MELoRA: Mini-Ensemble Low-Rank Adapters
for Parameter-Efficient Fine-Tuning

Pengjie Ren[1*], **Chengshun Shi**[1*], **Shiguang Wu**[1], **Mengqi Zhang**[1], **Zhaochun Ren**[2],
Maarten de Rijke[3], **Zhumin Chen**[1], **Jiahuan Pei**[4†]

[1] Shandong University   [2] Leiden University
[3] University of Amsterdam   [4] Centrum Wiskunde & Informatica
{shichengshun,shiguang.wu}@mail.sdu.edu.cn,
{renpengjie,mengqi.zhang,chenzhumin}@sdu.edu.cn,
z.ren@liacs.leidenuniv.nl, m.derijke@uva.nl, jiahuan.pei@cwi.nl

Abstract

Parameter-efficient fine-tuning (PEFT) is a popular method for tailoring pre-trained large language models (LLMs), especially as the models' scale and the diversity of tasks increase. Low-rank adaptation (LoRA) is based on the idea that the adaptation process is intrinsically low-dimensional, i.e., significant model changes can be represented with relatively few parameters. However, decreasing the rank encounters challenges with generalization errors for specific tasks when compared to full-parameter fine-tuning. We present MELoRA, a mini-ensemble low-rank adapters that uses fewer trainable parameters while maintaining a higher rank, thereby offering improved performance potential. The core idea is to freeze original pretrained weights and train a group of mini LoRAs with only a small number of parameters. This can capture a significant degree of diversity among mini LoRAs, thus promoting better generalization ability. We conduct a theoretical analysis and empirical studies on various NLP tasks. Our experimental results show that, compared to LoRA, MELoRA achieves better performance with 8 times fewer trainable parameters on natural language understanding tasks and 36 times fewer trainable parameters on instruction following tasks, which demonstrates the effectiveness of MELoRA.

1 Introduction

Large language models (LLMs) have emerged as the default paradigm for natural language processing (NLP) (Brown et al., 2020). Fine-tuning (FT) is a prevailing way for tailoring LLMs for specific downstream tasks (Ding et al., 2023b). However, as the models' scale and the diversity of the tasks increase, fully FT becomes infeasible. Parameter-efficient fine-tuning (PEFT) has been proposed to alleviate memory demands by reducing trainable parameters (Rebuffi et al., 2017; Li and Liang,

---

* Equal contribution.
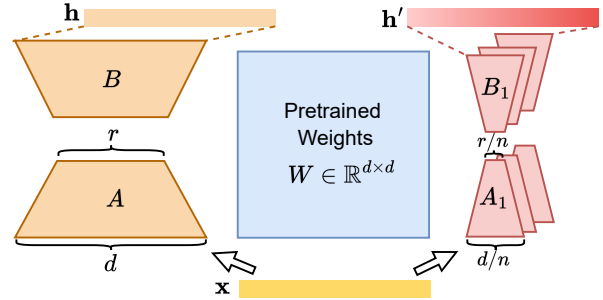† Corresponding author.

Figure 1: Comparison between LoRA (left) and the proposed MELoRA (right). The core idea of MELoRA is to freeze original pretrained weights and train a group of mini LoRAs in parallel with only a small number of parameters.

2021; Lester et al., 2021; Hu et al., 2022). Typically, the core idea of PEFT methods is to update only a small fraction of the parameters, such as adapter weights (Rebuffi et al., 2017; Hu et al., 2022) and prompt weights (Li and Liang, 2021; Lester et al., 2021).

Low-rank adaptation (LoRA) (Hu et al., 2022) is widely being used due to its minimal additional memory overhead and because it comes without additional inference latency. As illustrated in Figure 1 (left), LoRA uses low-rank matrices (A, B) to approximate the updates of pre-trained weights (W). As the rank is smaller than the model's hidden dimension, the overall number of trainable parameters of LoRA is much smaller than full FT. Despite the significant computational advantage, low-rank approximation may lead to a substantial performance gap when compared to full FT (Hu et al., 2022; Zi et al., 2023). Therefore, the following is a critical challenge:

> *How to enable a higher rank variation while preserving the computational advantage?*

To increase the rank of LoRA without introducing more trainable parameters, ReLoRA (Lialin et al., 2023) and COLA (Xia et al., 2024) append multiple

LoRAs to pre-trained weights. They progressively merge old LoRA to pre-train weights and stack new LoRAs during training. In essence, these methods train multiple LoRAs in series. However, there may be overlap between the series of LoRA modules. Therefore, employing the direct sum of multiple LoRA modules in these methods does not necessarily guarantee an increase in rank. In this work, we propose a simple yet effective method, called mini-ensemble low-rank adapters (MELoRA), that stacks multiple mini LoRAs in parallel, as shown in Figure 1 (right). We demonstrate theoretically that MELoRA ensures a higher rank without imposing an additional parameter overhead. We concatenate multiple mini LoRAs simultaneously along the diagonal to construct an equivalent block diagonal LoRA matrix. Each mini LoRA is therefore independent of the other and the final rank will be the sum of the rank of each mini LoRA. Each mini LoRA rank just learns the different dimensions of the hidden state. The shape of the trainable weights A and B in mini LoRA will be much thinner.

We conduct extensive experiments across diverse tasks and models to demonstrate the efficacy of MELoRA. Evaluations are performed using RoBERTa-base on natural language understanding tasks and Llama-2-7B on instruction following tasks. Results indicate that MELoRA achieves superior performance while using significantly fewer parameters. For instance, with 36 times fewer trainable parameters of LoRA, MELoRA outperforms LoRA on all instruction following datasets.

We summarize our contributions as follows:

- We propose a new method (MELoRA) on top of LoRA that makes it achieve a higher rank and better performance with fewer parameters.
- We theoretically demonstrate that MELoRA maintains a higher and flexible rank, as well as lower complexity, compared to LoRA.
- Extensive experiments show that MELoRA outperforms LoRA in terms of parameter quantity and performance.

