MoKA: Parameter Efficiency Fine-Tuning via Mixture of Kronecker Product Adaption

Beiming Yu* **and Zhenfei Yang*** **and Xiushuang Yi†**
Northeastern University, Shenyang, China
{beiming403, zhenfyang}@163.com, yixiushuang@cse.neu.edu.cn

Abstract

With the rapid development of large language models (LLMs), traditional full-parameter fine-tuning methods have become increasingly expensive in terms of computational resources and time costs. For this reason, parameter efficient fine-tuning (PEFT) methods have emerged. Among them, Low-Rank Adaptation (LoRA) is one of the current popular PEFT methods, which is widely used in large language models. However, the low-rank update mechanism of LoRA somewhat limits its ability to approximate full-parameter fine-tuning during the training process. In this paper, we propose a novel PEFT framework, MoKA (Mixture of Kronecker Product Adaptation), which combines the Kronecker product with the Mixture-of-Experts (MoE) method. By replacing the low-rank decomposition of the weight update matrix with Kronecker products and utilizing a sparse MoE architecture, MoKA achieves parameter efficiency and better model performance. Additionally, we design an efficient routing module to further compress the parameter size. We conduct extensive experiments on the GLUE benchmark, E2E NLG Challenge, and instruction tuning tasks for LLMs. The results demonstrate that MoKA outperforms existing PEFT methods.

1 Introduction

In recent years, the rapid development of large language models (LLM), such as GPT-3 (Brown, 2020) and LLaMA (Touvron et al., 2023), has significantly enhanced the ability to handle a variety of downstream tasks through self-supervised pretraining on extensive, unlabeled text data. However, these models typically have over a billion parameters, making traditional fine-tuning methods prohibitively expensive in terms of computational resources and time. Considering these challenges,

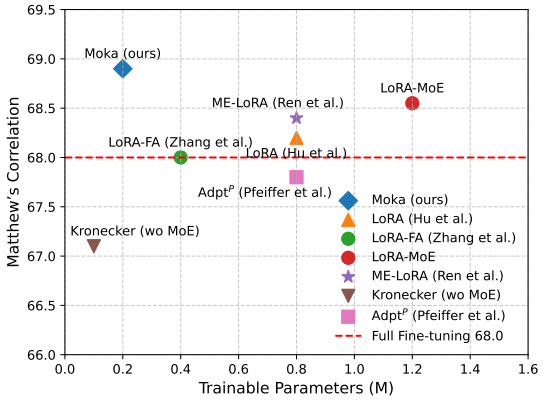

Figure 1: Comparison of the PEFT methods on GLUE Benchmark. MoKA achieves higher scores with a smaller number of trainable parameters.

parameter-efficient fine-tuning (PEFT) methods, which involve fine-tuning only a relatively small number of parameters, have emerged as a potential solution to offset the huge computational and storage costs associated with full-parameter fine-tuning.

Among the current PEFT methods, Adapter-based methods (Houlsby et al., 2019; Wang et al., 2022) perform fine-tuning by inserting small, trainable linear modules into the existing network architecture. Prompt-based methods (Lester et al., 2021; Razdaibiedina et al., 2023) add trainable soft tokens as prompts at the beginning of the input, and these prompts are fine-tuned during training. Low-Rank Adaptation (LoRA) (Hu et al., 2021) employs low-rank decomposition to represent weight updates with smaller matrices, allowing the model to adapt to new data while keeping the core weight matrices unchanged. Among these methods, LoRA has been the most widely used due to its strong adaptability across various downstream tasks and ease of implementation. However, as model sizes grow and tasks become more complex, a performance gap remains between parameter-efficient fine-tuning and full-parameter fine-tuning.

*Equal Contributions.
†Corresponding author.

Proceedings of the 31st International Conference on Computational Linguistics, pages 10172–10182
January 19–24, 2025.

Recently, combining PEFT with Mixture-of-Experts (MoE) has emerged as a promising strategy that further pushes the performance upper bound of PEFT. Such work currently focuses on applying MoE to LoRA by learning multiple low-rank matrices (called LoRA experts) and routing the required expert weights for each input. Dou et al. (2023) explore a novel approach that integrates LoRA and MoE to mitigate LLMs' tendency to forget world knowledge, while also ensuring alignment with downstream tasks. Wu et al. (2024) combines multiple LoRA modules in a MoE framework to improve task performance through hierarchical control and branch selection. Although such approaches show promising results, two major limitations remain: (i) The low-rank approximation of LoRA remains, which may lead to a significant performance gap with full parameter fine-tuning. (ii) The efficiency of routing parameters in MoE-based PEFT methods has not been explored.

