CoMoL: Efficient Mixture of LoRA Experts via Dynamic Core Space Merging

Jie Cao[1*], Zhenxuan Fan[1*], Zhuonan Wang[1], Tianwei Lin[1], Ziyuan Zhao[2], Rolan Yan[2], Wenqiao Zhang[1†], Feifei Shao[1†], Hongwei Wang[1], Jun Xiao[1], Siliang Tang[1]

[1]Zhejiang University [2]Wechat, Tencent

{caojie, zxfan, wnzz, lintw, wenqiaozhang, sff, siliang}@zju.edu.cn

{joshuazhao, rolanyan}@tencent.com , hongweiwang@intl.zju.edu.cn, junx@cs.zju.edu.cn

Abstract

Large language models (LLMs) achieve remarkable performance on diverse downstream and domain-specific tasks via parameter-efficient fine-tuning (PEFT). However, existing PEFT methods, particularly MoE-LoRA architectures, suffer from limited parameter efficiency and coarse-grained adaptation due to the proliferation of LoRA experts and instance-level routing. To address these issues, we propose Core Space Mixture of LoRA (**CoMoL**), a novel MoE-LoRA framework that incorporates expert diversity, parameter efficiency, and fine-grained adaptation. Specifically, CoMoL introduces two key components: Core Space Experts and Core Space Routing. Core Space Experts store each expert in a compact core matrix, preserving diversity while controlling parameter growth. Core Space Routing dynamically selects and activates the appropriate core experts for each token, enabling fine-grained, input-adaptive routing. Activated core experts are then merged via a soft-merging strategy into a single core expert, which is combined with a shared LoRA to form a specialized LoRA module. Besides, the routing network is projected into the same low-rank space as the LoRA matrices, further reducing parameter overhead without compromising expressiveness. Extensive experiments demonstrate that CoMoL preserves the adaptability of MoE-LoRA architectures while achieving parameter efficiency comparable to Standard LoRA, consistently outperforming existing methods across multiple tasks. Our code is available at https://github.com/DCDmllm/CoMoL.

1 Introduction

The rapid advancement of large language models (LLMs) (Achiam et al., 2023; Yang et al., 2025; Grattafiori et al., 2024) has greatly enhanced natural language processing across a variety of tasks. However, their massive size makes it challenging to adapt them efficiently to downstream or domain-specific applications. To address this, parameter-efficient fine-tuning (PEFT) methods (Houlsby et al., 2019; He et al., 2022; Li and Liang, 2021; Lester et al., 2021; Hu et al., 2021) have emerged, enabling adaptation by updating a small fraction of model parameters, which substantially reduces training costs and alleviates catastrophic forgetting. Among them, Low-Rank Adaptation (LoRA) (Hu et al., 2021) achieves competitive performance through low-rank decomposition of weight matrices with minimal trainable parameters. Unfortunately, recent studies reveal that naively increasing the LoRA rank does not lead to proportional performance gains (Chen et al., 2022; Zhu et al., 2023).

Inspired by Mixture-of-Experts (MoE) (Shazeer et al., 2017), recent PEFT studies integrate LoRA into the MoE architecture to enhance model capacity, giving rise to the MoE-LoRA paradigm. Specifically, MoE-LoRA consists of a routing network and multiple LoRA experts, where the router dynamically selects and activates experts conditioned on the input condition (Zadouri et al., 2023; Zhu et al., 2023). This design is motivated by two key considerations: **(1)** multiple LoRA experts can capture complementary representations, providing greater expressiveness than a single LoRA module; and **(2)** the routing network facilitates flexible, input-dependent expert selection, enabling more efficient representation learning.

Despite these advancements, existing MoE-LoRA methods still face two major limitations. **(1) Limited Parameter Efficiency:** The MoE-LoRA architecture adds a routing network and multiple LoRA modules, most of which participate in subsequent computations, leading to increased parameter counts and higher computational overhead; **(2) Coarse-grained Adaptation:** SMEAR (Muqeeth et al., 2024) constructs a single merged expert by computing its parameters as a weighted average of

[*]Equal contribution

[†]Corresponding author

Findings of the Association for Computational Linguistics: ACL 2026, pages 16471–16481

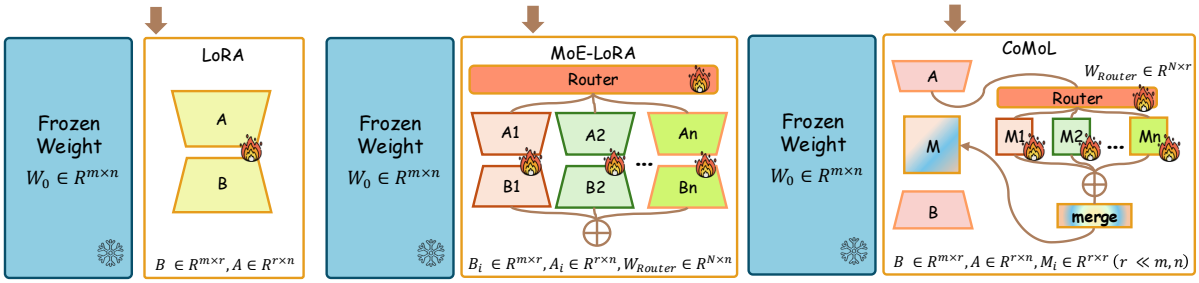

Figure 1: **Comparison of LoRA-based methods.** **(a)** LoRA injects trainable low-rank matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into transformer layers to approximate weight updates. **(b)** MoE-LoRA employs multiple LoRA experts with a routing network to select and activate experts based on input tokens. **(c)** CoMoL fuses expert parameters in the low-rank core space, preserving token-level routing while maintaining expert-specific expressiveness with minimal parameters.

multiple experts within a routing block. However, its routing mechanism operates at the instance level, relying on instance features rather than token-level features, which limits fine-grained adaptability.