2 Related Work

Full-parameter fine-tuning poses computational challenges with growing model sizes and the proliferation of downstream tasks. In response to these challenges, parameter-efficient fine-tuning (PEFT) , modifying only a small portion of parameters while leaving the majority of pre-trained model parameters unchanged, has received increasing attention

from researchers. Numerous studies (Rebuffi et al., 2017; Houlsby et al., 2019; Pfeiffer et al., 2021; Rücklé et al., 2021; Li and Liang, 2021; Lester et al., 2021; Hu et al., 2022) discuss key factors such as reduced inference overhead, memory efficiency, and storage optimization. In particular, LoRA (Hu et al., 2022) introduces trainable low-rank matrices to approximate weight updates during fine-tuning. Due to its simplicity in implementation without inducing any noticeable latency during inference, it is widely used in many fields.

A number of advanced techniques have been proposed that build on the basic principles of LoRA. These extensions fall into two main categories: adaptive rank and customized update strategies.

2.1 Adaptive Rank

Some studies (Zhang et al., 2022; Lawton et al., 2023) argue that while the popular Low-Rank Adaptation (LoRA) method is effective, it uses a fixed intrinsic rank that may not always be optimal. They highlight the efficacy of employing higher ranks for more important parameters. Notably, AdaLoRA (Zhang et al., 2022) adopts an adaptive approach to singular value pruning, tailoring rank selection based on the magnitude of individual singular values. Consequently, this method involves the use of different ranks across various layers. Similarly, Ding et al. (2023a) use a gate unit to facilitate the pruning of different ranks. In contrast, Zhang et al. (2023a) propose IncreLoRA, an incremental parameter allocation method that adaptively adds trainable parameters during training based on the importance scores of each module.

2.2 Customized Update Strategies

Another way to improve LoRA is to change the parameter update strategy. Some work is devoted to reducing the number of trainable parameters. Kopiczko et al. (2023) introduces a shared frozen random LoRA module applicable to all pre-trained weights and only trains scaling vectors between LoRA B and A matrices to curtail the number of trainable parameters. This approach reduces the trainable parameter count by a factor of 10 compared to conventional LoRA. Another approach by (Zhang et al., 2023b) involve freezing the A matrix within LoRA, effectively halving the count of trainable parameters. But both of the these methods incur a substantial drop in performance.

QLoRA (Dettmers et al., 2023) further leverages 4-bit quantization to effectively and efficiently fine-

tune LLMs. LoRAMoE (Dou et al., 2023) uses multiple LoRAs as adaptable experts and a router to gate them in the feed-forward network layer to address the problem that fine-tuning data can disrupt the world knowledge stored in LLMs.

To perform model inference in different rank settings, drawing inspiration from nested dropout techniques, Valipour et al. (2023) propose a dynamic parameter update strategy, enabling a single training process for multiple rank inferences. Delta-LoRA (Zi et al., 2023) updates not only the low-rank matrices A and B but also propagates the learning to the pre-trained weights W using the delta of the product of two low-rank matrices A and B.

Despite the computational advantages, so far, low rank approximation leads to a substantial performance gap. To address this limitation, ReLoRA (Lialin et al., 2023) and COLA (Xia et al., 2024) append multiple LoRAs to pre-trained weight to increase the rank of LoRA without introducing more trainable parameters. They progressively merge old LoRA layers to pre-train weight and stack new LoRA layers during training. However, there is no theoretical guarantee for the rank lower bound in training.

Unlike previous work, our proposed method concatenates multiple mini LoRAs in parallel along the diagonal to construct a block diagonal LoRA matrix. It ensures that the final rank will be the sum of the ranks of each mini LoRA.

3 Methodology

In this section, we introduce the proposed mini-ensemble low-rank adapters (MELoRA), a novel method that involves concatenating the outputs from several mini LoRA modules, as illustrated in Figure 1.

3.1 Preliminaries on Low-Rank Adapter

LoRA decomposes the weight update ΔW into a low-rank product BA. During training, the pre-trained weights W are frozen and do not receive gradient updates, while A and B contain trainable parameters as shown in Equation 1:

$$h = Wx + \Delta Wx = Wx + BAx, \quad (1)$$

where $W \in \mathbb{R}^{d \times d}$, $A \in \mathbb{R}^{r \times d}$, $B \in \mathbb{R}^{d \times r}$, $x \in \mathbb{R}^d$, and $r \ll d$. At the start of the training stage, A is randomly initialized via Gaussian initialization, and B is initialized to a zero matrix to make ensure the incremental update $BA = 0$ at initialization.

3.2 Matrix Rank Theory

In linear algebra, several useful inequalities govern the rank of matrices:

$$\mathcal{R}(M_1 + M_2) \leq \mathcal{R}(M_1) + \mathcal{R}(M_2), \quad (2)$$

$$\begin{aligned} \max(\mathcal{R}(M_1), \mathcal{R}(M_2)) & \\ \leq \ \mathcal{R}(\text{concat}(M_1, M_2)) & \quad (3) \\ \leq \ \mathcal{R}(M_1) + \mathcal{R}(M_2), & \end{aligned}$$

$$\mathcal{R}(\text{diag}_{i=0}^n M_i) = \sum_{i=1}^n \mathcal{R}(M_i), \quad (4)$$

where $\mathcal{R}(\cdot)$ denotes the operation to get the rank of a matrix. Equation 2 demonstrates that there is no lower bound when matrices undergo simple addition operations. Equation 3 indicates that the rank does not increase by concatenation, as the column vectors may exhibit linear correlations. However, when matrices are concatenated diagonally, as per Equation 4, the final rank becomes the sum of each matrix's rank.