Kronecker product does not rely on low-rank structures and has recently been tried to improve PEFT technology. He et al. (2022) attempted to apply the Kronecker product decomposition with PEFT methods and achieved superior parameter efficiency and performance in the image domain. Inspired by this, we propose a novel PEFT framework, called MoKA, which combines both parameter efficiency and high-rank approximation by combining the Kronecker product decomposition and the mixture of experts (MoE) techniques. Unlike LoRA, the MoKA method approximates the weight update matrix using the Kronecker product of two trainable mini-matrices. To further improve the model capacity and parameter efficiency, we replace the single Kronecker decomposition with a sparse MoE architecture that uses multiple matrices as expert modules for better data adaptability through conditional computation.

In the parameter-efficient fine-tuning framework incorporating MoE structure, the parameter efficiency is mainly affected by the joint influence of the expert module and the router module. We significantly improve the parameter efficiency of the expert module by introducing the Kronecker product decomposition approximation for gradient updating. However, the relatively large parameter overhead of the router module becomes more and more noticeable in the overall model. To address this issue, we propose an efficient router module that further compresses the parameter scale of MoKA. Figure 1 illustrates the parameter efficiency of MoKA compared to the most recent PEFT methods.

Overall, our main contributions are as follows:

1. We propose a novel PEFT method, MoKA, which combines the MoE mechanism with the Kronecker product decomposition in an efficient way, and also design a parameter-efficient router that further compresses the model parameters.

2. We conduct extensive experiments on natural language understanding, natural language generation, and instruction tuning tasks. The results show that our PEFT methods achieves significant parameter efficiency along with better performance.

2 Related Work

2.1 Parameter-efficient fine-tuning (PEFT)

Parameter-efficient fine-tuning (PEFT), which selectively tunes a limited number of parameters or introduces additional trainable parameters, has received increasing attention from researchers. To approximate the updated weights (ΔW) during fine-tuning, Hu et al. (2021) introduces a Low-Rank Adaptation method (LoRA) that utilizes trainable low-rank matrices to perform approximate weight updates, which is widely used in many fields. Kim et al. (2024) combines parallel and sequential branch to improve the PEFT performence. Ren et al. (2024) stacks multiple mini LoRAs in parallel, maintaining a higher rank and thus providing better performance potential. However, compared to full-parameter fine-tuning, LoRA is constrained by low-rank updates and exhibits lower accuracy. The Kronecker product has recently been used to improve the PEFT technique and is often used to parametrically reconstruct the weight matrix. For example, He et al. (2022) attempts to apply decomposition with the Kronecker product to parametrically efficient fine-tuning methods in the image domain, achieving superior parameter efficiency and performance. Braga et al. (2024) proposes an adapter-based fine-tuning using Kronecker products.

2.2 Mixture-of-Experts

Mixture-of-Experts (MoE) is often viewed as a system consisting of multiple sub-modules (i.e., "experts") and a router. These sub-modules are selectively activated through a routing mechanism that depends on input features to integrate the outputs of various expert models. This technique has demonstrated strong performance in other domains