To address the challenges of MoE-LoRA architectures, we propose Core Space Mixture of LoRA (**CoMoL**), a novel MoE-LoRA framework that incorporates expert diversity, parameter efficiency, and fine-grained adaptation. As illustrated in Figure 1(c), CoMoL introduces two key components: Core Space Experts and Core Space Routing. Core Space Experts store each expert in a compact core matrix within a shared core space, whose dimensionality matches the LoRA rank, preserving diverse expert knowledge while controlling parameter overhead. Core Space Routing dynamically selects and activates the appropriate core experts for each input token, enabling fine-grained, token-level adaptation. Activated experts are then merged through a soft-merging strategy into a single core expert, which is combined with a LoRA module to form a specialized LoRA.

Compared to previous MoE-LoRA methods, CoMoL preserves expert knowledge with only a minimal set of parameters in the core space. Importantly, by performing token-level routing and soft-merging entirely within the core space, CoMoL enables fine-grained adaptation with minimal overhead. To further reduce the parameter cost of the routing module, CoMoL leverages the inherent low-rank structure of LoRA, projecting the router into the same low-rank space. This design simultaneously maintains strong performance and reduces the overall parameter count to a level comparable to Standard LoRA. The contributions of this paper are summarized as follows:

- We highlight that existing MoE-LoRA methods primarily suffer from limited parameter efficiency and coarse-grained adaptation, stemming from the proliferation of LoRA experts and instance-level routing over experts.
- We introduce Core Space Mixture of LoRA (**CoMoL**), an enhanced MoE-LoRA framework that stores expert parameters within a compact core space, enabling token-level routing while achieving parameter efficiency comparable to Standard LoRA.
- Extensive experiments against state-of-the-art MoE-LoRA methods demonstrate that CoMoL achieves superior scalability and robustness across a wide range of tasks and settings.

2 Revisiting MoE-LoRA Methods

2.1 LoRA Basics

LoRA (Hu et al., 2021) injects trainable low-rank matrices into transformer layers to approximate the weight updates. For a frozen pre-trained weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, LoRA represents its update with a low-rank decomposition:

$$\boldsymbol{W} + \Delta\boldsymbol{W} = \boldsymbol{W} + \boldsymbol{B}\boldsymbol{A}, \tag{1}$$

where $\boldsymbol{B} \in \mathbb{R}^{m\times r}$, $\boldsymbol{A} \in \mathbb{R}^{r\times n}$ are tunable parameters, and the rank $r \ll min(m, n)$. For an input token $\boldsymbol{x}$ to the linear projection in the transformer layer, the computation of LoRA is as follows:

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{A}\boldsymbol{x}. \tag{2}$$

2.2 MoE-LoRA Methods

The MoE-LoRA structure consists of N different LoRA experts $\{\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_N\}$, where each

Methods	Params (×)	FLOPs (×)	Routing Level	Routing Latency
LoRA	1.0×	1.0×	-	-
Soft-weighted MoE-LoRA	$N\times$	$N\times$	Token	Low
Sparse MoE-LoRA	$N\times$	$k\times$	Token	High
Soft-merging MoE-LoRA	$N\times$	1.0×	Instance	Low
CoMoL	1.0×	1.0×	Token	Low

Table 1: Comparison of LoRA and MoE-LoRA variants. Metrics for "Params" and "FLOPs" reflect only the LoRA experts; routing overhead is omitted for simplicity. $N\times$ indicates the ratio relative to the LoRA baseline.

LoRA expert $\boldsymbol{E}_i$ is represented by a pair of matrices $\{\boldsymbol{B}_i \in \mathbb{R}^{m\times r}, \boldsymbol{A}_i \in \mathbb{R}^{r\times n}\}$. The MoE-LoRA structure incorporates a gating network (Router) $\boldsymbol{G}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^N$ that computes the weight $\boldsymbol{G}(\boldsymbol{x})_i$ for each expert $\boldsymbol{E}_i$ given an input token $\boldsymbol{x}$. The forward process of MoE-LoRA is computed as:

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \sum_{i\in\mathcal{T}(\boldsymbol{x})} \boldsymbol{G}(\boldsymbol{x})_i \cdot \boldsymbol{B}_i\boldsymbol{A}_i\boldsymbol{x}, \quad (3)$$

where the output $\boldsymbol{h}$ combines the frozen linear projection output and the router-weighted combination of activated expert outputs, $\mathcal{T}(\boldsymbol{x})$ denotes the set of indices for activated experts determined by the routing mechanism.

MoE-LoRA employs two types of routing mechanisms: *soft routing mechanism* and *sparse routing mechanism*, which correspond to *soft-weighted MoE-LoRA* and *sparse MoE-LoRA*, respectively.

Soft routing mechanism. Given an input token $\boldsymbol{x}$, the soft routing mechanism typically employs a dense linear layer followed by a softmax function:

$$\boldsymbol{G}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{W}_g\boldsymbol{x}), \quad (4)$$

where $\boldsymbol{W}_g \in \mathbb{R}^{N\times n}$ is the trainable projection. Soft-weighted MoE-LoRA with soft routing mechanism activates all LoRA experts and combines their outputs weighted by the routing scores $\boldsymbol{G}(\boldsymbol{x})$.

Sparse routing mechanism. The sparse MoE-LoRA selectively activates a subset of experts, typically the top-k LoRA experts based on routing scores $\boldsymbol{G}(\boldsymbol{x})$, and aggregates only the outputs from the activated experts:

$$\boldsymbol{G}(\boldsymbol{x}) = \text{top-k}(\text{softmax}(\boldsymbol{W}_g\boldsymbol{x})). \quad (5)$$

2.3 Soft-merging MoE-LoRA Method

To reduce the computational costs of LoRA experts, SMEAR (Muqeeth et al., 2024) proposes an instance-level LoRA expert fusion method. SMEAR first makes a single routing choice for an entire input example:

$$\boldsymbol{G}(\hat{\boldsymbol{x}})^{ins} = \text{softmax}(\boldsymbol{W}_g\hat{\boldsymbol{x}}), \quad (6)$$

where $\hat{\boldsymbol{x}}$ denotes the average representation of the input example. Then, based on the routing scores, the LoRA experts are merged into a single expert:

$$\boldsymbol{B}_{ins} = \sum_{i=1}^{N} \boldsymbol{B}_i\boldsymbol{G}(\hat{\boldsymbol{x}})_i^{ins}, \boldsymbol{A}_{ins} = \sum_{i=1}^{N} \boldsymbol{A}_i\boldsymbol{G}(\hat{\boldsymbol{x}})_i^{ins}, \quad (7)$$

Finally, in the forward process of SMEAR, the merged expert is applied to compute each token in the sample:

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{B}_{ins}\boldsymbol{A}_{ins}\boldsymbol{x}, \quad (8)$$

It is important to note that while in other MoE-LoRA methods each token in a sample is processed by different expert combinations, in SMEAR, all tokens in the sample use the same merged expert.