3.3 Mini-Ensemble Low-Rank Adapter

In MELoRA, we employ n mini LoRAs on pre-trained weights, as depicted on the right in Figure 1. For the convenience of comparison with LoRA, we set the rank of each mini LoRA to $\frac{r}{n}$. MELoRA is defined as the concatenation of several mini LoRAs across different hidden dimensions:

$$\begin{aligned} h &= Wx + \Delta Wx \\ &= Wx + (\text{concat}_{i=0}^n B_i A_i x_i) \\ &= Wx + (\text{diag}_{i=0}^n B_i A_i) \, x \\ &= Wx + (\text{diag}_{i=0}^n B_i) (\text{diag}_{i=0}^n A_i) \, x, \end{aligned} \quad (5)$$

where $A_i \in \mathbb{R}^{\frac{r}{n} \times \frac{d}{n}}$, $B_i \in \mathbb{R}^{\frac{d}{n} \times \frac{r}{n}}$, $x, h \in \mathbb{R}^d$, $x_i \in \mathbb{R}^{\frac{d}{n}}$ is the feature split from x, and n represents how many mini LoRA modules we use to concatenate.

As deduced in Equation 5, MELoRA may be regarded as a form of sparse LoRA. Figure 2 illustrates the process of obtaining equivalent B and A matrices by padding zeros to B_i and A_i along the non-diagonal lines. According to Equation 4, the rank of equivalent B, A is the sum of individual ranks B_i and A_i. Because $\frac{r}{n} \ll d$, each B_i and A_i possesses the same rank $\frac{r}{n}$; the resulting equivalent rank is $n \times \frac{r}{n} = r$.

We employ an identical initialization method to that of LoRA, wherein each A_i undergoes random Gaussian initialization and B_i is initialized to zero.

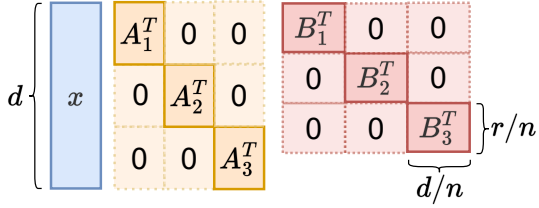Compared to LoRA, MELoRA has the following three advantages:

Figure 2: An illustration of how in MELoRA adopt a group of mini LoRA modules to obtain sparse equivalent B, A. $x \in \mathbb{R}^d$ denote a representation with d dimensions, $A_i \in \mathbb{R}^{\frac{r}{n} \times \frac{d}{n}}$, $B_i \in \mathbb{R}^{\frac{d}{n} \times \frac{r}{n}}$ ($r \ll d$), and 0 denotes zero metrics requiring no training.

(1) MELoRA maintains a higher rank with fewer parameters. The capability of MELoRA to achieve a higher rank with fewer parameters is notable. As discussed in Section 3.2, the simple summation or concatenation of matrices may not inherently increase rank due to potential overlaps between them. In MELoRA, the matrices B_i and A_i are arranged in distinct columns and rows, ensuring that the rank of $\text{diag}_{i=0}^n B_i$ and $\text{diag}_{i=0}^n A_i$ is the sum of individual ranks B_i and A_i. Figure 2 illustrates that the number of trainable parameters in MELoRA is determined by the expression $n \times (\frac{d_{\text{in}}}{n} \times \frac{r}{n} + \frac{r}{n} \times \frac{d_{\text{out}}}{n}) = \frac{d_{\text{out}} \times r + r \times d_{\text{in}}}{n}$. When achieving the same rank, the number of trainable parameters is $d_{\text{out}} \times r + r \times d_{\text{in}}$ (Hu et al., 2022) for LoRA. Importantly, the number of trainable parameters in MELoRA is proportionally reduced by a factor of n compared to LoRA. This suggests the potential for attaining a larger rank while utilizing fewer parameters within the MELoRA framework.

(2) MELoRA has a more flexible rank. The ability to alter the rank without necessitating changes in parameter count is another advantage of MELoRA. Recent studies (Hu et al., 2022; Valipour et al., 2023) emphasize the significance of rank variation across different datasets in influencing model performance. In MELoRA, we might as well set the rank of each mini LoRA to r. So individual mini LoRA modules denoted as A in $\mathbb{R}^{r \times \frac{d}{n}}$ and B in $\mathbb{R}^{\frac{d}{n} \times r}$ are configured, with a total count of trainable parameters being $2 \times r \times d$ and the equivalent rank expressed as $n \times r$. Adjusting the hyperparameter n allows for modulation of the equivalent rank without necessitating an increase in the overall parameter count.

(3) MELoRA has lower complexity. We can compare the complexity of LoRA and MELoRA under equal rank conditions, where $A \in \mathbb{R}^{r \times d}$, $B \in \mathbb{R}^{d \times r}$, $A_i \in \mathbb{R}^{\frac{r}{n} \times \frac{d}{n}}$, $B_i \in \mathbb{R}^{\frac{d}{n} \times \frac{r}{n}}$, and $x \in \mathbb{R}^d$. The time complexity of LoRA is $d^2 r + dr^2$,

while MELoRA is $n((\frac{d}{n})^2 \frac{r}{n} + \frac{d}{n}(\frac{r}{n})^2) = \frac{d^2 r + dr^2}{n^2}$. MELoRA exhibits n^2 times fewer operations.

4 Experimental Setups

4.1 Baselines

We compare MELoRA with LoRA and a number of state-of-the-art LoRA variants:

- **LoRA** (Hu et al., 2022) uses the multiplication of two low-rank matrices to learn the incremental updates with reduced GPU memory cost.
- **DyLoRA** (Valipour et al., 2023) randomly selects a rank r for LoRA modules during learning.
- **AdaLoRA** (Zhang et al., 2022) focuses on determining the optimal rank for incremental updates. It employs an adaptive approach to singular value pruning, tailoring the rank selection to the magnitude of each singular value. Consequently, distinct ranks are employed for different layers.
- **Delta-LoRA** (Zi et al., 2023) not only updates the low-rank matrices A and B but also propagates the learning to the pre-trained weights W via updates using the delta of the product of two low-rank matrices $(A^{(t+1)} B^{(t+1)} - A^{(t)} B^{(t)})$. We follow their setups to reproduce NLU experimental results for a fair comparison.

