LoRACoE: Improving Large Language Models via Compostion-based LoRA Expert

Anonymous ACL submission

Abstract

The Mixture of Experts (MoE) architecture improves large language models (LLMs) by utilizing sparsely activated expert sub-networks with a routing module, yet it typically demands high training cost. Previous work introduces parameter-efficient fine-tuning (PEFT) modules, e.g., LoRA, to achieve a lightweight MoE for efficiency. However, they construct static experts by manually splitting the LoRA parameters into fixed groups, which limits flexibility and dynamism. Furthermore, this manual partitioning also hinders the effective utilization of well-initialized LoRA modules. To tackl the challenges, we first delve into the parameter patterns in LoRA modules, revealing that there exists task-relevant parameters that are concentrated along the rank dimension. Based on this, we redesign the construction of experts and propose the LoRACoE (<u>LoRA</u> <u>C</u>omposition <u>o</u>f <u>E</u>xperts) method. Specifically, when confronted with a task, it dynamically builds experts based on rank-level parameter composition, i.e., experts can flexibly combine rank-level parameters in LoRA module. Extensive experiments demonstrate that compared to other LoRA-based MoE methods, our method achieves better task performance across a broader range of tasks.

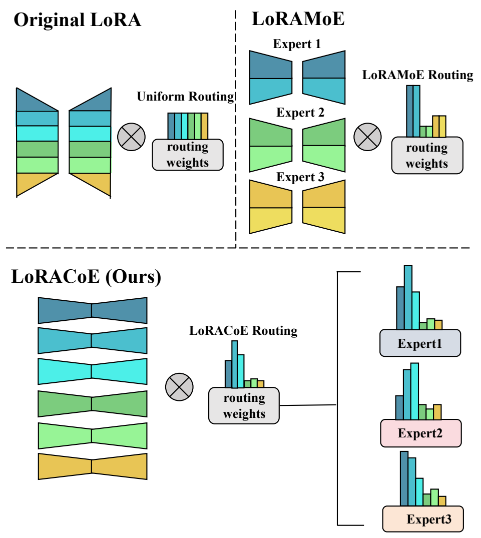

Figure 1: Comparison of construction in LoRA, LoRAMoE, and LoRACoE (Ours). Our method is based on a finer-grained partitioning of LoRA parameters, incorporating a redesigned expert mechanism that dynamically composite LoRA parameters at output, thereby achieving improved task performance.

1 Introduction

Recent advanvements show that the Mixture of Experts (MoE) architecture (Fedus et al., 2022; Jiang et al., 2024; Liu et al., 2024; Pióro et al., 2024; Yu et al., 2024) enhances the performance of large language models (LLMs) over traditional dense architectures (Chen et al., 2024a). In MoE, the model's parameters are organized into groups known as "experts". During each forward pass, these experts are sparsely activated via a routing mechanism (Fedus et al., 2022; Jiang et al., 2024), reducing the inference cost of LLMs. However, during fine-tuning (Wei et al., 2021; Taori et al., 2023), all model parameters (or experts) still need to be optimized, which makes the training process inefficient (Jiang et al., 2024; Liu et al., 2024; Pióro et al., 2024; Yu et al., 2024).

To achieve lightweight, training-efficient MoE, recent work integrates MoE with Parameter-Efficient Fine-Tuning (PEFT) techniques (Hu et al., 2021; Dettmers et al., 2023; Xu et al., 2024), represented by LoRA-based MoE (Huang et al., 2023; Zhu et al., 2023; Dou et al., 2024; Feng et al., 2024b; Li et al., 2024). In LoRA-based MoE, only the added low-rank adapters are updated during fine-tuning. However, they manually partition low-rank adapters into parameter groups based on the rank dimension to define experts (see Figure 1). This static construction approach fixes the number of experts and the parameters assigned to each expert, limiting the flexibility and dynamism of the

MoE architecture (Dou et al., 2024; Ning et al., 2024). Furthermore, such approach also hinders the effective utilization of well-initialized LoRA modules (Hayou et al., 2024), thereby increasing the training cost.

In this work, we rethink the design of experts in LoRA-based MoE to achieve a lightweight, dynamic, and flexible architecture. We begin by analyzing parameter importance (Molchanov et al., 2019; Zhang et al., 2022, 2024) within LoRA modules and observe that each parameter holds varying importance across different tasks. More precisely, this variation occurs along the rank dimension, i.e., for certain tasks, parameters in some ranks are more crucial than those in other ranks (see Section 2.2). This phenomenon inspires us to re-weight the outputs of different ranks in the LoRA module, allowing for better utilization of task-related parameters.

Based on this insight, we propose (LoRA Composition of Experts) that can provides a flexible and dynamic construction of experts. Instead of manually partitioning parameters to construct experts, it defines an expert as a weighted combination of ranks. To be more specific, when confronted with a task, the route module predicts the importance weight of each rank, and an expert is build upon the rank parameters weighted by these predictions. We state that this architectural shift from partitioning to compositional expert construction provides finer control over LoRA parameters, optimizing their utilization for better performance.

Extensive experiments on commonsense reasoning and mathematical tasks, conducted across six backbone models (Touvron et al., 2023; Zhu et al., 2024; Yang et al., 2024) and thirteen datasets (Hu et al., 2023; Mitra et al., 2024) demonstrate that LoRACoE outperforms both the original LoRA method and LoRA-based MoE approaches by significant margins. Our contributions are summarized as follows: [1]

1. We reveal the task-specific importance distribution across the rank dimension within the original LoRA method through parameter-importance analysis.