2.4 Comparison of MoE-LoRA Methods

Table 1 provides a comprehensive comparison between soft-weighted, sparse, and soft-merging MoE-LoRA architectures, evaluated across parameters, FLOPs, routing granularity, and routing latency. All three variants maintain an $N\times$ increase in LoRA expert parameters, which fundamentally diverges from the core principles of Parameter-Efficient Fine-Tuning (PEFT). Furthermore, previous research (Tian et al., 2024; Lin et al., 2024; Cao et al., 2025) indicates that these multiple experts often suffer from significant parameter redundancy.

While soft-weighted MoE-LoRA activates all experts, resulting in $N\times$ FLOPs, sparse MoE-LoRA attempts to mitigate this by activating only a subset (typically top-2). However, the empirical analysis (Cao et al., 2025) reveals an efficiency paradox: despite its lower theoretical FLOPs, sparse MoE-LoRA incurs significantly higher wall-clock

latency than its soft-weighted counterpart. Under identical experimental settings (e.g., $N = 8$), sparse routing increases training time by approximately $2\times$ and inference time by $1.5\times$ (Cao et al., 2025), primarily due to the substantial computational overhead of the sparse routing mechanism. Conversely, while soft-merging MoE-LoRA reduces the burden to a single merged expert computation, it sacrifices token-level dynamism by degrading to an instance-level approach, thus limiting the model's fine-grained expressive capacity.

Overall, an ideal MoE-LoRA variant should achieve a favorable trade-off between expressivity and efficiency. Specifically, it is imperative to: **(i)** Minimize the total parameter footprint while preserving the specialized characteristics of individual LoRA experts; **(ii)** Reduce the computational overhead associated with the activated experts to enhance overall efficiency; **(iii)** Sustain a fine-grained token-level routing mechanism while ensuring the routing latency remains negligible.

3 Technical Details of CoMoL

3.1 Core Space of LoRA

Let $\{\boldsymbol{B} \in \mathbb{R}^{m\times r}, \boldsymbol{A} \in \mathbb{R}^{r\times n}\}$ denote a LoRA expert within the MoE-LoRA framework. Each expert-specific weight update, $\Delta\boldsymbol{W} = \boldsymbol{BA}$, can be re-parameterized via the reduced SVD of its constituent matrices:

$$\begin{aligned} &\boldsymbol{B} = \boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^\top, \boldsymbol{A} = \boldsymbol{U}_A\boldsymbol{\Sigma}_A\boldsymbol{V}_A^\top, \\ &\Delta\boldsymbol{W} = \boldsymbol{U}_B\boldsymbol{\Sigma}_B\boldsymbol{V}_B^\top\boldsymbol{U}_A\boldsymbol{\Sigma}_A\boldsymbol{V}_A^\top, \end{aligned} \tag{9}$$

where the dimensions are partitioned as $\boldsymbol{U}_B \in \mathbb{R}^{m\times r}$, $\boldsymbol{\Sigma}_B \in \mathbb{R}^{r\times r}$, $\boldsymbol{V}_B^\top \in \mathbb{R}^{r\times r}$, $\boldsymbol{U}_A \in \mathbb{R}^{r\times r}$, $\boldsymbol{\Sigma}_A \in \mathbb{R}^{r\times r}$, and $\boldsymbol{V}_A^\top \in \mathbb{R}^{r\times n}$.

Following the definition of the **Core Matrix** $\boldsymbol{M} \in \mathbb{R}^{r\times r}$ (Panariello et al.):

$$\boldsymbol{M} = \boldsymbol{\Sigma}_B\boldsymbol{V}_B^\top\boldsymbol{U}_A\boldsymbol{\Sigma}_A, \tag{10}$$

the expert weight update can be compactly decoupled as:

$$\Delta\boldsymbol{W} = \boldsymbol{U}_B\boldsymbol{M}\boldsymbol{V}_A^\top. \tag{11}$$

In this formulation, the matrices $\boldsymbol{U}_B$ and $\boldsymbol{V}_A^\top$ span the singular subspaces that determine the feature orientations, while the core matrix $\boldsymbol{M}$ modulates the coupling strength and weight allocation across these latent dimensions.

3.2 Core Space Mixture of LoRA

To address the parameter redundancy inherent in MoE-LoRA experts, we hypothesize that individual experts share a substantially similar latent subspace defined by the singular bases $\boldsymbol{U}_B$ and $\boldsymbol{V}_A^\top$. By confining expert-specific adaptations exclusively to the compact core matrix $\boldsymbol{M}_i$, we achieve a significant reduction in the parameter footprint. Under this formulation, the forward process for a given input $\boldsymbol{x}$ is defined as:

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \sum_{i=1}^{N} \boldsymbol{G}(\boldsymbol{x})_i \cdot (\boldsymbol{U}_B\boldsymbol{M}_i\boldsymbol{V}_A^\top)\boldsymbol{x}, \tag{12}$$

where $G(\boldsymbol{x})_i$ is a scalar that denotes the token-level routing weight for the i-th expert.

Parameter Efficiency Unlike standard MoE-LoRA, which replicates the full $\{\boldsymbol{B}_i, \boldsymbol{A}_i\}$ pairs N times, our approach constrains the per-expert parameters to the core matrix $\boldsymbol{M}_i$. This reduces the expert-related parameter complexity from $\mathcal{O}(N \cdot (m+n)r)$ to $\mathcal{O}((m+n)r+N\cdot r^2)$. Since $r \ll m, n$, our method maintains a near-constant parameter footprint as the number of experts N scales, effectively satisfying the core principles of PEFT.