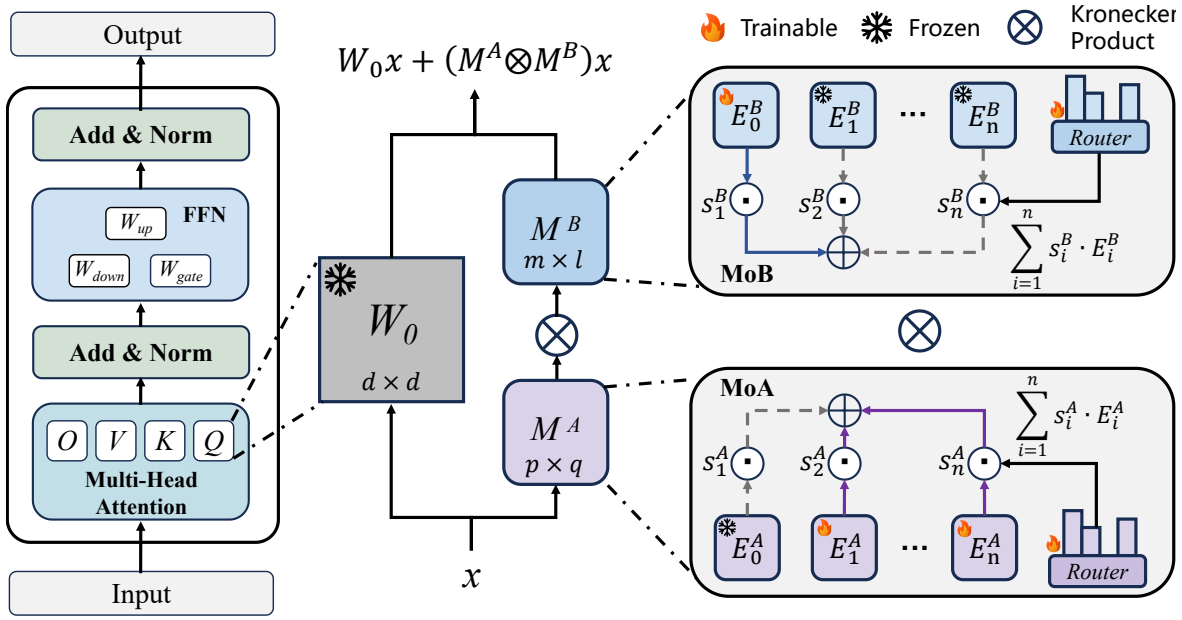

Figure 2: The overall framework of our MoKA method. MoKA is applied to each transformer block of the pre-trained model. It utilises kronecker product factors as dynamically selected experts and a router to determine the activation of the expert sets them at each transformer block. During the training process, only the experts and the router are optimised.

(Shazeer et al., 2017). However, in Parameter-Efficient Fine-Tuning (PEFT) approaches, model weights are typically applied to every input, which can lead to suboptimal parameter efficiency since a given input may not necessarily require the full capacity of the model. Recently, the application of MoE in the PEFT field has garnered significant attention from researchers. Advances in research have combined the LoRA framework with MoE to further improve model adaptability and performance. Dou et al. (2023) integrates LoRA modules through a network of routers to mitigate catastrophic forgetting. Luo et al. (2024) utilizes the MoE architecture to enable the dynamic combination of multiple LoRA modules to better catering to the requirements of downstream tasks. While these studies contribute valuable insights into the fusion of MoE and LoRA, current exploration of MoE in PEFT is still in its early stages: first, the performance of LoRA-MoE is constrained by inherent low-rank approximations, and on the other hand, while the parameter scale of MoE routers in fully fine-tuned large language models (LLMs) is negligible, it becomes non-negligible in the PEFT context. To address this, we propose a novel architecture, MoKA, which decomposes the weight matrix using Kronecker product and incorporates the concept of Mixture-of-Experts.

3 Method

In this section, we elaborate on the implementation of the MoKA method. MoKA combines the Mixture of Experts (MoE) with the Kronecker Product enhancing the model's rank and expressive capacity while maintaining a small trainable parameter size. Moreover, an efficient routing mechanism further reduces the parameter scale of the MoKA architecture. Figure 2 illustrates the overall architectural design of the MoKA method. Next, we will explore the core mechanisms of the approach from three key aspects: Kronecker product, Mixture-of-Experts, and Efficient Router.

3.1 Kronecker product

The Kronecker product is an operation on two input matrices $A \in \mathbb{R}^{p \times q}$ and $B \in \mathbb{R}^{m \times l}$, resulting in a block matrix $W \in \mathbb{R}^{pm \times ql}$. Its mathematical form is as follows:

$$W = A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1q}B \\ \vdots & \ddots & \vdots \\ a_{p1}B & \cdots & a_{pq}B \end{pmatrix} \quad (1)$$

where $\otimes$ represents the Kronecker product.

During model fine-tuning, the weight updates can be reparameterized as follows:

$$h = W_0x + \Delta Wx \quad (2)$$

where $x \in \mathbb{R}^{d_{in}}$ denotes the input activation feature, $h \in \mathbb{R}^{d_{out}}$ denotes the output feature, $W_0 \in \mathbb{R}^{d_{in} \times d_{out}}$ denotes the model parameter weight, $\Delta W \in \mathbb{R}^{d_{in} \times d_{out}}$ denotes the the weight adjustment matrix.

Unlike previous work (Hu et al., 2021; Luo et al., 2024), we use the Kronecker product decomposition for the reparameterization of ΔW to overcome the limitations of the low-rank decomposition. This process is formalized as follows:

$$\begin{aligned} x_r &= reshape(x) \\ h &= W_0 x + (A \otimes B)x \\ &= W_0 x + flatten\left(B x_r A^\top\right) \end{aligned} \tag{3}$$

where $d_{in} = p \times m$ and $d_{out} = q \times l$, the $reshape$ operation transforms x into an $\mathbb{R}^{p \times m}$ matrix, which is then converted back to a d_{out} dimensional vector by the flatten operation following the Kronecker product operation.