2. Based on the findings, we propose a new expert construction method called LoRACoE that shifts from static parameter partitioning to dynamic parameter combination.

3. We conduct extensive experiments on commonsense reasoning and mathematical tasks across different six models and thirteen datasets to demonstrate the effectiveness of our method.

[1]We will release our implementations to the public.

2 Preliminaries and Observations

2.1 Preliminaries

Low-Rank adaptation. Low-Rank Adaptation (LoRA) (Hu et al., 2021) is a parameter efficient fine-tuning technique for large pre-trained models. Traditional fine-tuning approaches update all model parameters, which can be computationally expensive. LoRA addresses this by inserting trainable low-rank matrices into the FFN layers or attention matrices of models to capture the necessary updates. This approach significantly reduces the number of trainable parameters, thereby lowering computational and storage costs.

Concretely, given a pre-trained weight matrix $W_0 \in R^{d\times k}$, LoRA approximates the update to W_0 as the product of two low-rank matrices,and the updated weights W are calculated through:

$$W = W_0 + BA,$$

where $B \in R^{d\times r}$ and $A \in R^{r\times k}$, with $r \ll \min(d, k)$. During fine-tuning, only the matrices A and B are updated, while the original weights W_0 remain frozen, making the fine-tuning process both memory and computation-efficient.

Mixture of experts. Mixture of Experts (MoE) (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2020) utilizes a sparse parameter activation pattern, enabling the model to scale the number of parameters while maintaining a constant computational cost. MoE architecture divides the parameters of the traditional transformer Feed-Forward Network (FFN) layer into N experts, denoted as $\{E_i\}_{i=1}^{N}$, and designs a corresponding router g. For a given input x, the output y of the MoE layer is a weighted sum of outputs from N experts:

$$y = \sum_{i=1}^{N} g_i(x)E_i(x),$$

where $E_i(x)$ is the output of expert i, and $g_i(x)$ is the routing function's output. The routing function varies depending on the specific routing algorithm design.

Parameter importance. In previous studies on the capabilities of LLM parameters, researchers have identified regions within the model parameters that are highly task-relevant (Zhang et al., 2024; Chen et al., 2024b). This insight motivates us to investigate similar regions within the LoRA modules. We adopt a commonly used method from previous work (Molchanov et al., 2019; Zhang et al., 2022) on parameter sensitivity analysis to apply to the LoRA modules.

The assumption in these studies is that removing a parameter (by setting its value to zero) and evaluating its impact on the model's loss function can reveal its importance. Specifically, given a dataset $\mathcal{D}$ and a set of model parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d] \in \mathbb{R}^d$, with θ_j representing the j-th parameter. During training, the objective is to minimize the loss function L, which depends on both the dataset $\mathcal{D}$ and the model parameters $\boldsymbol{\theta}$. The importance of the j-th parameter θ_j is denoted as $I_j(\theta)$. The importance of a parameter can be quantified by the error introduced when that parameter is removed, which, under the i.i.d. assumption, can be approximated by calculating the squared difference in loss before and after removing the parameter:

$$\mathcal{I}_j(\theta) = |\mathcal{L}(\mathcal{D}, L(\theta)) - \mathcal{L}(\mathcal{D}, \theta|\theta_j = 0)|\,. \quad (1)$$

However, calculating this importance by removing each parameter and measuring the change in loss is computationally expensive, particularly when the model has a large number of parameters. Therefore, following prior work (Molchanov et al., 2019; Zhang et al., 2022), we can use the Taylor expansion formula for $\mathcal{L}$ at $\theta_j = 0$:

$$\begin{aligned}\mathcal{L}(\mathcal{D}, \theta) &= \mathcal{L}(\mathcal{D}, \theta|\theta_j = 0) + \\ &\frac{\partial \mathcal{L}}{\partial \theta_j}(\theta_j - 0) + \frac{1}{2!}\frac{\partial^2 \mathcal{L}}{\partial \theta_j^2}(\theta_j - 0)^2 + \cdots \end{aligned} \quad (2)$$

After performing the Taylor expansion, calculating the higher-order terms still remains a resource-intensive task. Therefore, we approximate the importance scores using only the first-order term of the Taylor expansion:

$$I_j(\theta) \approx \left|\frac{\partial L}{\partial \theta_j} \cdot \theta_j\right| \quad (3)$$

2.2 Observation of LoRA Modules

To investigate the properties of the LoRA module parameters, we trained LoRA using datasets from multiple commonsense reasoning and mathematical tasks. For the commonsense tasks, we select 75k samples from the commonsense task training set of Hu et al. (2023), which includes eight datasets: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-c, ARC-e (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). For the mathematical tasks, we curated 75k samples from the OrcaMath (Mitra et al., 2024) to form our math task dataset. We add the LoRA module to different components of the model, including the q_proj, k_proj, v_proj of attention modules, as well as the up_proj and down_proj of FFN modules. The LoRA modules are then trained using the constructed training set. After training the LoRA modules, we employ the method described in Section 2.2 to compute the parameter importance scores on the validation set for different tasks, ultimately obtaining importance scores for each parameter in relation to the tasks.

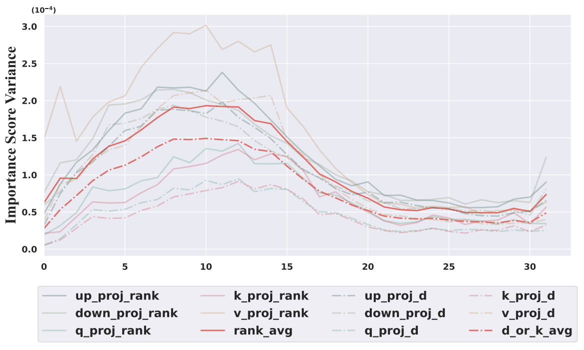

Figure 2: Average variance of the importance of LoRA module parameters across different dimensions. Here, *rank_avg* and *d_or_k_avg* represent the average variance calculated in the rank dimension and the input or output dimensions of the LoRA module, respectively.

