LoRaDA: Low-Rank Direct Attention Adaptation for Efficient LLM Fine-tuning

Zhangming Li[1,2], Qinghao Hu[1,3]*, Yiqun Chen[1,2], Peisong Wang[1,2,3], Yifan Zhang[1,2,4], Jian Cheng[1,2,3]*,

[1]The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, Institute of Automation, Chinese Academy of Sciences,
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences, [3]AiRiA
[4]University of Chinese Academy of Science, Nanjing

{lizhangming2023,huqinghao2014,chenyiqun2021}@ia.ac.cn, {peisong.wang,yfzhang,jcheng}@nlpr.ia.ac.cn

Abstract

As the parameter size of language models becomes extremely large, fine-tuning them with limited resources has become a challenging task. Latest advancements in parameter-efficient fine-tuning (PEFT) techniques allow for adjustments to only a minor fraction of the parameters of these LLMs. Yet, most of PEFT methods may suffer from the following limitations: (1) As the rank decreases sharply, PEFT methods like LoRA and Adapter tuning will exhibit significant performance degradation in downstream tasks. (2) An accuracy gap between these methods and full fine-tuning (Full-FT) still exists. To tackle these problems, we propose a Low-Rank Direct Attention Adaptation (LoRaDA) method for efficient LLM fine-tuning. Specifically, we introduce a novel Low-rank Multi-head Attention Map Module (LMAM), which can bring negative attention to self-attention modules and learn low-rank attention weights directly, capturing the characteristics of downstream tasks. Furthermore, LMAM can serve as a plug-in to existing methods, such as LoRA and Adapter, providing state-of-the-art performance even with extreme low rank setting. Extensive experiments on various downstream tasks demonstrate the superior performance of our LoRaDA method. Specifically, LoRaDA even outperforms the full fine-tuning method by up to 2.1% on GLUE benchmark. As a plug-in, LMAM boosts the accuracy of LoRA by up to 27.7% with LLaMA-7B on Commonsense Reasoning benchmark.

1 Introduction

Most traditional NLP tasks acquire domain knowledge from extensive amounts of text data, a naive solution is fine-tuning all model parameters. With the rapid expansion of model parameters and data volume, a multitude of large language models (LLM) have emerged, such as GPT-3 (175B) (Brown et al., 2020), LLaMA (7B-65B) (Touvron et al., 2023), and OPT (125M-175B) (Zhang et al., 2022). The advent of these models has made full fine-tuning increasingly impractical due to the enormous resource consumption.

To address this challenge, researchers consider tuning only a small subset of parameters of LLM while keeping most parameters frozen. For example, LoRA (Hu et al., 2021) employs the concept of low-rank approximation by updating only a subset of the parameters to fine-tuning the model. Adapter (Houlsby et al., 2019) introduces small, trainable modules into each layer of a pre-trained model, allowing for efficient fine-tuning with minimal changes to the original model parameters. All these methods are called parameter-efficient fine-tuning (PEFT) , which significantly reduce the trainable parameters and resource consumption.

However, most PEFT methods (Hu et al., 2021; Houlsby et al., 2019), including their variants(Biderman et al., 2024; Bershatsky et al., 2024; Yang et al., 2024), achieve lower performance compared to full fine-tuning. In addition, the performance of the latest methods (Liu et al., 2024; Yao et al., 2024; Li et al., 2024; Zhou et al., 2024; Lin et al., 2024; Feng et al., 2025) tend to degrade sharply when the rank is significantly reduced.

We observe that most PEFT methods represent attention map values as positive numbers between 0 and 1, and such a positive attention mechanism in LLMs often allocates considerable attention to irrelevant contexts, degrading model performance, as illustrated in (Ye et al., 2024; Liu et al., 2023; Kamradt, 2023). To cure this problem, recent works (Naderi et al., 2024; Laplante et al., 2018; Ye et al.,

*Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 12638–12655
November 4-9, 2025