Token-level soft-merging of experts in Core Space Another key advantage of our formulation is the ability to perform core space expert merging at the token level without significant computational overhead. By exploiting **the distributive property of matrix multiplication**, the forward process described by Equation 12 can be equivalently expressed as

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{U}_B\left(\sum_{i=1}^{N} \boldsymbol{G}(\boldsymbol{x})_i\boldsymbol{M}_i\right)\boldsymbol{V}_A^\top\boldsymbol{x}. \tag{13}$$

This allows the model to first aggregate the small $r \times r$ core matrices into a single merged matrix $\boldsymbol{M}_{merged}$ for each token. Consequently, the high-dimensional projections $\boldsymbol{U}_B$ and $\boldsymbol{V}_A$ are only computed once per token, reducing the total FLOPs to a level comparable to a single LoRA layer while preserving the fine-grained token-level dynamism of the MoE mechanism.

To provide a rigorous computational complexity analysis, we compare the two expert aggregation strategies. Consider a batch of L tokens with a model dimension n, a LoRA rank r, and N total experts. Under the output-level fusion strategy (as defined in Eq.12), the FLOPs required for expert

computation scale as $2\times L\times r\times(2n+r)\times N$, while the subsequent aggregation of expert outputs incurs $L\times n\times(N-1)$ FLOPs. In contrast, our proposed soft-merging in core space (Eq. 13) streamlines this process. The FLOPs for expert computation are reduced to $2\times L\times r\times(2n+r)$, and the fusion of core matrices requires only $L\times r^2\times(N-1)$ FLOPs. Notably, our Core Space Merging reduces the primary expert computational overhead to a factor of $1/N$ compared to the output-level baseline. Furthermore, given that $r^2 \ll n$, the cost of aggregating core matrices is significantly lower than that of fusing high-dimensional expert outputs. This demonstrates that our method achieves superior efficiency in both primary transformation and expert orchestration.

3.3 Core Space Routing

In conventional MoE-LoRA architectures, the parameter count of the router $\boldsymbol{W}_g \in \mathbb{R}^{N\times n}$ scales linearly with the number of experts N. This dependency introduces a significant computational bottleneck and a substantial memory footprint, thereby undermining the inherent efficiency of the MoE-LoRA framework. To circumvent this limitation, we propose a Core Space Routing mechanism, which significantly reduces the parameter overhead while effectively leveraging the low-rank properties of LoRA. Specifically, we reuse the intermediate result of the input $\boldsymbol{x}$ in the low-rank projection space, denoted as $\hat{\boldsymbol{x}} = \boldsymbol{V}_A^\top \boldsymbol{x} \in \mathbb{R}^r$, as the input for the router. The layer output $\boldsymbol{h}$ is then computed as follows:

$$\hat{\boldsymbol{x}} = \boldsymbol{V}_A^\top \boldsymbol{x} \tag{14}$$

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{U}_B \left(\sum_{i=1}^{N} \boldsymbol{G}(\hat{\boldsymbol{x}})_i \boldsymbol{M}_i \right) \hat{\boldsymbol{x}}. \tag{15}$$

By performing routing within this latent core space, the parameter complexity of the router weight $\boldsymbol{W}_g$ is reduced from $\mathcal{O}(N\cdot n)$ to $\mathcal{O}(N\cdot r)$. Given that the rank r is typically orders of magnitude smaller than the model dimension n (e.g., $r=8$ vs. $n=4096$), this mechanism drastically curtails the storage requirements and computational cost associated with the router without sacrificing the fine-grained dynamism of token-level routing.

4 Experiments

In this section, we present a comprehensive empirical evaluation to validate the effectiveness of CoMoL across a diverse range of benchmarks. We first assess its mathematical reasoning capabilities using the Qwen3-8B and its larger counterpart, Qwen3-14B, to examine performance across different parameter scales. Subsequently, to verify the cross-architecture robustness of our method, we evaluate CoMoL on coding tasks using representative models from distinct families, specifically Qwen3-8B and Llama3.1-8B.

To facilitate a rigorous and comprehensive comparative analysis, we benchmark our proposed method against Standard LoRA and several state-of-the-art MoE-LoRA variants across different paradigms: **(i)** the vanilla Standard LoRA (Hu et al., 2021); **(ii)** soft-weighted MoE-LoRA methods, including MoLoRA (Zadouri et al., 2023) and HydraLoRA (Tian et al., 2024); **(iii)** sparse MoE-LoRA frameworks, represented by MoLA (Gao et al., 2024), AdaMoLE (Liu and Luo, 2024) and SparseMoA (Cao et al., 2025); and **(iv)** other advanced LoRA-based methods such as DenseLoRA (Mu et al., 2025) and FlyLoRA (Zou et al.).

4.1 Mathematical reasoning

Datasets. To evaluate the performance on mathematical reasoning tasks, we utilize Math14k (Hu et al., 2023) as the fine-tuning corpus, which comprises the training subsets of GSM8K and AQuA, augmented with Chain-of-Thought (CoT) reasoning paths to enhance logical transparency. During the evaluation phase, we systematically benchmark our model across six diverse datasets: GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MultiArith (Roy and Roth, 2016), AddSub (Hosseini et al., 2014), AQuA (Ling et al., 2017), and SingleEq (Koncel-Kedziorski et al., 2015).