4.2 Datasets

We evaluate the performance on two groups of datasets: GLUE (Wang et al., 2019) and INSTRUCTEVAL (Chia et al., 2023). The statistics are shown in Table 1. The GLUE benchmark is for NLU tasks (Wang et al., 2019) and includes classification tasks, similarity and paraphrase tasks, and natural language inference tasks:

- **MRPC** (Dolan and Brockett, 2005) is a corpus of sentence pairs automatically extracted from online news sources. The task is to determine whether the sentences in a given pair are semantically equivalent.
- **RTE** comes from a series of annual textual entailment challenges, i.e., RTE1 (Bentivogli et al., 2009), RTE2 (Bar-Haim et al., 2014), and RTE3 (Giampiccolo et al., 2007). The task is to predict whether the premise entails the entailment or not.
- **QQP** is a collection of question pairs from the community question-answering website Quora.[1] The task is to determine whether a pair of questions are semantically equivalent.

---

[1] https://huggingface.co/datasets/glue/viewer/qqp

- **CoLA** (Warstadt et al., 2019) consists of English acceptability judgments drawn from books and journal articles on linguistic theory. The task is to predict whether it is a grammatically correct English sentence.
- **STSB** (Cer et al., 2017) is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data. Each pair is human-annotated with a similarity score from 1 to 5. The task is to predict these scores.
- **SST-2** (Socher et al., 2013) consists of sentences from movie reviews and human annotations of their sentiment. The task is to predict the sentiment of a given sentence.
- **QNLI** (Rajpurkar et al., 2016) is a question-answering dataset consisting of question-paragraph pairs. GLUE converts the task into sentence pair classification by forming a pair between each question and each sentence in the corresponding context. The task is to determine whether the context sentence contains the answer to the question.
- **MNLI** (Williams et al., 2018) is a crowdsourced collection of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral).

We train our model on the cleaned Alpaca dataset[2] and evaluate the performance on INSTRUCTEVAL, an instruction following benchmark (Chia et al., 2023):
- **MMLU** (Hendrycks et al., 2020) is designed to measure world knowledge and problem-solving ability in multiple subjects. It evaluates models in zero-shot and few-shot settings, making it more challenging and closer to how humans are evaluated. The benchmark covers 57 subjects across STEM, humanities, social sciences, and other areas, ranging in difficulty from elementary to advanced professional levels. Each sample has 4 choices. Given the instruction, the task is to predict the correct answer.
- **BBH** (Srivastava et al., 2023) is a subset of 23 challenging tasks from the BIG-Bench benchmark, which focuses on tasks believed to be beyond the capabilities of current language models.

BM	Corpus	#Train	#Dev	#Test
GLUE	MRPC	3.7k	408	1.7k
	RTE	2.5k	276	3k
	CoLA	8.5k	1k	1k
	STS-B	7k	1.5k	1.4k
	SST2	67k	872	1.8k
	QQP	364k	40k	391k
	QNLI	108k	5.7k	5.7k
	MNLI	393k	20k	20k
INSTRUCT EVAL	MMLU	99.8k	1.8k	14k
	BBH	0	0	6.5k
	DROP	77.4k	0	9.5k
	HEval	0	0	164

Table 1: Dataset statistics. "BM" is short for "Benchmark". Only the test sets of INSTRUCTEVAL are used. We train the models on the cleaned Alpaca dataset.

It requires models to follow challenging instructions such as navigation, logical deduction, and fallacy detection.
- **DROP** (Dua et al., 2019) is a math-based reading comprehension task that requires a system to perform discrete reasoning over passages extracted from Wikipedia articles. To perform well on DROP, a system must resolve references in a question to suitable parts of the given passage, and perform discrete operations such as addition, counting, or sorting.
- **HumanEval (HEval)** (Chen et al., 2021) is a problem-solving benchmark used for evaluating large language models trained on code. It consists of 164 original programming problems that assess language comprehension, algorithms, and simple mathematics, with some problems comparable to simple software interview questions.

4.3 Implementation Details

For all experiments, we only fine-tune W_Q and W_V (Zhang et al., 2022). All models and datasets are downloaded from Huggingface.[3] All models are fine-tuned on NVIDIA A800 GPUs. The results are averaged with 5 different random seeds.

On the GLUE benchmark, we use RoBERTa-base with as the backbone LLM. For fair comparison, the training configurations are selected according to Zi et al. (2023). We set the rank of LoRA and its variants to 8. For MELoRA, we performed experiments with two settings. First, we set the rank of each mini LoRA to 8 in MELoRA

Method	#Params	MRPC	RTE	CoLA	STS-B	SST-2	QQP	QNLI	MNLI	Avg.
FT*	125.0M	88.23	84.11	64.57	90.56	94.26	91.96	92.73	87.51	86.74
DyLoRA*	295k	89.46	84.47	61.12	91.06	94.26	90.17	92.22	86.33	86.14
AdaLora*	295k	90.19	85.19	61.64	91.16	94.49	90.14	93.08	87.34	86.65
DeltaLoRA*	295k	90.19	**87.00**	63.82	<u>91.57</u>	<u>95.06</u>	**90.87**	<u>93.09</u>	**87.50**	<u>87.38</u>
LoRA	295k	89.92	85.92	62.43	91.36	94.38	<u>90.78</u>	92.64	86.91	86.79
MELoRA	37k	<u>90.69</u>	86.28	<u>64.07</u>	91.08	94.95	89.26	92.70	86.21	86.91
MELoRA	295k	**90.93**	<u>86.64</u>	**64.09**	**91.93**	**95.41**	90.77	**93.17**	<u>87.20</u>	**87.52**

Table 2: Results on GLUE for natural language understanding tasks. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. We also report the number of trainable parameters (#Params) for each method. * indicates the numbers published in (Zi et al., 2023). We use the same hyper-parameters as Zi et al. (2023). Boldface indicates the best results in terms of the corresponding metrics; the second-best results are <u>underlined</u>.

to get the same number of trainable parameters. Second, to assess performance with fewer trainable parameters, we conduct experiments with a rank of 1 for each mini LoRA. As the number of trainable parameters of MELoRA remains constant regardless of the number of mini LoRAs when the rank of mini LoRA is fixed, we explore the parameter n from the set {2, 4, 8} and report the best performance. And we also analyze the effect of n in Section 6.2.