Based on the importance scores obtained from the aforementioned experiments, we can draw the following conclusions:

Distribution patterns of task-relevant parameters. We computed the average variance of the parameters of the LoRA module at different positions, considering both the rank dimension and the input or output dimensions of the LoRA module. The statistical results are illustrated in the Figure 2. we observe that the variance along the rank dimension is larger compared to the variance along the input or output dimensions. Compared to the input or output dimensions, the distribution of LoRA parameters in the rank dimension is more uneven. This indicates that parameter importance tends to be more concentrated along the rank dimension rather than distributed along the input or output dimensions of the LoRA module. This phenomenon provides insights for the subsequent design of more effective utilization methods for LoRA parameters in Section 3.

Task-specific parameter activation patterns. We analyze the importance score patterns of parameters across different tasks based on the importance scores. From the parameter importance score correlation heatmaps across tasks in Figure 3, we observe significant positive and negative correlations in parameter importance between different tasks. For instance, in the BoolQ task, the distinct answer patterns compared to other question types lead to a notable divergence in parameter importance patterns relative to other commonsense tasks. Similarly, for math-related tasks, the activation patterns tend to show more negative correlations with commonsense tasks, owing to the differences in task nature. For other commonsense tasks with similar answer patterns, the parameter importance patterns exhibit a high degree of correlation, indicating that the model employs similar parameter utilization patterns when performing these tasks.

Correlation of parameter importance across ranks. We analyzed the correlation of parameter importance score in different LoRA ranks across all tasks. This analysis aims to illustrate the correlation in importance score patterns of different parameters within the LoRA module as influenced by varying inputs. As shown in the Figure 3, the parameter importance across different ranks exhibits either positive or negative correlations when performing different tasks. This observation suggests that parameters at different rank levels may have either synergistic or conflicting relationships. The original LoRA method, which does not apply weightings to parameters, may lead to suboptimal utilization of parameters.

Based on these observations, which reveal a task-specific concentration of parameter importance at the rank level, we can assume that the LoRA module naturally learns a rank-level importance distribution during training. This phenomenon suggests a certain "specialized" correspondence between different tasks and the parameters within the LoRA module. Therefore, given the inherent sensitivity or “expertise” exhibited by the LoRA module’s parameters, the conventional approach of manually dividing the LoRA parameters into expert groups at the rank level needs to be reconsidered.

2.3 Limitations of Partition-based LoRAMoE

Given a LoRA module consists of matrices $B \in R^{d\times r}$ and $A \in R^{r\times k}$, partition-based LoRAMoE methods will divide B and A into N sets of parameters. Therefore we acquire the result of matrices, $\{B_i\}_{i=1}^N$ and $\{A_i\}_{i=1}^N$. Expert E_i composes of a pair of $B_i \in R^{d\times r/N}$ and $A_i \in R^{r/N\times k}$ matrices.

$$E_i(x) = B_i A_i x \tag{4}$$

And with the pre-trained weight matrix $W_0 \in R^{d\times k}$, N partition-based LoraMoE experts, denoted as $\{E_i\}_{i=1}^N$ and a gating function $g(x)$, the output y of conventional partition-based methods typically follow this approach:

$$y = W_0 x + \sum_{i=1}^{N} g(x)_i E_i(x) \tag{5}$$

Under our observations and assumptions, this method presents two significant drawbacks: (1) By forcibly binding rank parameters to form experts, the granularity of utilization controling across parameters during the learning process is reduced. (2) The definition of experts in previous work is limited. Since experts are constructed based on rank partitioning, the routing and weighted output of these experts leverage only the parameters within the ranks they control, without considering the relationships between the parameters they control and those in other ranks. This constrains the flexibility and effectiveness of the expert models.

3 LoRA Compositional Experts

3.1 Rank Wise Parameter Paritioning

Based on the observations from the LoRA modules trained on multiple tasks in Section 2.2, we aim to develop a new expert design paradigm. First, following the approach in (He, 2024), which decomposes the FFN layer of Transformer models into vectors of dimension 1, we decompose the A and B matrices of the LoRA module into $\{A_i \in R^{1\times k}\}_{i=1}^r$ and $\{B_i \in R^{d\times 1}\}_{i=1}^r$. The fine-grained partition of LoRA parameters enables us to effectively control the model’s capabilities with the finest granularity possible. Meanwhile this finer-grained, non-binding partitioning allows us to avoid the need for capability recovery, as required