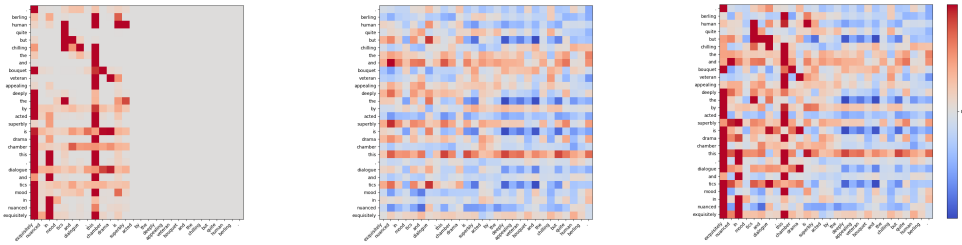
2024; Meng et al., 2025) stress that negative attention can cancel noise and boost the model's capability. For example, inspired by differential amplifiers(Laplante et al., 2018) in electrical engineering which output the difference between two signals to eliminate common-mode noise, DIFF Transformer(Ye et al., 2024) introduces negative attention via a differential attention mechanism, demonstrating the superior performance in long-context modeling, key information retrieval, hallucination mitigation and in-context learning. PolaFormer(Meng et al., 2025) points out that the non-negative constraints on attention maps result in significant information loss, leading to less discriminative attention maps. Yet, these methods aims to study new model architecture with negative attention, and such a model need to be trained from scratch. Thus, these methods can not be directly applied in fine-tuning tasks. Based on this, A naive approach of introducing negative attention to PEFT would be adding learnable attention maps with both negative and non-negative values to the original attention map. However, too many learnable parameters in attention maps will cause overfitting problems, as shown in (Li et al., 2025; Li and Zhang, 2021).

Therefore we introduce a novel static plug-in module called the Low-rank Multi-head Attention Map (LMAM), which learns two low-rank weights on attention maps, thus bring negative attention to standard self-attention modules. As shown in Figure 1 , the color blue indicates negative attentions which suppress less important tokens (e.g., "is", "the", "Berling", "acted"), while the color red represents salient tokens such as "Exquisitely", "superbly", "deeply" and "quit". Thus it can enhance model performance with a minimal rank and effectively capture the characteristics of downstream tasks. As a plug-in, LMAM can enable state-of-the-art PEFT methods to outperform Full-FT in downstream tasks. To demonstrate the effectiveness of our model, we conducted experiments on multiple benchmarks such as Commonsense Reasoning ($+1.2\%$ performance), GLUE ($+2.1\%$ performance), XSum and MT ($+1.24\%$ performance). Our contributions can be briefly summarized as follows:

- We propose a Low-Rank Direct Attention Adaptation (LoRaDA) method for efficient LLM fine-tuning, which can bring negative attention to self-attention modules while directly learning low-rank attention weights.
- LMAM can serve as a static plug-in to existing methods, such as LoRA and Adapter series methods, providing state-of-the-art performance even with extreme low rank setting.
- Extensive experimental results on six standard benchmarks, including Commonsense Reasoning, Long-sequence, Arithmetic Reasoning, GLUE, XSum and MT, demonstrate that the proposed model performs favorably against state-of-the-art fine-tuning methods. Specifically, LoRaDA even outperforms the full fine-tuning method by up to 2.1% on GLUE benchmark. As a plug-in, LMAM boosts the accuracy of LoRA by up to 27.7% with LLaMA-7B on Commonsense Reasoning benchmark.

2 Related Works

As the size of models increases, the limitations of full-parameter fine-tuning become apparent. The high economic cost and the issue of catastrophic forgetting are the two main challenges currently faced. These issues have prompted researchers and engineers to seek more efficient and cost-effective fine-tuning strategies. Parameter Efficient Fine-tuning (PEFT) involves fine-tuning a small number of parameters in large models to reduce costs and avoid catastrophic forgetting. The current PEFT methods can be classified into four primary categories: LoRA-based methods (Hu et al., 2022; Liu et al., 2024; Li et al., 2024; Biderman et al., 2024; Bershatsky et al., 2024; Lin et al., 2024), adapter-based methods (Houlsby et al., 2019; He et al., 2021a; Yang et al., 2024; Li et al., 2024; Zhou et al., 2024), prefix-based methods (Li and Liang, 2021; Zhou et al., 2024; Lester et al., 2021; Hambardzumyan et al., 2021), and others (Chen et al., 2024; Wang et al., 2024; Yao et al., 2024; Bhardwaj et al., 2024; Feng et al., 2025). The core idea of Low-Rank Adaptation (LoRA) (Hu et al., 2022) is to approximate the weight changes in the model through low-rank matrix decomposition. Nevertheless, these methods exhibit considerable limitations, as their performance deteriorates significantly with rank reduction, ultimately failing to adequately capture the nuances of downstream tasks. The second methods of PEFT is Adapter (Houlsby et al., 2019), which adapts to new tasks by inserting additional small network layers, known