On the INSTRUCTEVAL benchmark, we use LLaMA-2-7B as the backbone LLM, Alpaca dataset as train set, randomly select 2k samples as the development set. Following INSTRUCTE-VAL (Chia et al., 2023), we use 5-shot direct prompting for MMLU, 3-shot direct prompting for BBH, 3-shot direct prompting for DROP (dev), and 0-shot direct prompting for HEval. During training, we use AdamW as the optimizer and train the models for 3 epochs. For fair comparison, we keep the number of epochs consistent with the baselines. A linear learning rate schedule is applied with initial learning rate 3×10^{-4}. The batch size is set to 128. We explore the rank of LoRA from the set {8, 16, 32, 64, 128, 256} and report the optimal performance. For our method, MELoRA, we set the rank r to 1, explore the number of mini LoRAs n from the set {8, 16, 32, 64}, and report the optimal performance. More implementation details can be found in Appendix A.

5 Results

5.1 Performance on GLUE

The results of all methods on GLUE are shown in Table 2.

We can see that MELoRA outperforms LoRA on 7 out of 8 GLUE datasets under the same parameter setting. Even using 8 times fewer parameters, MELoRA still achieves better performance on 5 out of 8 datasets, underscoring the enhanced expressiveness and higher rank of MELoRA. It is worth noting that large improvements are achieved on MRPC, RTE, CoLA, and SST-2, which have limited training data. We think the reason is that MELoRA concatenates several mini LoRAs, which makes it more robust and has better generalization capability. MELoRA also achieves decent performance on the remaining datasets, including MNLI, QNLI and STS-B, which proves that MELoRA is stable and reliable across different settings.

5.2 Performance on INSTRUCTEVAL

The results of all methods on INSTRUCTEVAL are shown in Table 3.

Method	#Params	MMLU	DROP	HEval	BBH
w/o FT	-	45.96	31.55	12.20	32.04
FT	7B	**47.30**	29.12	12.80	32.72
LoRA	33.6M	45.64	32.46	15.09	32.40
QLoRA	33.6M	45.40	28.97	15.24	32.81
AdaLoRA	33.6M	45.96	31.94	14.02	32.85
MELoRA	0.5M	46.46	**32.65**	**16.16**	**33.01**

Table 3: Results on INSTRUCTEVAL for instruction-following tasks. We report the exact match for MMLU, DROP and BBH, pass@1 for HumanEval. Higher is better for all metrics. The boldface indicates the best results in terms of the corresponding metrics.

We can see that MELoRA consistently outperforms all baselines across all tasks while utilizing more than 36 times fewer trainable parameters. This highlights the effectiveness and efficiency of our proposed approach in instruction following

tasks. As shown in Section 3.3, we believe the reason is that MELoRA can achieve a higher rank with fewer parameters.

6 Analysis

In this section, we analyze two key hyperparameters in MELoRA: the number of mini LoRAs n and the rank of each mini LoRA r. According to Equation 5, the equivalent rank of MELoRA is denoted as $n \times r$. We investigate the effect of the equivalent rank in Section 6.1, and analyze n and r separately in Section 6.2 and 6.3.

6.1 Analysis of Equivalent Rank

In this section, we delve into the effect of the equivalent rank. We conduct experiments across different equivalent ranks, specifically 4, 8, and 16 on GLUE, and 16, 32, and 64 on INSTRUCTEVAL. The results are shown in Table 4 and 5, respectively.

We have two observations from the results. First, MELoRA consistently achieves superior or comparable performance across all equivalent rank settings. In Table 4 and Table 5, MELoRA achieves the best performance on most datasets with more than 2 times fewer trainable parameters. This indicates that equivalent rank is more important than the number of trainable parameters. Ideally, we should search the best equivalent rank settings on different datasets, but setting n to 2 is a good choice in most cases. Second, the optimal equivalent ranks vary across datasets and tasks. On GLUE, the optimal performance is generally achieved with $n = 2$. In contrast, on INSTRUCTEVAL, a higher value of n such as 4 or 8 is a more effective choice. We think that model sizes and task complexity are the main factors. For instance, the RoBERTa model, with only 125M parameters, is considerably smaller than Llama-2-7B. According to scaling laws (Kaplan et al., 2020), Llama-2-7B is more powerful. Consequently, larger models like Llama-2-7B may not necessitate a significant increase in trainable parameters to adapt, suggesting that MELoRA plays a more pivotal role in larger models.

To ascertain whether MELoRA performs a higher rank update than LoRA, we analyze the number of singular values exceeding 0.1 for both LoRA and MELoRA to estimate the real rank. As shown in Figure 3, MELoRA exhibits a significantly higher count of singular values compared to LoRA in the $n \times r$ setting. This observation suggests that MELoRA indeed achieves a high-rank
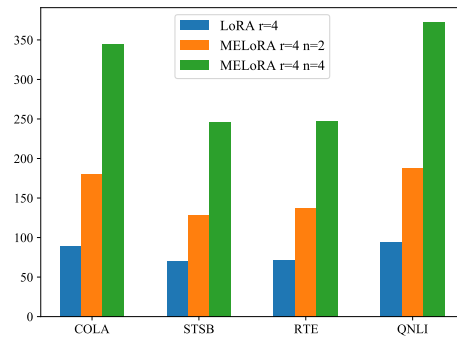
update by performing multiple low-rank updates.

Figure 3: The sum of singular values > 0.1 of $B \times A$ in LoRA and equivalent $B \times A$ in MELoRA.
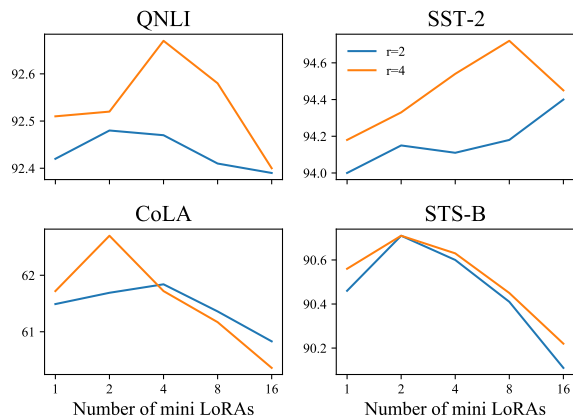

Figure 4: Performance with different number of mini LoRAs n and fixed rank r on different datasets. We report the same metrics as Table 2. More results can be found in Appendix C.