Unlike the low-rank decomposition employed in LoRA, the Kronecker product decomposition maintains the full rank of ΔW, enabling it to capture the complete information of a full-rank matrix without the limitations of low-rank approximations. However, a single Kronecker module's reliance on shared parameters to adapt to diverse input features and data can restrict both parameter efficiency and model capacity. This trade-off may lead to less than desirable performance when handling complex tasks requiring more specialized adaptation.

3.2 Mixture-of-Experts

Mixture-of-Experts (MoE) is a neural network that enables conditional computation by activating different experts through a gating mechanism. Based on this, we propose the MoKA method, which uses a MoE-based approach to re-model ΔW decomposition factors A and B. Specifically, we use a soft merging modeling approach (Zadouri et al., 2023) and utilize a lightweight sparse activation MoE network to model the two decomposition factors A and B as M^A and M^B, respectively.

Specifically, we apply a soft merging computation (Zadouri et al., 2023) and utilize a lightweight sparsely activated MoE network to represent these factors as M^A and M^B, respectively. This design allows for more flexible and efficient modeling of ΔW through the dynamic activation of specialized experts. The M^A is generated by the MoA module, which comprises a gating network G^A and a set of n experts $E_1^A, ..., E_n^A$. Each expert $E_i^A \in \mathbb{R}^{p \times q}$ represents a linear transformation matrix. Similarly, M^B is produced by the MoB module, consisting of a gating network G^B and a set of n experts $E_1^B, ..., E_n^B$, where expert $E_i^B \in \mathbb{R}^{m \times l}$.

First, we derive the router weights for both MoA and MoB using a combination of routing transformation and the softmax operation. We then apply the $TopK$ function to achieve sparse activation, reducing computational cost and improving efficiency. The formulas are given as follows:

$$S^A = \text{Softmax}(\text{TopK}(G^A x), k) \tag{4}$$

$$S^B = \text{Softmax}(\text{TopK}(G^B x), k) \tag{5}$$

where $S^A = \{s_1^A, s_2^A, ..., s_n^A\}$ and $S^B = \{s_1^B, s_2^B, ..., s_n^B\}$ denote the routing weights obtained after sparse activation by $TopK$ operation. Here, $\text{TopK}(G^A(x))_i := G^A(x)_i$ if $G^A(x)_i$ is among the top-k coordinates of logits $G^A(x) \in \mathbb{R}^n$, and $\text{TopK}(G^A(x))_i := -\infty$ otherwise. The hyperparameter k, which represents the number of experts selected per token, adjusts the amount of computational parameter required to process each token.

Then M^A and M^B are obtained by routing through soft merging.

$$M^A = \sum_{i=1}^{n} s_i^A \cdot E_i^A \tag{6}$$

$$M^B = \sum_{i=1}^{n} s_i^B \cdot E_i^B \tag{7}$$

The MoKA method improves flexibility and parameter efficiency by dynamically routing computational resources to relevant experts, minimizing redundant computations. Finally, the whole reparameterization process is modeled as:

$$h = W_0 x + (M^A \otimes M^B)x \tag{8}$$

3.3 Efficient Router

Efficient expert modeling in MoKA significantly reduces GPU memory footprint and improves parameter efficiency. However, compared to the extremely parameter-efficient expert module, the number of parameters in the router module instead becomes a performance bottleneck for fine-tuning. In the MoKA framework, we focus on fine-tuning the model weights W_Q as an illustrative example, where $d_{in} = d_{out} = d_q$. For the MoA module, we assume there are n experts, with each expert

E_i^A having dimensions $\sqrt{d_q} \times \sqrt{d_q}$. Consequently, each expert's parameter count is d_q. The routing module has a parameter count of $n \times d_q$, while the experts collectively contribute $n \times d_q$ parameters. This configuration results in the routing module constituting approximately 50% of the total parameter count in the MoKA method. Therefore, there is an urgent need to conduct in-depth research on routing mechanisms with higher parameter efficiency.