(a) Original Attention: $Attn$ (Neg Rat: 0.00%)
(b) LMAM Attention: $Attn_s$ (Neg Rat: 55.10%)
(c) Fused Attention: $\hat{Attn}$ (Neg Rat: 46.56%)

Figure 1: The attention heatmaps are generated using the Bert-base model on the SST2 dataset. They are derived from attention maps of the first head in the model's final layer. (a) represents the Original Attention generated from the multiplication of the Query and Key matrices. (b) indicates the Attention produced by the Low-Rank Multi-head Attention Map Module (LMAM). (c) represents the Fused Attention, obtained by adding (a) and (b). "Neg Rat" represents ratio of negative information. Red indicates positive values, while blue indicates negative values.

as Adapter layers, into the architecture of the pre-trained model, without the need to retrain the entire model. The third PEFT method is Prefix-tuning (Li and Liang, 2021), which adds a small, learnable prefix of parameters at each model layer. These prefixes serve as conditional information, guiding the model to generate outputs that align with specific tasks. Finally, we will introduce the other methods. For instance, QuanTA (Chen et al., 2024) introduce efficient high-rank fine-tuning to potential failures of LoRA's extreme low-rank approach in complex downstream tasks. Feng et al. (2025) introduce a variant of MoE that trains experts orthogonally to encourage diversity.

3 Methodology

In this section, we begin by presenting the preliminaries, including Multi-Head Attention and Differential Attention. We then give details of our proposed low-rank multi-head attention map module (LMAM). Finally, we discuss the extra operation and memory consumption brought by LMAM.

3.1 Preliminaries

Multi-Head Attention The original Attention Map (Attn) is computed by multiplying the query vector Q with the transpose of the key vector K. The specific calculation formula is as follows:

$$Attn = softmax(\frac{QK^T}{\sqrt{d}}), Q = XW_Q, K = XW_K \tag{1}$$

where $X \in \mathbb{R}^{N\times d}$ indicates the input feature, N represents the input length, d denotes the feature dimension. $W_Q, W_K \in \mathbb{R}^{d\times d}$ are the weights matrices. $Q \in \mathbb{R}^{h\times N\times (d/h)}$, $K \in \mathbb{R}^{h\times N\times (d/h)}$, $Attn \in \mathbb{R}^{h\times N\times N}$. h denotes the number of multi-heads. After obtaining the $Attn$, it is multiplied by the value vector V to produce the intermediate representation of the token vectors O:

$$O = Attn \cdot V, V = X \cdot W_V \tag{2}$$

where $V, O \in \mathbb{R}^{h\times N\times (d/h)}$. The outputs O are formed as a convex combination of the value vectors V, with coefficients provided by the attention map $Attn$, whose values range from 0 to 1.

Differential Attention Differential Attention in the DIFF Transformer (Ye et al., 2024) is computed as follows:

$$Diff = softmax(\frac{Q_1K_1^T}{\sqrt{d/2}}) - \lambda softmax(\frac{Q_2K_2^T}{\sqrt{d/2}}) \tag{3}$$

where $[Q_1; Q_2] = XW_Q \in \mathbb{R}^{N\times d}$, $[K_1; K_2] = XW_K \in \mathbb{R}^{N\times d}$, $Q_1, Q_2, K_1, K_2 \in \mathbb{R}^{N\times (d/2)}$. The value of λ can be obtained as follows:

$$\lambda = \exp(\lambda_{q_1} \cdot \lambda_{k_1}) - \exp(\lambda_{q_2} \cdot \lambda_{k_2}) + \lambda_{\text{init}}, \tag{4}$$

where $\lambda_{q_1}, \lambda_{k_1}, \lambda_{q_2}, \lambda_{k_2} \in \mathbb{R}^{d/2}$ are learnable vectors, and $\lambda_{\text{init}} \in (0, 1)$ is a constant.