6.2 Analysis of the Number of Mini LoRAs

As discussed in Section 3.3, maintaining a fixed rank for each mini LoRA results in an unaltered parameter count. Consequently, the ability to modify the equivalent rank by adjusting n does not necessitate an increase in the overall number of parameters. To analyze the effect of n, we conduct experiments by varying n while keeping r fixed at 2 and 4 on four datasets. The results are shown in Figure 4.

We have three observations from the results. First, the optimal n varies across datasets and sometimes differs for different values of r even on the same dataset. For instance, when r is set to 4, the optimal n for QNLI and SST-2 is 4, while for ColA and STSB, it is 2. This observation suggests that the specific task exerts an important influence on the behavior of the model. Second, the performance of MELoRA exhibits a pattern of initially increasing with n and then decreasing, regardless of the values of r. At first, increasing n results in a higher equivalent rank, which is beneficial to the

Method	$r \times n$	#Param.	RTE	CoLA	STS-B	SST-2	QNLI	Avg.
LoRA	8×1	295k	75.63	62.34	**90.71**	**94.50**	**92.55**	83.14
MELoRA	4×2	147k	**76.39**	**62.70**	**90.71**	94.33	92.52	**83.33**
MELoRA	2×4	73k	75.09	61.84	90.60	94.11	92.47	82.82
LoRA	16×1	590k	75.45	**63.28**	90.81	**94.70**	92.50	83.35
MELoRA	8×2	295k	**75.93**	63.10	**90.82**	94.61	**92.61**	**83.42**
MELoRA	4×4	147k	74.37	61.72	90.63	94.54	92.67	82.79
MELoRA	2×8	73k	73.65	61.36	90.41	94.18	92.41	82.40

Table 4: Performance on GLUE for natural language understanding tasks with different numbers of equivalent ranks ($r \times n$), with the same metrics as in Table 2. Boldface indicates best results in terms of the corresponding metrics.

Method	$r \times n$	#Param.	MMLU	DROP	BBH	Avg.
LoRA	16×1	8.4M	45.52	32.14	32.67	36.78
MELoRA	8×2	4.2M	45.63	**32.97**	32.61	37.07
MELoRA	4×4	2.0M	45.23	32.43	32.70	36.79
MELoRA	2×8	1.0M	**46.53**	**32.97**	**33.06**	**37.52**
MELoRA	1×16	0.5M	45.38	31.52	33.29	36.73
LoRA	32×1	16.8M	45.30	32.33	32.42	36.68
MELoRA	16×2	8.4M	45.92	32.60	32.78	37.10
MELoRA	8×4	4.2M	46.05	32.16	33.09	37.10
MELoRA	4×8	2.0M	**46.20**	**33.30**	33.11	**37.54**
MELoRA	2×16	1.0M	45.66	31.84	32.36	36.62
MELoRA	1×32	0.5M	45.60	31.55	**33.35**	36.83
LoRA	64×1	33.6M	45.66	32.46	32.43	36.85
MELoRA	32×2	16.8M	46.03	**32.76**	32.58	37.12
MELoRA	16×4	8.4M	46.15	32.68	**33.11**	**37.31**
MELoRA	8×8	4.2M	**46.43**	32.57	32.55	37.18
MELoRA	4×16	2.0M	46.26	32.57	32.67	37.17
MELoRA	2×32	1.0M	45.43	32.41	32.93	36.92
MELoRA	1×64	0.5M	46.20	31.66	32.45	36.77

Table 5: Performance on INSTRUCTEVAL for instruction following tasks with different equivalent ranks ($r \times n$), with the same metrics as in Table 3. Boldface indicates best results in terms of the corresponding metrics. More results can be found in Appendix B.

performance. However, note that excessively high equivalent ranks pose a risk of overfitting. That is why when n is too big, the performance drops. Third, the optimal n tends to be larger for datasets with more training samples or smaller values of r. As to training samples, for instance, QNLI and SST-2 have more training samples compared to other datasets, thus leading to an optimal n of 4, while for the others, it is 2. This phenomenon can be attributed to the need to allocate a higher equivalent rank to effectively leverage the abundance of training samples. As to rank r, on SST-2 and CoLA, the optimal n for $r = 2$ is greater than that for $r = 4$. That is because the equivalent rank of MELoRA is denoted as $n \times r$. To achieve a specific equivalent rank, the smaller the value of r is, the larger the value of n should be.

6.3 Analysis of the Rank of Mini LoRAs

To analyze the effect of r, we conduct experiments with varying r while keeping n fixed at 1 and 2 on four datasets. When setting n to 1, MELoRA degrades into LoRA. The results are shown in Figure 5. As r increases, the performance first improves and then stabilizes. This observation implies that a higher rank and a large number of trainable parameters are always favored in terms of performance when there is enough training data. However, a higher rank and a large number of trainable parameters usually mean higher training costs. Second, MELoRA consistently outperforms LoRA across all rank settings. That proves that MELoRA is more powerful with the same number of trainable parameters.
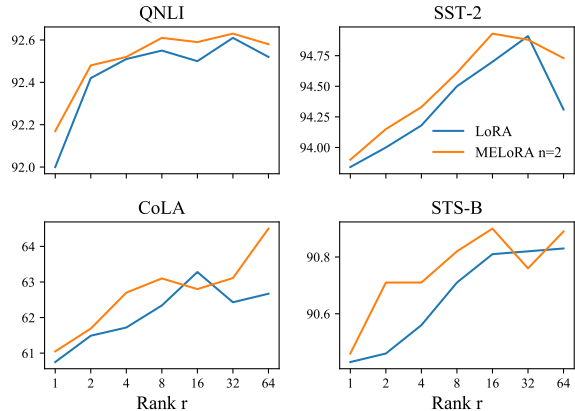

Figure 5: Performance of LoRA and MELoRA with different rank r and fixed n on different datasets. More results can be found in Appendix D.