The parameter efficient router method works by compressing the features of the input x_r in two dimensions, and then using the downscaled features x_a and x_b for the calculation of the routing weights:

$$x_a = f_{compress}(x_r, -1) \tag{9}$$
$$x_b = f_{compress}(x_r, -2) \tag{10}$$
$$S^A = \text{Softmax}(\text{TopK}(G_e^A x_a), k) \tag{11}$$
$$S^B = \text{Softmax}(\text{TopK}(G_e^B x_b), k) \tag{12}$$

where $compress$ is one of the set $\{\text{mean, max, weighted}\}$, the weighted represents a weighted sum of the dimensions to be compressed. The $G_e^A \in \mathbb{R}^{p \times n}$ and $G_e^B \in \mathbb{R}^{m \times n}$.

3.4 Parameter Efficiency Comparison

We compared the parameter efficiency of MoKA (including both the total number of parameters and the number of trainable parameters) with LoRA and LoRA-MoE, as shown in the Table 1.

Let's consider a transformer model with L fine-tuned layers, each consisting of number of weight matrices $W \in d \times d$. For simplicity and fairness of comparison, we set the hyperparameters appropriately for the different methods. For LoRA, we set the rank to r. For LoRA-MoE, we set the maximum number of experts to n and the number of activated experts to k. For MoKA, we set the size of the Kronecker factor A to be $p \times q$ and factor B to be $m \times l$. To obtain a higher efficiency of the parameters, we set $p = q = m = l = \sqrt{d}$. For MoE settings, we keep the number of experts consistent with LoRA-MoE.

For the approximation part of ΔW, it can be seen that the number of parameters required by LoRA-MoE grows linearly with r. Compared to LoRA-MoE, MoKA has a naturally high rank property because of the Kronecker product. For the routing part, we compress the router parameters by the efficient router method, which reduces the parameter overhead from the original nd to $n\sqrt{d}$. To Summary, the number of LoRA-MoE's parameters is significantly larger than that of MoKA, which demonstrates the advantage of MoKA in terms of parameter efficiency.

Method	Total Param	Trainable Param
LoRA	2rd	2rd
LoRA-MoE	2nrd + nd	2krd + nd
MoKA (wo ER)	2nd + 2nd	2kd + 2nd
MoKA	2nd + $2n\sqrt{d}$	2kd + $2n\sqrt{d}$

Table 1: Parameter Efficiency Comparison of MoKA, LoRA and LoRA-MoE, where ER represent Efficient Router Method

4 Experiment

In this section, we conduct a comprehensive set of experiments to evaluate our fine-tuning method across a variety of downstream language tasks. Moreover, we conduct further experiments to analyze the impacts of various details within this study.

4.1 Experiment setup

4.1.1 Evaluation Tasks

To fully demonstrate the effectiveness of our approach, we carry out extensive experiments on natural language understanding (NLU), natural language generation (NLG), and large language model instruction fine-tuning. The tasks we used to evaluate the performance are as follows:

GLUE benchmark: The GLUE benchmark is for NLU tasks and includes classification tasks, similarity and paraphrase tasks, and natural language inference tasks. The GLUE benchmark consists of eight types of natural language understanding tasks, Linguistic Acceptability (CoLA (Warstadt, 2019)), Sentiment Analysis (SST-2 (Socher et al., 2013)), Similarity and Paraphrase tasks ((MRPC (Dolan and Brockett, 2005)), STS-B (Cer et al., 2017), QQP (Wang et al., 2018)), and Natural Language Inference (MNLI (Williams et al., 2017), QNLI (Rajpurkar, 2016), RTE (Dolan and Brockett, 2005). We adopt the same evaluation metrics as used in Hu et al. (2021) to ensure consistency in comparison.

E2E NLG Challenge: The E2E NLG Challenge dataset is a widely recognized benchmark for evaluating Natural Language Generation (NLG) models. It comprises 51,200 samples distributed as follows: 42,200 for training, 4,600 for validation, and 4,600 for testing. This dataset is frequently utilized to

measure the effectiveness of NLG models and provides a robust framework for evaluating various NLG tasks. We evaluate model performance using the BLEU (Papineni et al., 2002), NIST (Doddington, 2002), Meteor (Banerjee and Lavie, 2005), ROUGE-L (Lin, 2004), and CIDEr (Vedantam et al., 2015) metrics in the E2E NLG Challenge dataset.

Instruction tuning: Instruction tuning is the fine-tuning of a model through a set of instructions or prompts used to improve the performance of a language model so that it can better understand and follow specific instructions. For instruction tuning experiments, we use the LLaMA2-7B models, trained on 100k samples of the MetaMathQA (Yu et al., 2023) dataset and evaluated on the GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) benchmarks.