3.2 Low-rank Multi-head Attention Map Module

As highlighted in previous works (Ye et al., 2024; Liu et al., 2023; Kamradt, 2023), despite the widespread adoption of Transformers in large language models, the conventional softmax-based attention mechanism often allocates considerable attention to irrelevant contexts, degrading model performance. Moreover, PolaFormer(Meng et al., 2025) points out that enforcing non-negative constraints on feature maps results in significant information loss, leading to less discriminative attention maps. To overcome these limitations, the DIFF Transformer utilizes a differential attention mechanism, as defined in Equations 3 and 4, introducing negative attention to effectively suppress noise and

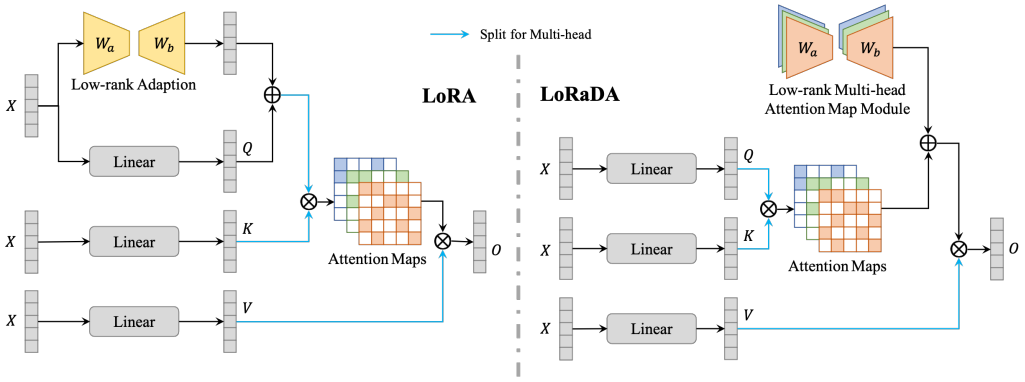


Figure 2: The overall architecture of the proposed Low-Rank Direct Attention Adaptation (LoRaDA) for Efficient LLM Fine-tuning. The Low-Rank Multi-head Attention Map Module, serving as the core structure of this framework, introduces low-rank factors to construct static multi-head self-attention weights, bringing negative information to the attention map. On the left, we apply LoRA solely to the linear transformation W_Q for simplicity.

significantly enhance the model's capability. However, this approach cannot be directly applied to fine-tuning tasks because it alters the model architecture by splitting the query (Q) and key (K) matrices. A naive approach to address this issue is to replace the term $-\lambda softmax(\frac{Q_2 K_2^T}{\sqrt{d/2}})$ with a trainable parameter matrix $W \in \mathbb{R}^{h \times N \times N}$, as they share the same representation range. Consequently, Equation 3 becomes:

$$Attn_{diff} = softmax(\frac{QK^T}{\sqrt{d}}) + W \quad (5)$$

Previous work(Li et al., 2025; Li and Zhang, 2021) has shown that allocating too many fine-tunable parameters can lead models to overfit and, consequently, undermine their robustness. Inspired by LoRA, we propose replacing W with two low-rank matrices, W_a and W_b, significantly reducing the number of trainable parameters. The resulting formulation is expressed as:

$$Attn_{diff} = softmax(\frac{QK^T}{\sqrt{d}}) + W_a W_b \quad (6)$$

where $W_a \in \mathbb{R}^{h_s \times n \times r}$ and $W_b \in \mathbb{R}^{h_s \times r \times n}$. The variable r represents the low-rank value, while n denotes a fixed token length that is independent of the input. Overall, we propose the Low-rank Multi-head Attention Map Module (LMAM), which can bring negative attention to self-attention modules, as illustrated in Figure 2.