7 Conclusion

In this paper, we have proposed a new parameter-efficient fine-tuning method, MELoRA, that stacks multiple mini LoRAs in parallel, with each mini LoRA rank learning the different dimensions of a hidden state. We have theoretically demonstrated

that MELoRA maintains a higher and flexible rank, as well as lower complexity. We have also shown empirically that MELoRA achieves a higher rank and better performance with fewer trainable parameters on multiple datasets.

Reproducibility

We release the source code to reproduce the results in this paper at https://github.com/ChasonShi/MELoRA.

Limitations

This work has the following limitations. First, we introduce a new hyper-parameter n, which indicates the number of mini-LoRAs. The best n is varied with different datasets. We need more tuning parameters to get a good performance. In future work, we plan to address these issues by applying hyper-parameter search methods like Bayesian optimization.

Ethical Considerations

We realize that there are risks in developing large language models, so it is necessary to pay attention to the ethical issues. We have used the public pre-trained LLMs, e.g., LLaMA2-7B, RoBERTa-base, and public datasets, i.e., GLUE and INSTRUCTEVAL, to conduct the experiments. All models and datasets are carefully processed by their publishers to ensure that there are no ethical problems.

Acknowledgements

References

Roy Bar-Haim, Ido Dagan, and Idan Szpektor. 2014. Benchmarking applied semantic inference: The PAS-CAL recognising textual entailment challenges. In *Language, Culture, Computation. Computing - Theory and Technology*, volume 8001 of *Lecture Notes in Computer Science*, pages 409–424. Springer.

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*. NIST.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. *CoRR*, abs/1708.00055.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Yew Ken Chia, Pengfei Hong, Lidong Bing, and Soujanya Poria. 2023. INSTRUCTEVAL: Towards holistic evaluation of instruction-tuned large language models. *CoRR*, abs/2306.04757.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized llms. *CoRR*, abs/2305.14314.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023a. Sparse low-rank adaptation of pre-trained language models. In *EMNLP*, pages 4133–4145, Singapore. Association for Computational Linguistics.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023b. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *IJCNLP-IWP*. Asian Federation of Natural Language Processing.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. LoRAMoE: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment. *CoRR*, abs/2312.09979.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *ACL-PASCAL*, pages 1–9, Prague. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *ICLR*.

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 *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *CoRR*, abs/2310.11454.

Neal Lawton, Anoop Kumar, Govind Thattai, Aram Galstyan, and Greg Ver Steeg. 2023. Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models. In *Findings of ACL*, pages 8506–8515, Toronto, Canada. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, pages 3045–3059. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL-IJCNLP*, pages 4582–4597. Association for Computational Linguistics.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. 2023. Stack more layers differently: High-rank training through low-rank updates. *CoRR*, abs/2307.05695.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *EACL*, pages 487–503. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, pages 2383–2392. The Association for Computational Linguistics.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *NeurIPS*, pages 506–516.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. Adapterdrop: On the efficiency of adapters in transformers. In *EMNLP*, pages 7930–7946. Association for Computational Linguistics.

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 *EMNLP*, pages 1631–1642. ACL.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *EACL*, pages 3266–3279. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*. OpenReview.net.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Trans. Assoc. Comput. Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, pages 1112–1122. Association for Computational Linguistics.

Wenhan Xia, Chengwei Qin, and Elad Hazan. 2024. Chain of LoRA: Efficient fine-tuning of language models via residual learning. *CoRR*, abs/2401.04151.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. IncreLoRA: Incremental parameter allocation method for parameter-efficient fine-tuning. *CoRR*, abs/2308.12043.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023b. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning. *CoRR*, abs/2308.03303.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2022. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. 2023. Delta-LoRA: Fine-tuning high-rank parameters with the delta of low-rank matrices.

A Hyper-parameters

The detailed hyper-parameter settings on the IN-STRUCTEVAL and GLUE datasets are listed in Table 6 and 7, respectively.

Hyper-Parameter	
Learning rate η	3e-4
Batch size	128
Number of epochs	3
Max sequence length	256
Rank r	4
LoRA dropout	0.05
LoRA alpha α	16
Trainable matrices	W_Q, W_V
LR scheduler	Linear
Warmup steps	100

Table 6: The hyper-parameter settings for INSTRUCTE-VAL. We use the same settings as Chia et al. (2023)

B Analysis of Equivalent Rank

We further conduct experiments with equivalent ranks 128, 256 and 4096 on INSTRUCTEVAL. The results are listed in Table 8. MELoRA still achieves the best performance on all equivalent rank settings. We have similar observations as in Table 5.

C Analysis of the Number of mini LoRAs

We report more results with different numbers of mini LoRAs in Figure 6 and 7. The performance of MELoRA still exhibits a pattern of initially increasing with n and then decreasing, regardless of the values of r on the QQP and RTE datasets. But on the MRPC and MNLI datasets, the optimal n is 1. That is because those two datasets prefer lower ranks. In that case, MELoRA degrade to LoRA when $n = 1$. On INSTRUCTEVAL, the performance is consistent with that in Figure 4. The best equivalent ranks of MMLU and BBH are 64 and 32, respectively. To achieve a specific equivalent rank, smaller the value of r is, larger the value of n should be. That is consistent with Figure 4 as well.