4.1.2 Implementation Details

We use an NVIDIA A800 GPU to train our model for extensive experiments on natural language understanding and generative tasks. For the GLUE benchmark and E2E challenge datasets, we set the total number of experts for our MoKA method to 16, the number of activated experts to 2, and the shapes of expert E^A and E^B to (32, 32). For the instruction tuning task, we set the total number of experts to 128, the number of activated experts to 16, and the shapes of E^A and E^B to (64,64). Full experimental details can be found in the AppendixA. We performed five runs using different random seeds, recorded the best epoch results for each run, and reported the median of these results.

To fully demonstrate the validity of our work, we conducted a comparison between AdapterP (Pfeiffer et al., 2021), PrefixTuning (Li and Liang, 2021), LoRA (Hu et al., 2021), LoRA-FA (Zhang et al., 2023), MELoRA (Ren et al., 2024), LoRA-MoE and MoKA and reuse their reported numbers whenever possible. It is worth noting that compared to our MoKA method, LoRA-MoE only replaces the Kronecker-Product module with LoRA, while all other settings remain unchanged.

4.2 Main Result

4.2.1 GLUE Benchmark

For the GLUE benchmark dataset, we use RoBERTa-Large (Liu, 2019) as the backbone language model on the GLUE benchmarks. The results of all methods on GLUE are shown in Table 2. We can see that MoKA exhibits very competitive performance, outperforming most mainstream PEFT methods such as LoRA overall. Especially for datasets such as MRPC, RTE and CoLA, MoKA has achieved significant improvement. Compared to LoRA-MoE, our method still has a performance advantage in terms of overall metrics under the condition that the number of trainable parameters is much smaller (Only 10% of LoRA-MoE). Overall, MoKA significantly reduces the number of trainable parameters and achieves enhanced performance on the vast majority of datasets, demonstrating a good balance between model performance and parameter efficiency.

4.2.2 E2E Benchmark

In the E2E NLG Challenge dataset, we use GPT-2 medium model (Radford et al., 2019) as the backbone language model and compared our method MoKA with LoRA and other baselines. The experimental results are shown in Table 3. With only 0.09M model parameters trained, MoKA shows excellent performance, outperforming the baselines significantly in most metrics. Through experiments on the E2E dataset, we demonstrate that MoKA is not only suitable for NLU, but still performs well on NLG tasks.

4.2.3 Instruction tuning

We fine-tuned the LLaMA2-7B model (Touvron et al., 2023) using the MetaMathQA dataset to evaluate its instruction tuning and mathematical reasoning capabilities on the GSM8K and MATH validation sets. Table 4. shows the results of our instruction tuning experiments. The results of applying MoKA to the large language model are competitive with LoRA and full fine-tuning methods. This demonstrates that the MoKA approach is still applicable on large-scale language models and is also capable of handling complex reasoning tasks.

4.3 Further Analysis

4.3.1 Impact of Compression Functions in Router

In order to explore the role of router compression methods in MoKA, including mean, max, and weighted mean. we tested the performance of various compression methods on the GLUE Benchmark. The results are presented in Table 5. It can be found that the use of max leads to a certain degree of performance degradation, indicating that there is significant information loss with this compression method. With parameter compression

Models	Param	MNLI	SST-2	MRPC	CoLA	QQP	RTE	STS-B	QNLI	Avg.
Full FT	355.0M	87.6	96.4	90.9	68.0	**92.2**	86.6	92.4	94.7	88.9
Adpt^{P}†	0.8M	90.5	**96.6**	89.7	67.8	91.7	80.1	91.9	94.8	87.9
LoRA†	0.8M	**90.6**	96.2	90.2	68.2	91.6	85.2	92.3	94.8	88.6
LoRA-FA†	0.4M	90.1	96	90	68	91.1	86.1	92.0	94.4	88.5
MELoRA†	0.8M	90.5	96.3	90.5	68.8	90.9	87.1	91.9	94.5	88.8
LoRA-MoE	2.0M	**90.6**	96.3	90.7	68.5	91.7	87.0	**92.5**	94.8	89.0
Kronecker	0.1M	90.0	95.9	90.3	66.7	90.2	86.4	91.8	94.7	88.3
MoKA(ours)	0.2M	**90.6**	96.3	**91.2**	**69.3**	91.1	**87.4**	92.2	**94.9**	**89.1**

Table 2: RoBERTa-large model performance on GLUE benchmark. We report Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for the remaining tasks. † indicates that the experimental results are from the original paper. Higher value is better for all metrics.