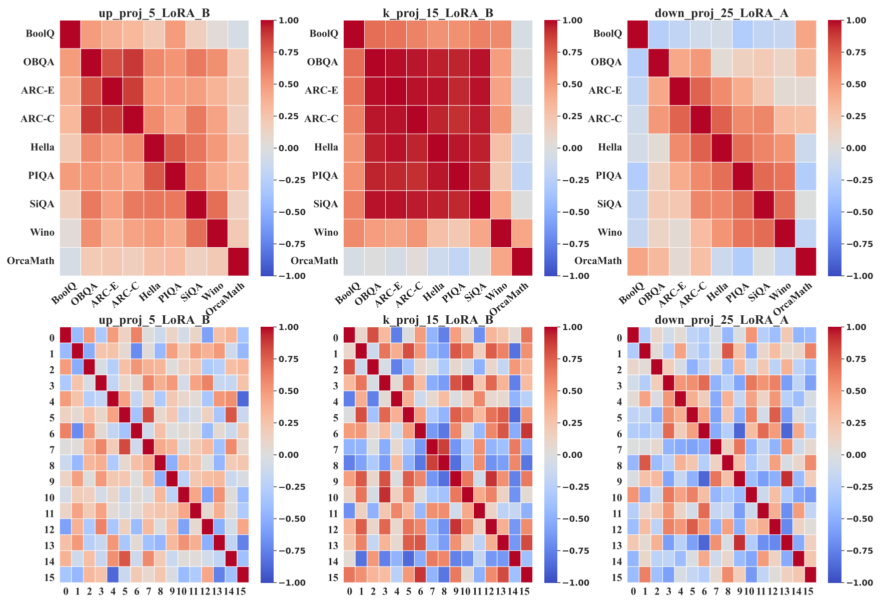

Figure 3: Importance score correlation heatmaps of LoRA parameters for three modules w.r.t. different tasks (top) and w.r.t. different ranks (bottom).

in upcycling-based MoE models (Zhu et al., 2024). Instead, we can achieve further optimization based on an well-initialized LoRA parameters.

3.2 Composition-Based Expert Construction

According to Section 2.2, LoRA parameters exhibit varying importance patterns across different tasks, and there exists a correlation in importance between different rank parameters. Consequently, in previous partition-based LoRAMoE algorithms, experts could only account for the importance of local parameters, failing to accurately capture the correlations among global parameters. Thus, by incorporating rank-level parameter partitioning, we propose LoRACoE, a design pattern based on combination-based experts. In LoRACoE, for a given partitioned LoRA matrices $\{A_i \in R^{1\times k}\}_{i=1}^{r}$ and $\{B_i \in R^{d\times 1}\}_{i=1}^{r}$, each expert outputs a linear combination G_i based on the input x. For E experts, we obtain E sets of linear combinations.

To implement this expert mechanism, we design a corresponding routing module $W_{route} \in R^{d\times E\times r}$. For the input to expert E_i, the corresponding parameter group $W_{route_i} \in R^{d\times r}$ from the routing module is used to obtain the weights G_i as follows:

$$G_i(x) = Softmax(W_{route_i} x)$$

Based on the parameter set $\{A_i \in R^{1\times k}\}_{i=1}^{r}$ and $\{B_i \in R^{d\times 1}\}_{i=1}^{r}$ of the LoRA module and the composition weights of rank-level parameters G from different experts, the LoRACoE module will output as follows:

$$LoRACoE(x) = W_0 x + \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{r} G_{ij} B_j A_j x$$

Here, x represents the input to the LoRACoE layer, B_j and A_j represent the j-th vectors in the decomposed LoRA B and A matrices, G_{ij} is the weight of the i-th expert for the j-th rank, and W_0 refers to the pre-trained weights. The output of the weighted LoRA module is then merged with the output of the pre-trained weight as the final output of the LoRACoE layer.

3.3 Training Procedure of LoRACoE

To ensure effective initialization of the LoRA module and a stable training process for LoRACoE, we followed a two-phase training procedure: LoRA warm-up and joint training. First, for LoRA warm-up, we perform a training of a standard LoRA module. This step helps achieve stable and efficient convergence in the final model (Dua et al., 2021). Next, we conduct joint training of both the LoRA parameters and the routing module.

4 Experiments

4.1 Experimental Setup

Dataset. We construct a multi-task dataset based on commonsense and mathematical reasoning

Model	Method	Commonsense	Math	Insturction Following	Avg.
Llama2-7b	FT	65.07	85.69	49.52	66.76
	LoRA	63.26	75.42	55.75	64.81
	LoRAMoE	74.01	82.94	**57.79**	71.58
	LoRACoE	**80.83**	**86.49**	56.47	**74.60**
Llama2-13b	FT	74.49	83.63	61.03	73.05
	LoRA	68.17	82.36	61.27	70.60
	LoRAMoE	79.18	87.10	62.23	76.17
	LoRACoE	**81.53**	**90.50**	**65.22**	**79.08**
Llama3-8b	FT	65.08	85.78	56.95	69.27
	LoRA	68.11	82.69	65.34	72.05
	LoRAMoE	78.89	88.98	**67.38**	78.42
	LoRACoE	**81.09**	**92.28**	64.62	**79.33**
Qwen2-0.5b	FT	53.98	**76.13**	**37.29**	55.80
	LoRA	55.15	69.57	35.61	53.44
	LoRAMoE	57.50	72.35	31.89	53.91
	LoRACoE	**64.24**	74.40	35.97	**58.20**
Qwen2-1.5b	FT	65.15	82.79	41.72	63.22
	LoRA	73.17	85.03	48.08	68.76
	LoRAMoE	73.81	85.98	46.64	68.81
	LoRACoE	**74.91**	**86.18**	**48.68**	**69.92**
Qwen2-7b	FT	**86.15**	93.71	61.63	80.50
	LoRA	84.78	93.88	62.95	80.54
	LoRAMoE	85.18	94.36	62.11	80.55
	LoRACoE	85.97	**94.71**	**64.26**	**81.65**

Table 1: Evaluating results of different methods on commonsense, math and Instruction following tasks. The best results are in **bold**.Our method is marked in blue.

tasks. For commonsense tasks, we randomly select 75k examples from the commonsense dataset in Hu et al. (2023) as the training set for common sense tasks. For mathematical tasks, we randomly sample a 75k subset from Mitra et al. (2024) as the training set for mathematical tasks. To evaluate the effectiveness of the method, we select the test set corresponding to the training set as the benchmark for commonsense tasks.While for mathematical tasks, we chose GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), AddSub (Hosseini et al., 2014), MultiArith (Roy and Roth, 2016), SingleEq (Koncel-Kedziorski et al., 2015). Additionally, to better test the generalization capability of our method, we also trained our approach on instruction-following tasks using datasets from prior work (Dong et al., 2024).We selected IFEval (Zhou et al., 2023) as the test set for instruction-following tasks.