Experimental setting. We employ Qwen3-8B and the larger-scale Qwen3-14B (Yang et al., 2025) as our backbone models. To ensure a fair and rigorous comparison, all experiments, including the proposed method and baselines, are implemented within a unified framework based on the Transformers library (Wolf et al., 2020), maintaining identical hyperparameter configurations wherever applicable. Following the setup in SparseMoA (Cao et al., 2025), we apply LoRA or MoE-LoRA adaptations to the Query (Q), Key (K), Value (V), Output (O), and Down-projection modules, while excluding the Gate and Up-projection layers to optimize the memory footprint of MoE-LoRA methods. For all MoE-LoRA methods, we set the LoRA rank to 8. The number of experts is set to 8 for Qwen3-8B,

Models	AddSub	AQuA	GSM8k	MultiArith	SingleEq	SVAMP	Average	Param
LoRA	$93.67_{\pm 0.39}$	$37.40_{\pm 2.58}$	$81.80_{\pm 0.67}$	$98.17_{\pm 0.86}$	$96.85_{\pm 0.23}$	$88.80_{\pm 0.53}$	$82.78_{\pm 0.47}$	24.77M
MoLoRA	$93.76_{\pm 1.30}$	$42.26_{\pm 2.41}$	$82.51_{\pm 2.35}$	$98.11_{\pm 0.38}$	$96.06_{\pm 1.04}$	$90.40_{\pm 0.30}$	$83.85_{\pm 1.17}$	107.35M
HydraLoRA	$93.16_{\pm 1.27}$	$39.11_{\pm 2.79}$	$81.78_{\pm 1.48}$	$98.11_{\pm 0.25}$	$95.21_{\pm 0.75}$	$88.80_{\pm 0.26}$	$82.70_{\pm 0.88}$	49.55M
MoLA	$93.33_{\pm 0.89}$	$42.78_{\pm 0.45}$	$81.50_{\pm 1.14}$	$98.33_{\pm 0.44}$	$96.13_{\pm 0.50}$	$89.83_{\pm 1.16}$	$83.65_{\pm 0.26}$	107.35M
AdaMoLE	$92.24_{\pm 1.05}$	$41.73_{\pm 4.26}$	$81.65_{\pm 1.00}$	$98.22_{\pm 0.25}$	$96.19_{\pm 0.80}$	$89.10_{\pm 0.75}$	$83.19_{\pm 0.85}$	108.38M
SparseMoA	$93.33_{\pm 0.15}$	$40.68_{\pm 2.95}$	$83.85_{\pm 0.35}$	$98.17_{\pm 0.44}$	$97.31_{\pm 0.11}$	$88.80_{\pm 1.66}$	$83.69_{\pm 0.76}$	24.49M
FlyLoRA	$93.00_{\pm 0.81}$	$39.11_{\pm 1.98}$	$86.20_{\pm 0.50}$	$98.44_{\pm 0.10}$	$95.87_{\pm 0.86}$	$89.23_{\pm 0.70}$	$83.64_{\pm 0.40}$	24.77M
CoMoL w/o CR	$93.92_{\pm 2.42}$	$44.09_{\pm 1.80}$	$83.67_{\pm 2.51}$	$98.28_{\pm 0.19}$	$97.31_{\pm 0.69}$	$89.53_{\pm 0.80}$	$\underline{84.47}_{\pm 1.25}$	33.39M
CoMoL	$94.60_{\pm 1.05}$	$44.62_{\pm 5.32}$	$83.93_{\pm 1.64}$	$98.22_{\pm 0.69}$	$96.46_{\pm 0.86}$	$89.03_{\pm 0.55}$	$\mathbf{84.48}_{\pm 1.40}$	25.16M

Table 2: Experimental results on Qwen3-8B for mathematical reasoning benchmarks. Accuracy is used as the evaluation metric. All methods are run with three random seeds; the mean and standard deviation are reported. "Param" indicates trainable parameters. "CoMoL w/o CR" denotes that Core Space Routing is not applied in CoMoL.

Models	AddSub	AQuA	GSM8k	MultiArith	SingleEq	SVAMP	Average	Param
LoRA	$96.20_{\pm 0.25}$	$41.34_{\pm 3.39}$	$87.49_{\pm 0.86}$	$97.83_{\pm 0.29}$	$97.64_{\pm 0.41}$	$90.60_{\pm 0.56}$	$85.18_{\pm 0.86}$	35.39M
MoLoRA	$96.37_{\pm 0.29}$	$44.88_{\pm 2.73}$	$87.59_{\pm 1.12}$	$98.33_{\pm 0.29}$	$97.90_{\pm 0.45}$	$91.50_{\pm 0.69}$	$86.10_{\pm 0.72}$	76.84M
HydraLoRA	$95.78_{\pm 1.39}$	$43.70_{\pm 2.39}$	$88.05_{\pm 1.20}$	$98.17_{\pm 1.04}$	$97.77_{\pm 0.41}$	$91.37_{\pm 0.81}$	$85.81_{\pm 0.34}$	40.47M
MoLA	$96.96_{\pm 0.67}$	$42.26_{\pm 4.55}$	$88.98_{\pm 0.68}$	$98.67_{\pm 0.17}$	$97.83_{\pm 0.52}$	$91.63_{\pm 0.57}$	$86.06_{\pm 0.66}$	153.68M
AdaMoLE	$96.20_{\pm 0.25}$	$40.55_{\pm 4.54}$	$88.02_{\pm 1.26}$	$98.50_{\pm 0.17}$	$98.10_{\pm 0.23}$	$91.13_{\pm 0.70}$	$85.42_{\pm 0.54}$	78.36M
SparseMoA	$96.29_{\pm 0.89}$	$44.62_{\pm 3.44}$	$88.20_{\pm 1.02}$	$98.11_{\pm 0.35}$	$98.10_{\pm 0.11}$	$91.63_{\pm 0.99}$	$\underline{86.16}_{\pm 0.77}$	34.50M
FlyLoRA	$95.27_{\pm 0.15}$	$41.86_{\pm 2.24}$	$88.53_{\pm 0.52}$	$98.11_{\pm 0.35}$	$96.72_{\pm 0.50}$	$91.47_{\pm 0.15}$	$85.33_{\pm 0.40}$	35.39M
CoMoL w/o CR	$96.03_{\pm 0.96}$	$45.28_{\pm 4.47}$	$87.72_{\pm 2.46}$	$98.11_{\pm 0.67}$	$97.83_{\pm 0.20}$	$91.50_{\pm 0.96}$	$86.08_{\pm 1.23}$	47.90M
CoMoL	$96.62_{\pm 0.53}$	$46.06_{\pm 4.09}$	$87.89_{\pm 0.38}$	$98.11_{\pm 0.35}$	$97.44_{\pm 0.68}$	$91.93_{\pm 0.91}$	$\mathbf{86.34}_{\pm 1.01}$	35.82M

Table 3: Experimental results on Qwen3-14B for mathematical reasoning benchmarks.

and to 4 for Qwen3-14B for all methods except MoLA, in order to prevent GPU out-of-memory (OOM) issues. To maintain parity in the number of trainable parameters, the ranks for the standard LoRA baseline, DenseLoRA, and FlyLoRA are configured as 16, 256, and 32, respectively. Across all configurations, the LoRA scaling factor α is set to $1 \times$ rank. All methods are trained for one epoch.