Models	Param.	BLEU	NIST	METEOR	ROUGE-L	CIDEr
Full FT	355.0M	68.2	8.62	46.2	71.0	2.47
LoRA†	0.35M	69.8	8.80	46.7	71.7	2.52
PrefixTuning	0.35M	70.3	**8.85**	46.2	71.7	2.47
LoRA-MoE	0.88M	70.3	8.84	47.2	72.4	2.56
Kronecker	0.04M	70.4	8.72	46.3	71.4	2.48
MoKA(ours)	0.09M	**71.4**	8.76	**47.9**	**73.3**	**2.56**

Table 3: GPT-2 medium model performance on E2E NLG Challenge, † indicates that the experimental results are from the original paper. Higher value is better for all metrics.

Models	Param.	GSM8K	MATH
Full FT	6738M	56.2	**10.4**
LoRA	160M	55.4	9.4
MELoRA	160M	55.1	9.8
Kronecker	2.4M	54.2	7.5
MoKA(ours)	38.3M	**56.5**	10.2

Table 4: LLaMA2-7B model performance on GSM8K and MATH Benchmarks, Higher value is better for all metrics.

using the mean method and the weighted average method, the model capability is almost unaffected. And without the router compression strategy, the parameter size of the router reaches 16.38K, even exceeding our expert module. It also proves that the exploration of router efficiency is of great significance.

Models	Param.	MRPC	CoLA	RTE
w/o compress	16.38K	91.1	**69.4**	87.6
max	0.51K	89.8	67.7	87.1
mean	0.51K	**91.2**	69.3	87.4
weighted	0.54K	90.8	68.5	**87.6**

Table 5: Impact of compression functions on selected GLUE tasks.

4.3.2 Impact of the Max Number of Experts

The results in Table 6 demonstrate the effect of the maximum number of experts on the model performance. To maintain the same number of trainable parameters, we keep the number of activated experts in the model constant while varying only the maximum number of experts. We observe that increasing the maximum number of experts significantly improves the accuracy of the model when the number of experts is less than 16. This trend suggests that more experts can expand the potential learning space for the model, allowing it to better adapt to complex data. However, when continuing to increase the number of experts, the model performance tends to stagnate. This may be due to the fact that too many experts cannot be adequately trained within the constraints of a fixed number of activated experts, resulting in a model that cannot be further optimized.

Expert Nums	MRPC	CoLA	RTE
2	90.6	67.9	86.6
4	90.9	68.6	86.8
8	90.8	68.7	87.2
16	**91.2**	**69.3**	87.4
32	91.2	68.9	**87.5**

Table 6: Performance according to the Max Number of Experts on selected GLUE tasks.

4.3.3 Impact of the Activated Number of Experts

In this section, we designed a series of experiments to investigate the impact of the number of activated experts on model performance. Specifically, we fixed the maximum number of experts at 16 and incrementally increased the number of activated experts from 1 to 16 during the experiments, observing the performance changes on the CoLA and RTE datasets.

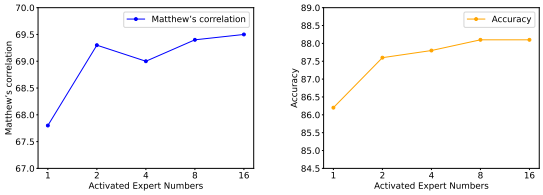

Figure 3: Performance according to Activated Number of Experts, CoLA(left), RTE(right).

The results in Figure 3 indicate that when the number of activated experts is low, the model's performance significantly improves as the number of activated experts increases. Notably, once the number of activated experts reaches a certain threshold, the performance gains begin to diminish and eventually stabilize, while the training costs continue to rise steadily. Therefore, in practical applications, it is essential to select an appropriate number of activated experts based on specific requirements and resource constraints.

4.3.4 Impact of Model Initialization

Table 7 demonstrates the impact of the initialisation strategies for the Kronecker product module and the router module on the performance of the model. The results show that the model achieves the best results when the Kronecker module E^A and router modules R are initialized using the uniform version of Kaiming initialization and the E^B is set to zero. Meanwhile, we observe that the effect of changing the initialisation strategy on the model is relatively weak, proving the stability and robustness of our proposed MoKA.

