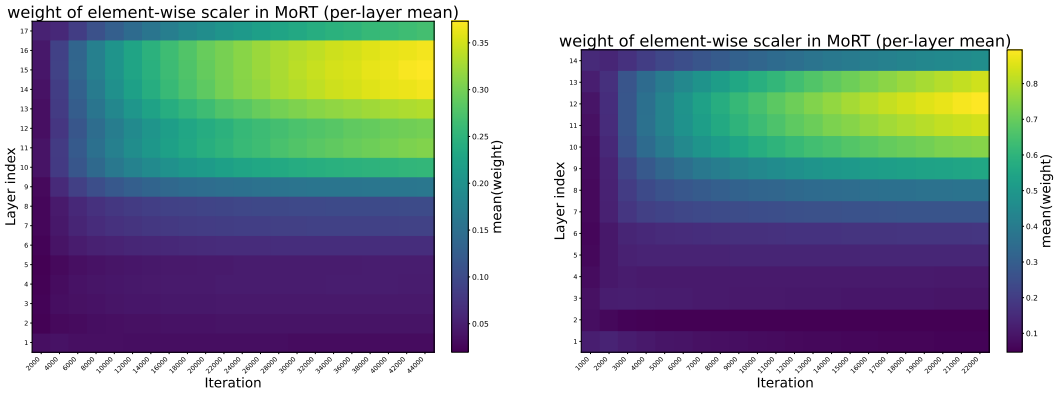
To better understand the behaviour of the sparse embedding parameters after training, we analyze the element-wise scalers that modulate the contribution of ReToken/MoRT tables at each layer.

Recall that in both ReToken and MoRT, the token-indexed vector retrieved from the embedding tables is multiplied by a learned element-wise scaler before being injected into the residual stream.



(a) 18-layer model (b) 15-layer model

Figure 6: Evolution of the per-layer mean element-wise scalers in ReToken for the 3B–A250m (18 layers) and 9B–A0.8B (15 layers) backbones. The x-axis denotes training iterations and the y-axis denotes layer index.



(a) 18-layer model (b) 15-layer model

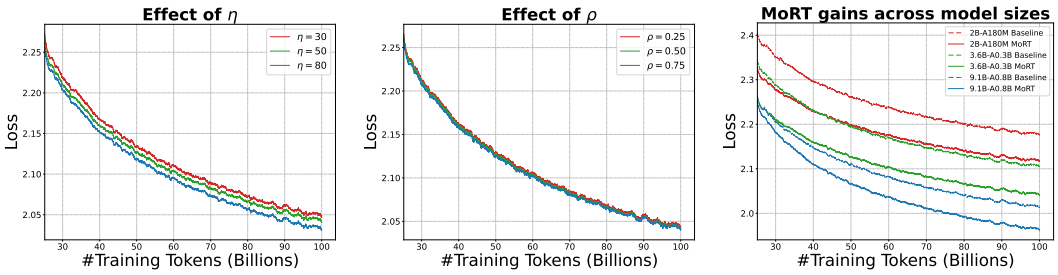
Figure 7: Evolution of the per-layer mean element-wise scalers in MoRT for the 3B–A250m (18 layers) and 9B–A0.8B (15 layers) backbones. The x-axis denotes training iterations and the y-axis denotes layer index.

Intuitively, the magnitude of this scaler controls how much the sparse embedding parameters can influence the hidden states. If the scaler stays close to zero, the backbone behaves almost like the vanilla Transformer; if it grows larger, the token-indexed parameters play a more prominent role.

For each checkpoint and each layer, we therefore compute the mean value of the element-wise scaler along the feature dimension and visualize it as a heatmap over training steps (horizontal axis) and layer index (vertical axis). We report results for two backbones: an 18-layer deep–thin 3B–A250m MoE model and a 15-layer shallow–wide 9B–A0.8B MoE model. Figure 6 shows the evolution of the ReToken scalers, and Figure 7 shows the corresponding MoRT scalers.

Several consistent patterns emerge:

- **Magnitude and stability.** Across all layers and checkpoints, the mean scalers remain well below 1.0 and neither explode nor vanish. This indicates that the sparse embedding parameters act as small residual corrections on top of the backbone features rather than dominating the representation.
- **Depth-wise selectivity.** The model does not use all layers equally. For ReToken, we observe stronger scalers in the lower and upper layers, with a valley of weaker modulation in the middle layers. For MoRT, the scalers gradually increase with depth so that the upper



(a) Effect of parameter ratio η on MoRT.

(b) Effect of activation sparsity ρ on MoRT.

(c) Consistent MoRT gains across model sizes.

Figure 8: MoRT training dynamics under different (η, ρ) configurations and scale-invariant gains across model sizes.

layers receive the strongest modulation, while early layers are barely affected. This suggests that the network automatically learns where in depth the token-indexed parameters are most useful and keeps other layers close to the baseline MoE.

- **Cross-architecture consistency.** The deep–thin (3B–A250m) and shallow–wide (9B–A0.8B) backbones exhibit very similar depth-wise patterns. This robustness across backbone shapes suggests that the qualitative characteristics of the learned sparse embedding parameters are not specific to a particular model size or aspect ratio.