D Analysis of the Rank of mini LoRAs

We report the results of different ranks of mini LoRAs on the rest datasets of GLUE in Figure 8. MELoRA outperforms LoRA, and the performance first improves and then stabilizes as r increases, which is consistent with Figure 5. Because QQP
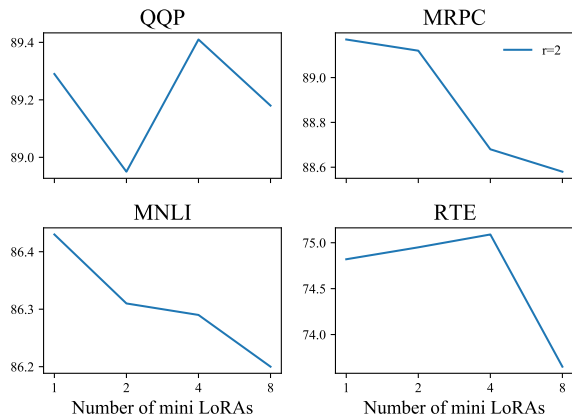

Figure 6: Performance of LoRA and MELoRA with different number of mini LoRAs n and fixed r on GLUE.
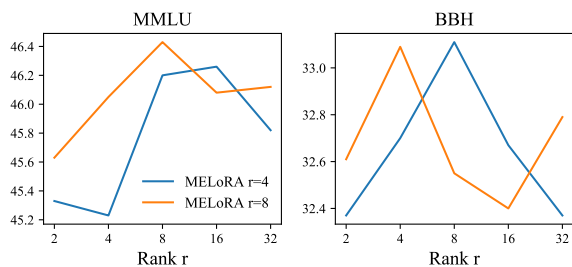

Figure 7: Performance of LoRA and MELoRA with different rank n and fixed r on MMLU and BBH.

and MNLI have more training samples, the optical rank is higher. That also proves that MELoRA is more powerful with the same number of trainable parameters.
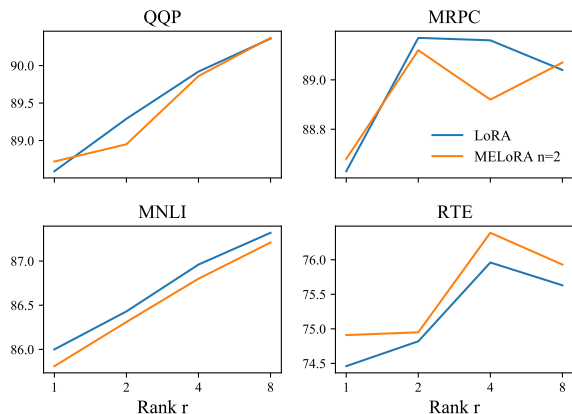

Figure 8: Performance of LoRA and MELoRA with different rank r and fixed n on GLUE.

Hyper-Parameter	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Learning Rate η	5e-4	5e-4	4e-4	4e-4	4e-4	4e-4	4e-4	4e-4
Batch Size	128	128	128	64	128	128	128	128
Number of Epochs	30	60	30	80	25	25	80	40
Weight Decay β	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max Sequence Length	256	256	256	256	256	256	512	256
Start Steps K	2000	400	10	100	800	400	200	200
Update Ratio λ	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Rank r	8	8	8	8	8	8	8	8
Alpha α	16	16	16	16	16	16	16	16
LR Scheduler	Linear	Linear	Linear	Linear	Linear	Linear	Linear	Linear
Trainable Matrices	W_Q,W_V	W_Q,W_V	W_Q,W_V	W_Q,W_V	W_Q,W_V	W_Q,W_V	W_Q,W_V	W_Q,W_V
Warmup Ratio	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
Evaluation Metrics	Accuracy	Accuracy	Accuracy	Matthews Correlation	Accuracy	Accuracy	Accuracy	Pearson

Table 7: The hyper-parameter settings for GLUE. For fair comparison, we use the same settings as Zi et al. (2023).

Method	$r \times n$	#Param.	MMLU	DROP	HEval	BBH	Avg.
LoRA	128×1	67.1M	45.36	32.52	14.63	32.32	31.20
MELoRA	64×2	33.6M	46.05	32.59	15.85	32.54	31.76
MELoRA	32×4	16.8M	46.05	**32.78**	15.24	**33.18**	31.81
MELoRA	16×8	8.4M	**46.40**	32.49	**16.46**	32.85	**32.05**
MELoRA	8×16	4.2M	46.08	32.57	15.24	32.40	31.57
MELoRA	4×32	2.0M	45.82	32.38	15.85	32.37	31.61
MELoRA	2×64	1.0M	45.54	31.49	12.80	32.78	30.65
MELoRA	1×128	0.5M	45.71	31.69	14.02	32.20	30.91
LoRA	256×1	134.2M	45.27	32.28	**16.46**	31.86	31.47
MELoRA	128×2	67.1M	45.95	32.73	**16.46**	32.51	31.91
MELoRA	64×4	33.6M	45.94	32.95	15.85	**33.25**	**32.00**
MELoRA	32×8	16.8M	**46.33**	**32.98**	15.24	32.98	31.88
MELoRA	16×16	8.4M	46.26	32.73	14.02	32.30	31.33
MELoRA	8×32	4.2M	46.12	32.44	14.63	32.79	31.50
MELoRA	4×64	2.0M	46.28	31.31	12.80	32.46	30.71
MELoRA	2×128	1.0M	45.40	32.04	14.63	31.74	30.95
MELoRA	1×256	0.5M	45.25	31.60	14.02	32.65	30.88
MELoRA	2048×2	1073.7M	45.76	32.76	17.07	32.62	32.05
MELoRA	1024×4	536.9M	45.80	32.82	**18.29**	32.93	**32.46**
MELoRA	512×8	268.4M	46.35	**32.89**	15.24	32.82	31.83
MELoRA	256×16	134.2M	46.11	32.51	15.85	32.52	31.75
MELoRA	128×32	67.1M	46.10	32.60	13.41	32.91	31.26
MELoRA	64×64	33.6M	45.96	32.04	15.85	32.14	31.49
MELoRA	32×128	16.8M	46.33	31.85	12.80	32.55	30.88
MELoRA	16×256	8.4M	**46.45**	32.30	13.41	**32.79**	31.24
MELoRA	8×512	4.2M	46.40	32.09	14.63	32.97	31.52
MELoRA	4×1024	2.1M	**46.45**	32.12	14.02	32.87	31.37
MELoRA	2×2048	1.0M	45.99	32.33	13.41	32.41	31.04

Table 8: Performance on INSTRUCTEVAL for instruction following tasks with different equivalent ranks ($r \times n$), with the same metrics as in Table 3. Boldface indicates best results in terms of the corresponding metrics.