To capture the token ordering, we add positional embeddings to W_a and W_b following (Gehring et al., 2017).

$$PE_{pos} = 1/10000^{2pos/n} \quad (7)$$

$$W_a[:, pos, :] = W_a[:, pos, :] + PE_{pos} \quad (8)$$

$$W_b[:, :, pos] = W_b[:, :, pos] + PE_{pos} \quad (9)$$

where pos is the position. We just add PE_{pos} to the second dimension of W_a, and third dimension of W_b. During actual training, the main parameters of the network are kept frozen, while W_a and W_b remain trainable parameters. Parameters W_a and W_b are multiplied through matrix multiplication to generate an $h_s \times n \times n$ attention map:

$$Attn_s = (W_a \cdot W_b) \cdot s \quad (10)$$

where the s is a fixed hyperparameter, which controls the magnitude of the $Attn_s$. Due to the multiplication of W_a and W_b, the resulting cumulative product can numerically far exceed the original Attention Map. To achieve numerical consistency, we introduce a scaling factor s. For decoder-only architectures like LLaMA (Touvron et al., 2023), $Attn_s$ inherently applies a causal mask:

$$Attn_s = Attn_s \odot M_{causal} \quad (11)$$

where $\odot$ represents the dot product of the corresponding positions. M_{causal} is a lower triangular matrix composed exclusively of the elements 0 and 1, used to mask the features of subsequent positions. For other model architectures, the causal mask is not applied. Then, the result is further added to the original Attention Map:

$$\hat{Attn} = Attn + Attn_s \quad (12)$$

where the $+$ is an in-place operation. See the Supplementary Materials for the exact insertion points of $Attn_s$ in the attention map across tasks, and for the theoretical justification of our method.

Finally, this attention map is multiplied with the value vector V in an inner product operation to generate the final output:

$$O = \hat{Attn} \cdot V \quad (13)$$

Table 1: Comparison of differences between LoRA and LMAM across various aspects.

Methods	Attention Map (indirect/direct)	Surpass Full FT (yes/no)	Negative Attention (yes/no)	KV Cache (yes/no)
LoRA	indirect	no	no	yes
LMAM	direct	yes	yes	yes

Table 2: Comparison of the distinct additional operations between LoRA and LMAM during inference.

Methods	Extra Multi.	Extra Add.
LoRA	0	0
LMAM	0	$h\{min(n,N)\}^2 \rightarrow 0$

Table 3: Comparison of additional memory consumption between LMAM and Full-rank Multi-head Attention Map Module (FMAM) in a single LLM layer, where $r = 2$.

Methods	Training	Inference
LMAM	$2hn(12r+n)$	$2hnn$
FMAM	$12hnn$	$2hnn$
LLaMA-7B (LMAM)	$1.19MB$	$1MB$
LLaMA-7B (FMAM)	$6MB$	$1MB$
LLaMA2-70B (LMAM)	$2.38MB$	$2MB$
LLaMA2-70B (FMAM)	$12MB$	$2MB$

As demonstrated in the Table 1, we further clarify the differences between LoRA and LMAM across several specific aspects. Compared to LoRA, our method LMAM utilizes two low-rank matrices to directly fine-tune the attention map. Additionally, as shown in the Figure 1, our method introduces negative information, which can address the limitation: as the rank decreases sharply, PEFT methods like LoRA tuning will exhibit significant performance degradation in downstream tasks. At last, our method can outperform full fine-tuning, whereas LoRA cannot.

3.3 Operation and Memory Consumption

As shown in Table 2, we compare the additional computation operations between LoRA and LMAM during inference. There are no additional computations for the multiplication operations in either LoRA or LMAM. In terms of addition operations, LMAM incurs an additional computational cost of $h\{min(n,N)\}^2$ addition operations, which amounts to approximately $1.97 \times 10^{-5} \times (t_a/t_m)$ of the original LLM's FLOPS. This overhead becomes negligible in LLaMA-7B, where $N = 512$, $d = 4096$, $n = 128$, and $h = 32$. t_a and t_m represent the time cost of a single addition and multiplication operation, respectively. Generally spearking, $t_a \ll t_m$. Using ten runs on a single NVIDIA A100 GPU for the OBQA benchmark(Mihaylov et al., 2018), the merged-LoRA approach completed inference in 624.38 seconds, while our method required 627.26 seconds—about 0.46 % slower. Furthermore, we also show the additional memory consumption in LMAM, as shown in Table 3. We assume that the weights and activations are stored in a 16-bit floating point format, meaning each element requires 2 bytes of storage. During training, the additional memory consumption includes weights parameters W_a and W_b (4hnr), gradients (4hnr), Adam (16hnr), and activations (2hnn), resulting in a total of $2hn(12r+n)$, which amounts to approximately $1.19MB$ in a single LLaMA-7B layer. During the inference phase, the offline weight $Attn_s$ is directly added to the $Attn$, which requires only an additional $1MB$ of memory consumption in a single LLaMA-7B layer. Even when the size of the large language model increases to 70 billion parameters (LLaMA2-70B), it requires only 2.38 MB for training and 2 MB for inference. These indicates that LMAM requires only minimal additional memory during both training and inference.