Overall, these results show that the learned sparse embedding parameters are structured, depth-selective and stable, and that the backbone learns to use them as targeted residual refinements.

H MoRT TRAINING DYNAMICS ACROSS DIFFERENT η, ρ AND MODEL SIZE

To further investigate MoRT's training dynamics and performance gains under different (η, ρ) settings, we take the A0.3B-T3.6B MoE model as the backbone and train MoRT variants with multiple (η, ρ) configurations for 100B tokens. To deeply examine how the improvements from MoRT transfer across different model sizes, we train both baseline MoE and MoRT models with three backbone sizes (A180m-T2B, A0.3B-T3.6B, and A0.8B-T9.1B) under a fixed MoRT configuration $(\eta, \rho) = (50, 0.5)$ for 100B tokens.

Training Dynamics Under Different (η, ρ). Figure 8a and figure 8b present the training dynamics on the A0.3B-T3.6B MoE backbone. As shown in Fig. 8a, increasing the parameter ratio η from 30 to 80 yields consistent and approximately parallel downward shifts in the loss curve, with the final loss decreasing by 0.02-0.03.

In contrast, activation sparsity ρ has a much more limited impact (Fig. 8b). Varying ρ from 0.25 to 0.75 results in a total loss reduction of less than 0.01, which is significantly smaller than the effect of η. Given that larger ρ values increase the volume of modulation vectors that need to be accessed ($K = \rho \cdot n_e$ vectors per token), thereby impacting inference latency, a moderate value serves as an optimal trade-off between marginal performance gains and communication overhead.

Consistent MoRT gains across model sizes. Fig. 8c shows that MoRT's benefits are remarkably consistent across model scales. With a fixed configuration (η, ρ)=(50, 0.5), MoRT reduces the final loss by 0.059 on the 2B–A180M backbone, by 0.064 on the 3.6B–A0.3B backbone, and by 0.051 on the 9.1B–A0.8B backbone, corresponding to relative improvements of 2.7%, 3.0%, and 2.55%, respectively. These consistently positive and tightly clustered gains ($\approx 2.75\% \pm 0.25\%$) across a 4.5× range in model size provide strong empirical support for our theoretical framework: the benefits of token-indexed parameters don't diminish at larger models; thus, they constitute a genuinely orthogonal scaling dimension.

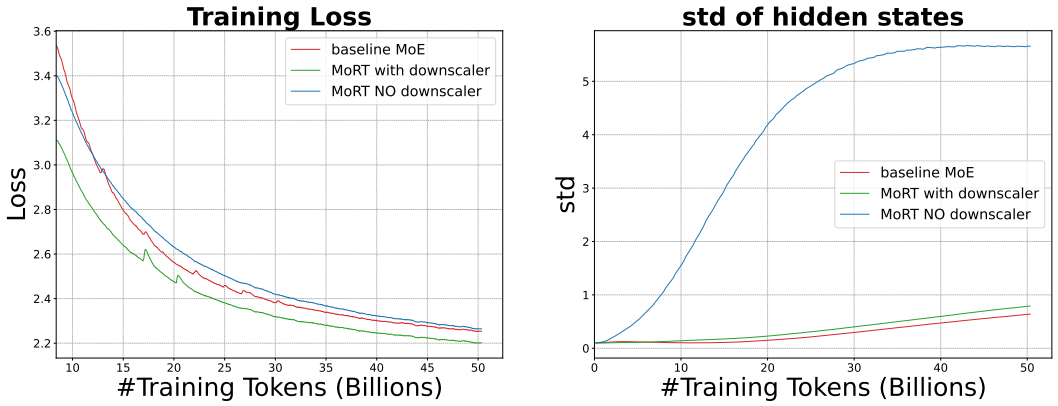


Figure 9: Ablation of the scaling factor $\frac{1}{\sqrt{2N_\ell}}$ in MoRT on a 3B–A250m MoE backbone. Left: training loss. Right: averaged std of hidden states over layers and tokens.

I Ablation on scaling factor $\frac{1}{\sqrt{2N_l}}$ in MoRT

To stabilize the variance of hidden states when injecting MoRT residuals, we scale the modulation update in each layer with a factor $\frac{1}{\sqrt{2N_l}}$ (Eq. 5).

We perform an ablation on a 3B–A250m MoE backbone comparing (i) the baseline MoE, (ii) MoRT *with* the scaling factor (our default), and (iii) MoRT *without* the scaling factor. All models are trained under identical data and optimization settings.

Figure 9 (left) reports the training loss, while Figure 9 (right) shows the standard deviation of hidden states averaged over all layers and tokens. Without the scaling factor, the activation std grows rapidly to above 5.5 within the first 50B training tokens and the model slightly underperforms the baseline MoE in terms of loss. In contrast, MoRT with the proposed $\frac{1}{\sqrt{2N_\ell}}$ scaling keeps close to baseline in a well-behaved range and consistently achieves lower training loss than the baseline. These results confirm that the scaling factor is crucial to prevent variance explosion and to make MoRT's modulation effectively improve optimization.

J LLM Usage Statement

We used the advanced language models only as writing assistants during manuscript preparation. Its role was limited to correcting grammar, improving clarity, and refining the flow of sentences, while keeping the intended meaning unchanged. All research ideas, methods, and results are entirely the work of the authors.