Main results. Tables 2 and 3 summarize the experimental results for the Qwen3-8B and Qwen3-14B backbones. Notably, CoMoL achieves a superior balance between parameter efficiency and overall performance. On the Qwen3-8B model, CoMoL reaches an average accuracy of 84.48%, outstripping the Standard LoRA baseline by a margin of 1.7 percentage points while maintaining parameter parity. It achieves state-of-the-art (SOTA) performance on the AddSub and AQuA benchmarks, surpassing LoRA by a significant 7.2 points on the latter. Furthermore, CoMoL consistently outperforms other competitive LoRA-based variants, such as SparseMoA and FlyLoRA. Impressively, it even exceeds the performance of MoLoRA and MoLA, despite these models possessing four times the number of trainable parameters. These findings substantiate the presence of significant expert redundancy in traditional MoE-LoRA architectures and demonstrate that CoMoL can effectively capture the collective expertise of multiple LoRA experts with negligible parameter overhead.

The performance gains of CoMoL are consistently maintained across different model scales, outperforming all baselines on both the Qwen3-8B and the larger Qwen3-14B variants. Such results highlight the exceptional stability and robustness of our CoMoL across varying parameter capacities.

4.2 Code Generation Task

Datasets. To evaluate the code generation proficiency of our method, we adopt the experimental protocol established by FlyLoRA (Zou et al.) and utilize the CodeAlpaca-20k (Chaudhary, 2023) dataset as our instruction-tuning corpus. The evaluation is conducted on the HumanEval (Chen, 2021) benchmark. To provide a holistic and comprehensive assessment of the model's coding capabilities, we report the pass@k metrics (where $k \in \{1, 5, 10\}$), which measure the probability of generating at least one functionally correct solution within k attempts.

Experimental setting. For the code generation task, we employ Qwen3-8B and Llama3.1-8B as our backbone models, adopting hyperparameter configurations analogous to those used in the mathematical reasoning experiments. Recognizing that various baselines exhibit high sensitivity to train-

	Llama3.1-8B				Qwen3-8B			
Methods	**Pass@1**	**Pass@5**	**Pass@10**	**# Parameters**	**Pass@1**	**Pass@5**	**Pass@10**	**# Parameters**
LoRA	26.22	54.41	67.07	23.06M	39.69	72.29	81.71	24.77M
MoLoRA	<u>34.15</u>	63.15	<u>72.56</u>	100.14M	43.78	73.49	82.93	107.35M
HydraLora	33.35	60.96	69.51	45.09M	<u>46.89</u>	75.61	84.15	49.55M
MoLA	28.72	56.53	67.68	100.14M	41.10	71.60	81.10	107.35M
AdaMoLE	21.52	44.58	56.10	101.12M	40.91	66.65	76.83	108.38M
DenseLoRA	31.58	58.71	67.07	26.74M	46.82	76.04	84.75	27.00M
FlyLoRA	32.32	<u>63.21</u>	71.95	23.06M	20.18	38.15	48.17	24.77M
CoMoL w/o CR	31.10	59.42	68.90	27.16M	43.23	**77.25**	**87.19**	33.40M
CoMoL	**35.00**	**63.65**	**73.17**	23.24M	**48.11**	<u>77.22</u>	<u>86.59</u>	24.97M

Table 4: Experimental results for the code generation task on Llama3.1-8B and Qwen3-8B. Pass@k is used as the evaluation metric. "# Parameters" indicates trainable parameters. Bold numbers indicate the highest performance scores, and underlined numbers indicate the second-highest performance scores.

ing duration in coding tasks, we perform model selection by searching for the optimal checkpoint based on validation set performance. Specifically, the models are trained for 5 epochs on Qwen3-8B and 1 epoch on Llama3.1-8B, ensuring that each method is evaluated at its peak proficiency.

Main results. CoMoL achieves or surpasses other methods with minimal trainable parameters, showing its parameter efficiency. As shown in the Table 4, methods with more trainable parameters, MoLA and AdaMoLE, actually perform worse than HydraLoRA and DenseLoRA with fewer parameters, indicating that methods with excessive parameters are prone to overfitting on the code generation task. In contrast, CoMoL maintains strong performance on both base models.

We observe that FlyLoRA's sparse learning approach with rank-wise expert activation in the up-projection matrix of LoRA performs well on Llama3.1-8B but poorly on Qwen3-8B. This is because FlyLoRA's sparsity reduces its learning capacity. The code generation training and test sets are both instruction fine-tuning datasets, which align closely with Llama3.1-8B's original capability distribution. However, Qwen3-8B is a long-thinking model, requiring the adaptation module to have sufficient capacity to modify its long-thinking behavior. Although CoMoL uses only a small number of expert parameters in the core space, its learning capacity remains stable, surpassing other MoE-LoRA methods and DenseLoRA.

4.3 Scaling with Rank

Figure 2 presents the performance and trainable parameter count of CoMoL and Standard LoRA across different ranks on the code generation task. **CoMoL consistently outperforms LoRA across all ranks, achieving the best performance at rank 16**. CoMoL w/o CR surpasses LoRA at all ranks except rank 32, and even exceeds CoMoL at rank 8. Notably, with a fixed number of experts, CoMoL maintains a parameter count comparable to LoRA while achieving consistent improvements.

4.4 Scaling with Expert Number

Table 5 reports the performance of CoMoL and CoMoL w/o CR with varying numbers of experts on the mathematical reasoning benchmark. On both Qwen3-8B and Qwen3-14B, the two methods achieve their best performance with 8 experts. Notably, both methods scale efficiently to 64 experts, whereas other approaches, such as HydraLoRA, encounter out-of-memory (OOM) issues with only 16 experts under the same GPU memory constraints.