Models. As for the base models, we select LLaMA2-7B, LLaMA2-13B (Touvron et al., 2023), LLaMA3-8B (Dubey et al., 2024), Qwen2-0.5B, Qwen2-1.5B and Qwen2-7B (Yang et al., 2024) to validate the effectiveness of the method on base model training at different parameter scales.

Baselines. For comparison methods in the peft framework, we select LoRA (Hu et al., 2021) and the partition-base LoRAMoE describe in Section 2.2.To further substantiate the effectiveness of our LoRACoE method, we included HydraLoRA(Tian et al., 2024), a variant of LoRAMoE, as a comparative baseline.The relevant results can be found in the Appendix9. We also perform fine-tuning of all parameters in models for comparison.

Implement details. In our experiments, we set the rank for LoRA, LoRAMoE, and LoRACoE to 16, with α set to 32. For LoRAMoE, we configured the number of experts to 4, while for LoRACoE, we set it to 2. For the LLaMA series models, we used a batch size of 16, and for the Qwen2 series models, we set the batch size to 32.To ensure fairness in training, we set the number of training epochs for all PEFT methods to 4. For the LoRACoE method requiring two-stage training, we conduct two epochs of initialization training followed by two epochs of joint training. Detailed hyperparameters can be found in Appendix A.To achieve better inference and training efficiency for LoRACoE, we

# Experts	# LoRA Rank	CS Avg.	Math Avg.
2	8	79.93	84.87
2	16	**80.46**	86.78
2	32	79.02	**87.36**
2	16	**80.46**	86.78
4	16	80.22	86.69
8	16	80.31	**86.93**
2(inference w/o router)	16	80.02	86.11

Table 2: The ablation of experts and rank on Commonsense (CS) and Math tasks.

Method	CS Avg.	Math Avg.
LoRA	63.25	75.42
LoRAMoE	74.01	82.94
LoRACoE		
with LoRA & Router warmup	80.82	86.49
with LoRA warmup	80.46	86.78
without warmup	-	-

Table 3: Ablation on initialization of LoRA module and router module on commonsense(CS) and mathematical reasoning tasks.

performed computation optimizations tailored to the architectural characteristics of LoRACoE and the FFN layers. The detailed optimization methods and the resulting efficiency improvements are thoroughly analyzed in the Appendix C.

4.2 Experimental Results and Discussion

Main results. The results of our main experiments are in Table 1. Generally, we can observe that in terms of the overall performance across the three tasks, the methods with MoE architecture consistently outperform the standard LoRA approaches across different models, demonstrating the promise of the sparse architecture. Secondly, our composition-based LoRACoE achieves significant performance improvements over the partition-based LoRAMoE. Specifically, it outperforms LoRAMoE by 3.02%, 2.91%, and 0.91% on the Llama2-7b, Llama2-13b, and Llama3-8b models, respectively; on Qwen2 serie of models, LoRACoE outperforms LoRAMoE by 4.29%, 1.11%, and 1.01% on the Qwen2-0.5b, Qwen2-1.5b, and Qwen2-7b models, respectively. This highlights the advantages of this dynamic and flexible expert construction approach across all base model sizes, demonstrating the robustness and scalability of our method..

Ablation on rank number. We conduct an ablation study on LoRA rank using the Llama2 7B model on commonsense and mathematical reasoning tasks. The results are presented in Table 2. We find that for mathematical tasks, performance improves as the rank increases, but this trend does not hold for commonsense tasks. Considering that higher ranks result in greater computational overhead, we set the rank to 16 in our main experiments.

Influence of parameter initialization in LoRA module and routing module. To achieve a balance between performance and training efficiency in LoRACoE, we only train the LoRA module for initialization while leaving other modules randomly initialized. Here, we conduct two ablation experiments: (1) applying warm-up initialization to the LoRA module and the routing module; (2) skipping warm-up initialization for both the LoRA and routing modules. The results are shown in Table 3. Note that we did not conduct experiments with only the routing module warmed up because the routing module cannot provide meaningful weighted information to an untrained LoRA module.

The results show that without warm-up initialization for both the LoRA and routing modules, the training becomes unstable, and the final training loss fails to converge, so we do not report the performance. In contrast, when only the LoRA module is warmed up, the performance remains stable, highlighting the importance of warm-up initialization for the LoRA module.

Parameter importance characteristics of different methods. To investigate the dynamic characteristics of rank-level parameter importance across different methods after training, we analyzed parameter importance for LoRA, LoRAMoE, and LoRACoE on HellaSwag (Zellers et al., 2019) and BoolQ (Clark et al., 2019). The analysis results are shown in the Figure 4.

First, both LoRAMoE and LoRACoE, allow parameters from different regions to exhibit varying importance to different tasks. This means they can utilize distinct parameters based on the task confronted. In contrast, the original LoRA model does not show a clear distinction in important parameter regions between two different tasks.