4 Experiments

4.1 Experimental Setups

4.1.1 Datasets.

Following previous works (He et al., 2021a; Chen et al., 2023, 2024; Liu et al., 2024; Zhou et al., 2024), we conduct our experiments on six different benchmark datasets. (1) GLUE benchmark (Wang et al., 2018). It comprises eight subtasks: CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, and RTE. (2) XSum benchmark (Narayan et al., 2018). This is a natural language generation task primarily focused on generating summaries for English news articles. (3) MT benchmark (Bojar et al., 2016). We just conduct on English to Romanian translation task. (4) Commonsense Reasoning benchmark. It includes eight different reasoning tasks: HellaSwag (Zellers et al., 2019), OBQA (Mihaylov et al., 2018), SIQA (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2020), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), ARC-c and ARC-e (Clark et al., 2018). (5) Long-sequence benchmark. We

train on RedPajama (Weber et al., 2024) and evaluate perplexity using the cleaned Arxiv Math proof-pile dataset (Azerbayev et al., 2022), with sequence lengths ranging from 2048 to 8192. (6) Arithmetic Reasoning benchmark. Following QuanTA, we use the MATH10K dataset (Hu et al., 2023) for tuning, and evaluate testing performance on AQuA(Ling et al., 2017), GSM8K(Cobbe et al., 2021), MAWPS(Koncel-Kedziorski et al., 2016), and SVAMP(Patel et al., 2021).

4.1.2 Evaluation Protocol.

The accuracy score is utilized to evaluate the performance of various methods on the GLUE benchmark, the Commonsense Reasoning benchmark and the Arithmetic Reasoning benchmark. The XSUM benchmark utilizes ROUGE-1, ROUGE-2, and ROUGE-L scores (Lin, 2004) for evaluation. BLEU scores (Papineni et al., 2002) are applied to the MT dataset. The Long-sequence benchmark employs perplexity as its evaluation metric.

4.1.3 Setup.

Following the MAM Adapter (He et al., 2021a) and SoRA (Ding et al., 2023), RoBERTa$_{\text{BASE}}$(Liu et al., 2019) and DeBERTaV3$_{\text{base}}$(He et al., 2021b) are utilized as the underlying pretrained model for the GLUE tasks, while BART$_{\text{LARGE}}$ (Lewis et al., 2019) and mBART$_{\text{LARGE}}$(Liu, 2020) are employed for the XSUM and MT tasks, respectively. In the experiments, the number of attention heads varies from one to the maximum number of heads, and the adapter bottleneck sizes are selected from the set $\{1, 2, 4, 8, 10, 16, 18, 24, 64\}$. We select LLaMA-7B, LLaMA-13B, LLaMA2-7B, LLaMA2-70B, and LLaMA3-8B to conduct large language model fine-tuning experiments on the Commonsense Reasoning benchmark. For the multi-head attention configuration, we choose the number of heads from the set $\{1, 2, 4, 8, 16, 32, 64\}$, with corresponding ranks chosen from the set $\{1, 2, 4, 8\}$. For the Long-sequence benchmark, the LMAM's rank and length are set to 2 and 128, respectively. For the Arithmetic Reasoning benchmark, our rank and length are set to 2 and 256, respectively. More details can be found in supplementary materials.

4.2 Experimental Results

4.2.1 Comparison of Performance on Medium-size LMs.

We start by conducting experiments on a set of medium-size LMs, including RoBERTa$_{\text{BASE}}$, BART$_{\text{LARGE}}$, and mBART$_{\text{LARGE}}$. In this setting, our method LMAM incorporates Adapter-FFN (He et al., 2021a) for tuning FFN modules, resulting in a new variant called LoRaDA-A. As shown in Table 5, LoRaDA-A (0.5%) significantly outperforms all other fine-tuning methods while using the same proportion of trainable parameters, achieving a 2.1% accuracy gain over Full-FT. Moreover, LMAM serves as an effective plug-in enhancement, boosting the accuracy of Adapter-FFN by 3.1%. To evaluate LoRaDA-A's effectiveness under extreme low-rank conditions ($rank \leqslant 2$), we compared LoRaDA-A (0.06%) with Adapter-FFN (0.06%). The results show that LoRaDA-A (0.06%) improves model performance by 6.1%, demonstrating LMAM's robustness even with severe rank constraints. Similar experimental results are also observed on the XSum and MT benchmarks; the GLUE benchmark is also evaluated using DeBERTaV3$_{\text{base}}$, with detailed results provided in the supplementary materials.