Core Space Routing reduces router parameters while maintaining stable performance. Comparing the accuracy and trainable parameter count of CoMoL and CoMoL w/o CR across different numbers of experts, we observe that the two methods achieve comparable performance. However, CoMoL with Core Space Routing introduces virtually no additional router parameters as the number of experts increases. This demonstrates that Core Space Routing effectively leverages the low-rank property of LoRA.

Notably, scaling the number of experts to 64 does not yield consistent improvements over 8 experts. This observation mirrors a well-known phenomenon in Standard LoRA, where simply increasing the rank does not necessarily translate to pro-

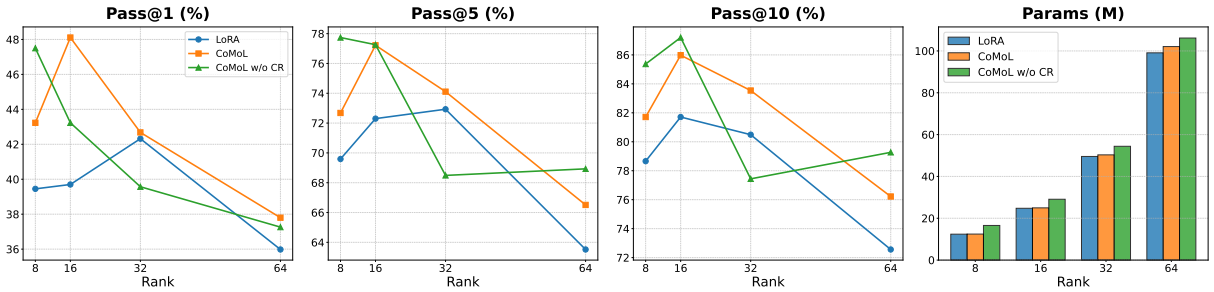

Figure 2: **Performance and trainable parameter comparison between CoMoL and LoRA across different ranks on the code generation benchmark.**

	CoMoL w/o CR		CoMoL	
# Experts	**Math**	**# Param**	**Math**	**# Param**
Qwen3-8B				
8	84.47	33.39M	84.48	25.16M
16	82.50	42.02M	84.00	25.56M
32	82.49	59.28M	82.24	26.34M
64	84.50	93.78M	82.72	27.91M
Qwen3-14B				
8	86.08	47.90M	86.34	35.82M
16	85.92	60.46M	86.24	36.26M
32	86.13	85.52M	86.06	37.13M
64	86.32	135.66M	86.23	38.87M

Table 5: Comparison between CoMoL and CoMoL w/o CR with varying numbers of experts on mathematical reasoning. "# Param" indicates trainable parameters.

Method	**Training Time**	**Inference Time**
LoRA	00:56:47	18.40
MoLoRA	04:44:50	19.67
HydraLoRA	04:15:15	19.16
MoLA	07:59:02	30.03
AdaMoLE	08:18:54	40.78
CoMoL w/o CR	02:33:27	18.62
CoMoL	02:19:55	18.57

Table 6: Wall-clock time comparison of training and inference across methods. Training time (HH:MM:SS) is measured for one epoch on CodeAlpaca-20k (batch size 1). Inference time (minutes) is the cumulative duration for 10 generations per sample on HumanEval (batch size 16). All MoE-LoRA variants use k=8 experts.

portional performance gains (Chen et al., 2022). We attribute this to diminishing returns in expert scaling under data-constrained settings: when the quality and quantity of training samples are limited, a larger pool of experts cannot be adequately optimized, resulting in under-training and degraded specialization.

4.5 Computational Efficiency Analysis

Table 6 reports the wall-clock training and inference times of CoMoL alongside representative baselines under controlled conditions. To ensure a fair comparison, we measure training time as the total duration to complete one epoch on the CodeAlpaca-20k dataset with a batch size of 1, and inference time as the cumulative duration for 10 generations per sample on the HumanEval benchmark. Following the experimental protocol described in our main experiments, all MoE-based variants employ k=8 experts.

As shown in the table, CoMoL achieves substantially lower training cost than all competing MoE-LoRA methods, roughly half that of MoLoRA and HydraLoRA, while maintaining inference latency comparable to standard LoRA and consistently below that of other MoE-LoRA counterparts. The advantage is particularly pronounced against the two sparse MoE-LoRA methods (MoLA and AdaMoLE), whose training times exceed CoMoL's by approximately $4\times$ and whose inference times are $1.5\times$ to $2\times$ longer. These empirical wall-clock measurements corroborate our theoretical FLOPs analysis presented in Section 3.2, confirming that the efficiency gains of CoMoL translate directly into practical speedups.

5 Related work

Sparse MoE-LoRA. Sparse MoE-LoRA has been studied from multiple angles, including routing regularization, expert diversity, and redundancy reduction. SiRA (Zhu et al., 2023) adopts top-k routing with per-expert capacity constraints and gating dropout to stabilize routing and alleviate overfitting. MoELoRA (Luo et al., 2024) encourages expert specialization via contrastive learning

to mitigate random routing. MoLA (Gao et al., 2024) analyzes layer-wise redundancy by allocating different numbers of LoRA experts across layers, showing higher redundancy in lower layers. AdaMoLE (Liu and Luo, 2024) further introduces token-wise adaptive expert activation via a threshold network integrated into top-k routing.

Soft-weighted MoE-LoRA. These variants replace hard top-k selection with continuous mixtures over LoRA experts. MoLoRA (Zadouri et al., 2023) adopts token-level soft routing to combine experts per token. LoRAMoE (Dou et al., 2024) uses a linear router to integrate LoRA experts and mitigate forgetting of world knowledge. MOLE (Wu et al., 2024) enables efficient composition of multiple trained LoRAs by training only the per-layer router. HydraLoRA (Tian et al., 2024) further improves parameter efficiency via an asymmetric design with a shared matrix.