Additionally, concerning the number of significantly important ranks, it is evident from the graph that LoRACoE can flexibly adjust the quantity of important parameters compared to LoRAMoE methods. Specifically, on the better-performing HellaSwag task, LoRACoE achieves superior performance using fewer important parameters, while on the poorer-performing BoolQ task, LoRACoE utilize more important ranks to hold significant importance when learning this task, which results in better performance. However, LoRAMoE does not show any significant rank utilization ratio difference between the two tasks. The above observations may interpret the significant performance gap we observed in our experiments.

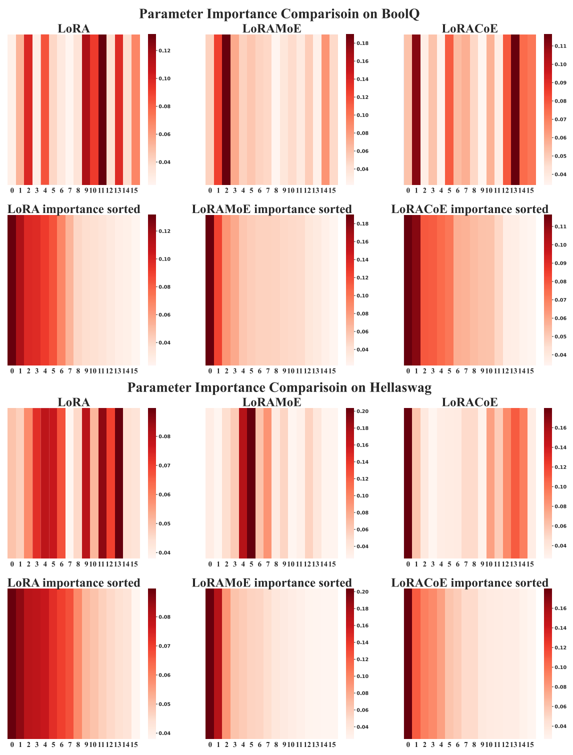

Figure 4: Parameter importance comparison across LoRA, LoRAMoE, and LoRACoE on HellaSwag and BoolQ datasets. We sort the importance to provide a more intuitive visual representation. This figure demonstrates LoRACoE dynamically adjusts significant parameters, optimizing HellaSwag performance with fewer important ranks and adapting more for BoolQ to boost learning.

5 Related Work

5.1 Mixture of Experts for LLMs

The Mixture of Experts (MoE) architecture was first introduced in Jacobs et al. (1991), aiming to reduce interference between different types of samples by employing multiple expert networks, with a gating network controlling their learning. The sparse-activated MoE design paradigm (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Jiang et al., 2024) has significantly reduced computational costs by limiting the number of activated model parameters, thereby enabling better scalability. To make more effective use of pre-trained large language models (LLMs) as initialization points, a series of upcycle methods have been proposed (Cai et al., 2024; Wei et al., 2024; Zhu et al., 2024). By leveraging pretrained models for initialization, these methods not only achieve better convergence and training stability but also reduce the computational resources required. In our work, we adopt a similar upcycle approach. By optimizing the expert design and preserving the properties of initialized LoRA modules, we effectively utilize these initialization points.

5.2 PEFT-based MoE

Recent work on PEFT-based MoE combines the effectiveness of Mixture of Experts (MoE) in multi-task scenarios with the efficiency of PEFT, showcasing superior performance. Due to the expert nature of MoE, some studies (Huang et al., 2023; Wu et al., 2024; Feng et al., 2024a; Ren et al., 2024) have developed optimization algorithms that combine LoRA modules trained on different tasks, enhancing the generalization capability of multiple single-task trained LoRA modules in multi-task settings. Another line of work trains LoRAMoE from scratch on mixed-task datasets (Zhu et al., 2023; Dou et al., 2024; Ning et al., 2024; **?**). However, the expert design paradigms in the aforementioned works rely on explicit partitioning of LoRA parameters, leading to a trade-off between the granularity of control and the difficulty of training the routing module. In our work, we fully exploit the advantages of pre-trained LoRA modules and finer-grained parameter control.

6 Conclusion

In this paper, we introduce LoRACoE, an efficient fine-tuning method for Mixture of Experts (MoE). We begin by analyzing the parameter importance patterns within LoRA modules, identifying task-relevant parameters that concentrate along the rank dimension. Building on this insight, we redesign the expert construction and propose LoRACoE, a method that dynamically builds experts through rank-level parameter composition. Experiments demonstrate that LoRACoE achieves significantly better performance compared to conventional LoRA and LoRAMoE methods, without a notable increase in computational resources. These results highlight the effectiveness of our approach and offer a new, dynamic, and flexible framework for constructing MoE models.

Limitations

In this paper, we achieve improved performance over LoRA and partition-based LoRAMoE by employing a finer-grained model partitioning and a composition-based expert design. However, our approach has two notable limitations. First, the introduction of a composition-based routing module increases the number of trainable parameters due to the larger output dimension compared to traditional LoRAMoE methods. This increase in parameters has led to diminished returns from adding more experts in our experiments, highlighting a need for more parameter-efficient routing methods that maintain performance. Second, our LoRA module initialization relies on a training-based approach, which requires much computational resources. Exploring more effective, training-free initialization methods for the LoRA modules could further improve the usability of LoRACoE.

References

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

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*.

Guanjie Chen, Xinyu Zhao, Tianlong Chen, and Yu Cheng. 2024a. Moe-rbench: Towards building reliable language models with sparse mixture-of-experts. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Lihu Chen, Adam Dejl, and Francesca Toni. 2024b. Analyzing key neurons in large language models. *arXiv preprint arXiv:2406.10868*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

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

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. 2024. Self-play with execution feedback: Improving instruction-following capabilities of large language models. *CoRR*, abs/2406.13542.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, et al. 2024. Loramoe: Alleviating world knowledge forgetting in large language models via moe-style plugin. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1932–1945.