4.2.2 Experiment Results on LLMs.

Our method LMAM integrates LoRA and Adapter-FFN (He et al., 2021a) for fine-tuning the FFN modules, resulting in a new variant called LoRaDA-LA. Additionally, incorporating LMAM into the QuanTA (Chen et al., 2024) method produces a variant named LoRaDA-Q. As shown in Table 6, LoRA achieves an average accuracy of 39.5% with 0.10% trained parameter. However, integrating the LMAM method results in a 27.7% increase in accuracy. In addition, adding LMAM to the Parallel method (He et al., 2021a) achieves a 5.9% improvement in accuracy. For the state-of-the-art method QuanTA, incorporating LMAM further improves model performance by 1.2%. To further demonstrate our method's effectiveness on more challenging tasks, we conducted experiments as shown in Table 7. With LMAM's length fixed at 256, LongLoRA+LMAM consistently achieves lower perplexity scores compared to LongLoRA alone across a wide range of context lengths (2048–8192 tokens) in the Proof-Pile dataset. These results highlight LMAM's capability to handle tasks involving significant variations in text length. Furthermore, on Arithmetic Reasoning benchmark, as shown in Table 8, LoRaDA-Q achieves accuracy improvements of 1% and 1.9% compared to QuanTA and LoRA, respectively, reinforcing the effectiveness of LMAM as a plug-in approach. Overall, these experimental results strongly confirm that

Table 4: Accuracy comparison of existing methods on eight Commonsense Reasoning datasets. The results for the PEFT methods (Prefix, Series, Parallel, LoRA), ChatGPT, and DoRA are sourced from ((Liu et al., 2024)), while others are sourced from their paper. The scores with underline denote the suboptimal results.

Model	Methods	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	LoRA	0.10	2.3	46.1	18.3	19.7	55.2	65.4	51.9	57	39.5
	IST	-	68.7	81.7	77.3	82.7	78.7	80.6	62.4	80.0	76.5
	SHiRA-SNIP	1.0	68.3	80.6	79.1	82.1	80.0	81.5	67.9	79.6	77.4
	DoRA	0.11	51.3	42.2	77.8	25.4	78.8	78.7	62.5	78.6	61.9
	DoRA	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	77.5
	DoRA	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	78.4
	QuanTA	0.041	71.6	83.0	79.7	91.8	81.8	84.0	68.3	82.1	80.3
	LoRaDA-LA	0.09	71.3	84.4	80.9	88.5	83.2	83.5	68.8	79.4	80.0
	LoRaDA-Q	0.044	72.9	84.2	80.9	92.0	83.2	86.0	70.8	81.9	**81.5**
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	OwLore-Full	-	85.1	80.3	34.5	59.8	80.5	80.1	51.5	39.2	63.9
	OwLore	-	85.4	80.7	34.2	60.3	82.2	80.6	51.0	39.1	64.2
	LaMDA++	0.08	71.8	80.6	79.5	84.0	82.7	81.5	66.0	80.6	78.3
	SHiRA-SNIP	1.0	70.42	81.71	79.01	89.78	80.51	83.25	68.6	81.0	79.3
	OMoE-DoRA	0.73	73.1	83.3	79.0	93.0	68.4	80.1	55.3	80.0	76.5
	DoRA	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	80.5
	DoRA	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
	QuanTA	0.041	72.4	83.8	79.7	92.5	83.9	85.3	72.5	82.6	81.6
	LoRaDA-LA	0.08	73.2	85.2	82.5	90.3	84.6	85.5	71.8	83.8	82.1
	LoRaDA-Q	0.044	74.3	85.1	81.9	93.9	83.7	85.7	71.6	85.2	**82.7**
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	OwLore-Full	-	86.8	81.6	34.2	62.9	84.1	81.9	53.3	40.2	65.6
	OwLore	-	86.6	82.3	33.8	63.0	83.5	83.2	55.3	38.6	65.8
	IST	-	72.7	88.3	80.5	94.7	84.4	89.8	79.9	86.6	84.6
	OMoE-DoRA	0.75	74.7	87.8	79.9	95.0	85.2	89.0	76.7	86.4	84.3
	DoRA	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	85.0
	DoRA	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	85.2
	QuanTA	0.035	74.3	88.1	81.8	95.1	87.3	91.1	81.7	87.2	85.8
	LoRaDA-LA	0.05	73.6	89.9	81.2	95.3	86.5	90.5	79.9	87.2	85.5
	LoRaDA-Q	0.037	74.8	90.4	82.3	96.4	88.3	92.1	81.6	88.2	**86.8**
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	80.8
	QuanTA	0.029	73.2	85.4	82.1	93.4	85.1	87.8	73.3	84.4	83.1
	LoRaDA-LA	0.05	73.7	86.6	82.9	93.9	86.5	86.3	72.6	84.8	83.4
	LoRaDA-Q	0.029	72.9	87.4	84.2	94.2	87.3	86.5	74.2	84.7	**83.9**
LLaMA2-70B	LoRaDA-LA	0.029	79.2	93.0	84.9	97.2	92.7	95.2	85.0	93.0	**90.0**