Soft-merging MoE methods. In contrast to methods that aggregate expert outputs, some approaches (AdaMix, SMEAR, MoV) adopt an alternative strategy of first merging expert parameters via routing before performing computation, thereby reducing computational overhead. For instance, AdaMix (Wang et al., 2022) employs a mixture of adapters with stochastic routing, while SMEAR (Muqeeth et al., 2024) proposes merging experts through a learned router. Notably, both AdaMix and SMEAR operate at the instance level. MoV (Zadouri et al., 2023), on the other hand, is a token-level expert parameter merging method. However, it is limited to IA3 (Liu et al., 2022) experts, which scale hidden states in the base model using trainable vectors. Currently, no existing method is capable of merging LoRA experts at the token level.

6 Conclusion

In this work, we address the limited parameter efficiency and coarse-grained adaptation of existing MoE-LoRA architectures by proposing Core Space Mixture of LoRA (CoMoL), a novel framework built upon two key components: Core Space Experts and Core Space Routing. By operating entirely within a compact core space, CoMoL enables fine-grained, input-adaptive expert merging while maintaining parameter and computational costs comparable to Standard LoRA. Moreover, projecting the routing network into the same low-rank space further reduces overhead without compromising expressiveness. Extensive experiments demonstrate that CoMoL consistently outperforms existing MoE-LoRA methods across diverse tasks, highlighting its effectiveness, scalability, and practical value for parameter-efficient adaptation of LLMs.

Limitations

Current research on MoE-LoRA and other PEFT methods primarily focuses on improving parameter efficiency, while the learning capacity of these methods across different fine-tuning scenarios remains underexplored. For instance, FlyLoRA exhibits substantially different performance on the code generation task when applied to Llama3.1-8B versus Qwen3-8B. There currently exists no systematic benchmark or evaluation framework to assess the learning capacity of different PEFT methods. Although CoMoL demonstrates strong performance across various tasks (e.g., mathematical reasoning and code generation), different model families, and varying model scales, a more comprehensive understanding of the capacity boundaries of different PEFT methods remains to be established. We leave this systematic investigation as future work.

Acknowledgments

This work was supported by the National Key R&D Program of China (2025ZD0123100), the NSFC (62272411), the Key R&D Projects in Zhejiang Province (No. 2025C01030, No. 2025C01128), the Zhejiang NSF (LRG25F020001), the Postdoctoral Fellowship Program of CPSF (GZC20251077), the China Postdoctoral Science Foundation (2025M781525), Fundamental Research Funds for the Central Universities (226-2025-00057), and the Tencent WeChat Rhino-Bird Special Research Program (Tencent WXG-FR-2023-10).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jie Cao, Tianwei Lin, Hongyang He, Rolan Yan, Wenqiao Zhang, Juncheng Li, Dongping Zhang, Siliang Tang, and Yueting Zhuang. 2025. Moa: Heterogeneous mixture of adapters for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2506.05928*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2612–2626.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. LoRAMoE: Alleviate World Knowledge Forgetting in Large Language Models via MoE-Style Plugin. *arXiv preprint*. ArXiv:2312.09979 [cs].

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Baochen Sun, Ruibo Liu, Daiyi Peng, Yawen Zhang, Xiaoyuan Guo, Jie Yang, and V. S. Subrahmanian. 2024. Higher Layers Need More LoRA Experts. *arXiv preprint*. ArXiv:2402.08562 [cs].

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a Unified View of Parameter-Efficient Transfer Learning. *arXiv preprint*. ArXiv:2110.04366 [cs].

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 523–533.

N. Houlsby, A. Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and S. Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. *ArXiv*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint*. ArXiv:2106.09685 [cs].

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models. *arXiv preprint*. ArXiv:2304.01933 [cs].

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Tianwei Lin, Jiang Liu, Wenqiao Zhang, Zhaocheng Li, Yang Dai, Haoyuan Li, Zhelun Yu, Wanggui He, Juncheng Li, Hao Jiang, and 1 others. 2024. Teamlora: Boosting low-rank adaptation with expert collaboration and competition. *arXiv preprint arXiv:2408.09856*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Zefang Liu and Jiahua Luo. 2024. AdaMoLE: Fine-Tuning Large Language Models with Adaptive Mixture of Low-Rank Adaptation Experts. *arXiv preprint*. ArXiv:2405.00361 [cs].

Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. 2024. MoELoRA: Contrastive Learning Guided Mixture of Experts on Parameter-Efficient Fine-Tuning for Large Language Models. *arXiv preprint*. ArXiv:2402.12851 [cs].

Lin Mu, Xiaoyu Wang, Li Ni, Yang Li, Zhize Wu, Peiquan Jin, and Yiwen Zhang. 2025. Denselora: Dense low-rank adaptation of large language models. *arXiv preprint arXiv:2505.23808*.

Mohammed Muqeeth, Haokun Liu, and Colin Raffel. 2024. Soft Merging of Experts with Adaptive Routing. *arXiv preprint*. ArXiv:2306.03745 [cs].

Aniello Panariello, Daniel Marczak, Simone Magistri, Angelo Porrello, Bartłomiej Twardowski, Andrew D Bagdanov, Simone Calderara, and Joost van de Weijer. Accurate and efficient low-rank model merging in core space. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

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

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. 2024. HydraLoRA: An Asymmetric LoRA Architecture for Efficient Fine-Tuning. *arXiv preprint*. ArXiv:2404.19245 [cs].

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-Adaptations for Parameter-efficient Model Tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Xun Wu, Shaohan Huang, and Furu Wei. 2024. Mixture of LoRA Experts. *arXiv preprint*. ArXiv:2404.13628 [cs].

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

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. 2023. Pushing Mixture of Experts to the Limit: Extremely Parameter Efficient MoE for Instruction Tuning. *arXiv preprint*. ArXiv:2309.05444 [cs].

Yun Zhu, Nevan Wichers, Chu-Cheng Lin, Xinyi Wang, Tianlong Chen, Lei Shu, Han Lu, Canoee Liu, Liangchen Luo, Jindong Chen, and Lei Meng. 2023. SiRA: Sparse Mixture of Low Rank Adaptation. *arXiv preprint*. ArXiv:2311.09179 [cs].

Heming Zou, Yunliang Zang, Wutong Xu, Yao Zhu, and Xiangyang Ji. Flylora: Boosting task decoupling and parameter efficiency via implicit rank-wise mixture-of-experts. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.