Dheeru Dua, Shruti Bhosale, Vedanuj Goswami, James Cross, Mike Lewis, and Angela Fan. 2021. Tricks for training sparse translation models. *arXiv preprint arXiv:2110.08246*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. 2024a. Mixture-of-loras: An efficient multitask tuning for large language models. *arXiv preprint arXiv:2403.03432*.

Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. 2024b. Mixture-of-loras: An efficient multitask tuning method for large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 11371–11380. ELRA and ICCL.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. The impact of initialization on lora finetuning dynamics. *CoRR*, abs/2406.08447.

Xu Owen He. 2024. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*.

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.

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*.

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*.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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.

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

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhiyuan Cheng, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. 2024. Mixlora: Enhancing large language models fine-tuning with lora based mixture of experts. *CoRR*, abs/2404.15159.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *arXiv preprint arXiv:2402.14830*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

Lin Ning, Harsh Lara, Meiqi Guo, and Abhinav Rastogi. 2024. Mode: Effective multi-task parameter efficient fine-tuning with a mixture of dyadic experts. *arXiv preprint arXiv:2408.01505*.

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

Maciej Pióro, Kamil Ciebiera, Krystian Król, Jan Ludziejewski, and Sebastian Jaszczur. 2024. Moe-mamba: Efficient selective state space models with mixture of experts. *CoRR*, abs/2401.04081.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Jiahuan Pei. 2024. Melora: Mini-ensemble low-rank adapters for parameter-efficient fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3052–3064. Association for Computational Linguistics.

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

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

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

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*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-Zhong Xu. 2024. Hydralora: An asymmetric lora architecture for efficient fine-tuning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Tianwen Wei, Bo Zhu, Liang Zhao, Cheng Cheng, Biye Li, Weiwei Lü, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Liang Zeng, et al. 2024. Skywork-moe: A deep dive into training techniques for mixture-of-experts language models. *arXiv preprint arXiv:2406.06563*.

Xun Wu, Shaohan Huang, and Furu Wei. 2024. Mixture of lora experts. *arXiv preprint arXiv:2404.13628*.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. 2024. Qa-lora: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Jiazuo Yu, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. 2024. Boosting continual learning of vision-language models via mixture-of-experts adapters. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 23219–23230. IEEE.

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

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *International conference on machine learning*, pages 26809–26823. PMLR.

Zhihao Zhang, Jun Zhao, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. Unveiling linguistic regions in large language models. *arXiv preprint arXiv:2402.14700*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *CoRR*, abs/2311.07911.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. 2024. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *arXiv preprint arXiv:2406.16554*.

Yun Zhu, Nevan Wichers, Chu-Cheng Lin, Xinyi Wang, Tianlong Chen, Lei Shu, Han Lu, Canoee Liu, Liangchen Luo, Jindong Chen, et al. 2023. Sira: Sparse mixture of low rank adaptation. *arXiv preprint arXiv:2311.09179*.

A More Implementation Details

We conducted our experiments on eight A100 GPUs. In our setup, the rank for LoRA, LoRAMoE, and LoRACoE is set to 16, with α set to 32. The number of experts for LoRAMoE is set to 4, and for LoRACoE, it is set to 2. For the LLaMA series models, we use a batch size of 16, and for the Qwen2 series models, we use a batch size of 32. For LoRA and LoRAMoE training experiments, we set the learning rate to 3e-4 and trained for 4 epochs. For LoRACoE training, we apply a staged learning rate schedule: during the LoRA warmup, the learning rate is set to 2e-4 for 2 epochs; for the joint training stage, we set the learning rate to 5e-5 over 2 epochs. For full fine-tuning, we set a learning rate of 1e-5 for llama2-7b, llama2-13b, llama3-8b, and Qwen2-7b. For Qwen2-1.5b and Qwen2-0.5b, we set the learning rate to 5e-5. All full-parameter fine-tuning is conducted over 2 epochs.

B Ablation on expert number.

As shown in Table 2, we evaluate the impact of varying the number of experts on model performance and we also removed the routing module during inference to test the role of the routing module in the inference process. We find that the performance on commonsense and mathematical reasoning tasks does not increase significantly with more experts. Considering that increasing the number of experts further leads to an increase in the parameters of the routing module, we select 2 as the expert number in main experiments.When the routing mechanism is used during training but removed during inference, the model also experiences a decrease in performance. However, this decline is smaller compared to removing the routing mechanism during training(original LoRA method).This phenomenon can be explained by the observations in Figure 44. Specifically, after introducing the routing mechanism during training, our method enables better alignment between the parameters in the LoRA module and the tasks, compared to the original LoRA method. This helps mitigate potential conflicts in parameter updates during multi-task training.

C Computation optimization for LoRACoE

First, to optimize the computational efficiency of LoRACoE, we conducted a theoretical analysis of the computational consumption of the LoRACoE method.

Method	Memory util(GB)	Relative ratio
Full Param Finetune	23.55	100%
LoRA	6.07	25.80%
LoRAMoE and Averaging	6.12	25.99%
LoRACoE	6.28	26.67%

Table 4: Memory usage of different methods based on Qwen2-1.5B after model and optimizer initialization.