5 Conclusion

We have introduced MoKA, a novel Parameter-Efficient Fine-Tuning (PEFT) technique that combines Kronecker products with Mixture-of-Experts (MoE) technology, effectively enhancing model performance under conditions of a lower scale of trainable parameters. Additionally, we have optimized the MoE router module and explored methods for compressing routing parameters. Experiments on the GLUE benchmark and E2E challenge demonstrate that MoKA outperforms existing PEFT methods in both parameter efficiency and performance. Furthermore, in instruction fine-tuning tasks, we have shown that MoKA can still provide competitive performance in the instruction fine-tuning of large-scale language models. Overall, MoKA achieves a convincing balance between parameter efficiency and model performance through the use of Kronecker products, MoE, and an efficient routing mechanism. In future work, we plan to extend our method to multi-task and multi-modal domains to more broadly and comprehensively explore the effectiveness of MoKA.

Kronecker Init.	Router Init.	MRPC	CoLA	RTE
$E^A \sim$ Kaiming, $E^B = 0$	$R \sim$ Kaiming	**90.9**	68.9	**88.1**
$E^A \sim$ Kaiming, $E^B = 0$	$R = 0$	90.7	68.3	87.6
$E^A = 0, E^B \sim$ Kaiming	$R \sim$ Kaiming	90.5	**69.1**	87.9
$E^A = 0, E^B \sim$ Kaiming	$R = 0$	90.6	68.5	87.6

Table 7: Impact of different initialization strategies on selected GLUE tasks.

6 Limitations

Although MoKA achieves a balance between parameter efficiency and model performance by combining the Kronecker product and MoE techniques, there are still some shortcomings. First, while MoKA has enhanced the MoE-based parameter-efficient fine-tuning mechanism and increased computational efficiency, it still fails to address the inherent limitations of the MoE mechanism itself. Specifically, the additional parameters introduced during fine-tuning cannot be merged, which increases the parameter processing requirements during the inference phase and consequently elevates inference costs. Furthermore, MoKA incorporates additional hyperparameters, such as the number of experts and top-k selection, which leads to the complexity of model parameter optimization.

Acknowledgments

This work is supported by the National Key R&D Program of China under Grant No. 2021YFC3300300; the National Natural Science Foundation of China under Grant No. 62032013, 62132004.

References

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Marco Braga, Alessandro Raganato, and Gabriella Pasi. 2024. Adakron: An adapter-based parameter efficient model tuning with kronecker product. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 350–357.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

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*.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, et al. 2023. Loramoe: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment. *arXiv preprint arXiv:2312.09979*, 4(7).

Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. 2022. Parameter-efficient fine-tuning for vision transformers. *arXiv preprint arXiv:2203.16329*, 3.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Sanghyeon Kim, Hyunmo Yang, Yunghyun Kim, Youngjoon Hong, and Eunbyung Park. 2024. Hydra: Multi-head low-rank adaptation for parameter efficient fine-tuning. *Neural Networks*, page 106414.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

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

P Rajpurkar. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, Jimmy Ba, and Amjad Almahairi. 2023. Residual prompt tuning: Improving prompt tuning with residual reparameterization. *arXiv preprint arXiv:2305.03937*.

Pengjie Ren, Chengshun Shi, Shiguang Wu, Mengqi Zhang, Zhaochun Ren, Maarten 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), pages 3052–3064.

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*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 1(2):4.

A Warstadt. 2019. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Xun Wu, Shaohan Huang, and Furu Wei. 2024. Mixture of lora experts. In *The Twelfth International Conference on Learning Representations*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermis, Acyr Locatelli, and Sara Hooker. 2023. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*.

A Hyperparameters

Hyper-Parameter	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Optimizer	AdamW							
Warmup Ratio	0.06							
LR Schedule	Linear							
Batch Size	32	32	32	32	32	32	32	32
# Epochs	20	30	30	30	20	20	30	30
Learning Rate	2E-03	2E-03	2E-03	1E-03	1E-03	1E-03	3E-03	4E-03
Weight Decay	0.1							
Max Seq. Len.	128							

Table 8: Hyperparameters and computing resources for natural language understanding experiments on the GLUE benchmark.

Hyper-Parameter	E2E NLG Challenge
Optimizer	AdamW
Warmup Ratio	0.06
LR Schedule	Linear
Batch Size	8
# Epochs	5
Learning Rate	1e-2
Weight Decay	0.1
Beam Num	10
No Repeat Ngram	5
length penalty	1.2
Max Seq. Len.	128

Table 9: The hyperparameters we used for GPT2-medium.

Hyper-Parameter	Instruction tuning
Optimizer	AdamW
Warmup Ratio	0.06
LR Schedule	Linear
Batch Size	64
# Epochs	2
Learning Rate	1e-3
Weight Decay	0.1
Max Seq. Len.	512

Table 10: The hyperparameters we used for LLaMA2-7B.