C.1 Parameter Analysis

For the original LoRA method, its parameters consist of `LoRA_A` and `LoRA_B`, with a total parameter count of $r(d+k)$. For the LoRACoE method, the parameter count of the LoRA part is $r(d+k)$. For the Router module, with E composite experts, each expert consists of a $d \times r$ matrix. Hence, the parameter count for the Router module is $r \cdot d \cdot E$. Therefore, the total parameter count in LoRACoE is $r(d+k+dE)$.

C.2 Computation Cost Analysis

For computation costs, under the same rank r, the computation cost of the LoRA module in LoRACoE is identical to that in the original LoRA method. The additional computation cost arises in the Router module, which mainly includes two parts:

C.2.1 Vector-Matrix Multiplication

The input vector multiplies with E $d \times r$ matrices to produce a tensor of shape (r, E). This step is equivalent to performing vector-matrix multiplication with E `LoRA_A` modules.In the main experimental setting, $E = 2$, $r = 16$, and considering $r \ll d$, the first step introduces computational costs comparable to the LoRA module itself. In our optimized implementation, the E $d \times r$ matrices in the Router and the $d \times r$ `LoRA_A` module are concatenated into a $d \times r(E+1)$ two dimension matrix for matrix multiplication.The output is then split in-place to obtain the outputs of `LoRA_A` module and the Router module. This approach fully utilizes GPU vector-matrix multiplication units (similar to the optimization used in vLLM for Attention).This implementation avoids introducing extra high-dimensional tensor operations or iterative computations, effectively controlling additional computational resource usage.

Method	Time(ms)	Proportion
LoRA A	62.79	16.56%
LoRA B	46.96	17.19%
Outer product	49.60	19.52%
Router	39.84	20.61%
Softmax and Averaging	41.35	26.10%
Total time	240.57	100%

Table 5: Profiling for LoRACoE without computation optimization.

C.2.2 Softmax and Averaging

The output tensor of shape (r, E) undergoes a softmax operation along the 0th dimension (dimension r) without reduction, followed by an averaging operation along the 1st dimension (dimension E). Since $r \ll d$ and $E \ll d$ in both the LoRA matrix and experimental settings, this step introduces only minimal computational cost.

C.3 Performance Profiling

Based on the theoretical analysis above, we profile the forward computation time of different modules in LoRACoE, and the results obtained are shown in the Table 5. It can be observed from the above figure that the computations of the outer product and Router modules account for 40% of the total computation time. The computational optimizations mentioned earlier eliminate high-dimensional outer product operations while effectively reducing forward computation time by merging the Router module with `LoRA_A` computations. The optimized profiling results are shown at Table 6 below. After optimization, the forward computation time of the LoRACoE module decreased from 240.57 ms to 183.19 ms, representing a reduction of 23.8% in computation time. We conducted further performance comparisons during the training phase under the same training setting, and the results

Method	Time(ms)	Proportion
LoRA A + Router	98.38	53.70%
LoRA B	44.99	24.56%
Softmax and Averaging	39.80	21.73%
Total time	183.19	100%

Table 6: Profiling for LoRACoE with computation optimization.

are shown in the Table 7. According to the re-

Method	Training Epoch	Total Training Time
LoRA	4	10,336.59 s
LoRAMoE	4	11,888.02 s
LoRACoE	4	10,828.51 s
LoRACoE w/o optim	4	32,398.48 s

Table 7: Training Time Consumption

sults, after optimization, our algorithm achieves training times comparable to those of the baseline LoRA method despite increase in the number of parameters. Overall, by integrating computational optimizations with the LoRACoE algorithm, LoRACoE not only achieves superior performance but also becomes more cost-effective for deployment in real-world scenarios.

D Analysis of Additional Parameters and Performance

As shown in the table, under the setting where the LoRA rank is r, input dimension is d, output dimension is k, and the number of experts is E, using the same LoRA rank, our method achieves better performance compared to the LoRA and partition-based LoRAMoE methods. However, the higher parameter complexity of our method leads to an inconsistency in the proportion of trainable parameters.

To address this, we reduce the rank number to maintain a consistent ratio of trainable parameters, and under this adjustment, our method still demonstrates better performance.

E Additional Experiments on HydraLoRA

To provide a more comprehensive comparison of LoRACoE's performance, we conducted experiments on the Qwen2 series models using HydraLoRA with the same settings as the main experiments, setting HydraLoRA's k to 3. The results are shown in the table, where it can be observed that LoRACoE generally exhibits better performance.

Model	Method	Lora Rank(r)	Expert Number(E)	Parameter Complexity	Trainable parameter ratio*	Math avg	CS avg
llama2-7b	LoRA	16	0	r(d+k)	0.467%	75.42	63.26
	LoRAMoE	16	4	r(d+k+d/E)	0.583%	82.94	74.01
	LoRACoE	8	2	r(d+k+dE)	0.467%	85.97	79.16
	LoRACoE	16	2	r(d+k+dE)	0.933%	86.49	80.83

Table 8: Model Performance Comparison.*Trainable parameter ratio refers to the proportion of newly added trainable parameters relative to the pre-trained parameters.

Model	Method	Commonsense	Math	Insturction Following	Avg.
Qwen2-0.5b	HydraLoRA	**64.72**	73.11	34.96	57.60
	LoRACoE	64.24	**74.40**	**35.97**	**58.20**
Qwen2-1.5b	HydraLoRA	**75.13**	86.04	47.36	69.51
	LoRACoE	74.91	**86.18**	**48.68**	**69.92**
Qwen2-7b	HydraLoRA	85.18	94.54	64.12	81.28
	LoRACoE	**85.97**	**94.71**	**64.26**	**81.65**

Table 9: Evaluating results of LoRACoE and HydraLoRA on commonsense, math and Instruction following tasks. The best results are in **bold**.