LMAM, as a plug-in method, can substantially enhance model performance.

As presented in Table 4, LoRaDA-Q achieves an accuracy improvement of 1.2% over state-of-the-art method QuanTA on LLaMA-7B, 0.8% on LLaMA-13B, 1.1% on LLaMA2-7B, and 1% on LLaMA3-8B. Furthermore, LoRaDA-LA outperforms state-of-the-art method QuanTA by 0.3% on LLaMA-13B and 0.5% on LLaMA2-7B in terms of accuracy. We also conduct experiments on the billion-parameter LLaMA2-70B model, achieving an average accuracy of 90.0%. Finally, we evaluated the stability of LoRaDA-LA, with the results presented in the supplementary materials.

For LLaMA-7B, PEFT methods like LoRA, as the number of parameters decreases (from 0.83% with rank = 32 to 0.10% with rank = 4), the model accuracy significantly drops from 74.7% to 39.5%. For the LoRA variant method DoRA, there is also a noticeable decline in model performance when the rank is extremely low. Specifically, as the parameters decrease from 0.84% to 0.11%, the model accuracy drops from 78.4% to 61.9%. However, our method LoRaDA-LA (0.09% with rank = 2) achieves 80.0% performance, which is 40.5% higher than LoRA (0.10%) and 18.1% higher than DoRA (0.11%). This indicates that while PEFT methods and their variants with extremely low-rank settings tend to fail, our approach LoRaDA-LA can still maintain high model performance even with a very low rank.

For LLaMA2-7B, we compare our method LoRaDA-LA with additional state-of-the-art methods, i.e., OwLore-Full(Li et al., 2024), OwLore(Li et al., 2024), LaMDA++(Azizi et al., 2024),SHIiRA-SNIP(Bhardwaj et al., 2024), DoRA and OMoE-DoRA(Feng et al., 2025) in our experiments. Under the same fine-tuning parameters ratio of 0.08%, our method still outperforms LaMDA++ with an average accuracy im-

Table 5: Accuracy of the base model RoBERTa$_{\text{BASE}}$ on the dev set of GLUE. The results (Full fine-tuning, Bitfit, Prefix, LoRA, Adapter, Adapter-FFN, MAM Adapter) for MNLI and SST2 are sourced from (He et al., 2021a). We have made our best effort to conduct experiments on the remaining GLUE sub-tasks following the methodology of (He et al., 2021a).

Method (# params)	MNLI	SST2	QQP	QNLI	STS-B	MRPC	RTE	CoLA	mean
Full fine-tuning (100%)	$87.6_{\pm.4}$	$94.6_{\pm.4}$	$91.9_{\pm.1}$	$92.8_{\pm.5}$	$\mathbf{91.2}_{\pm.3}$	$90.2_{\pm.3}$	$78.7_{\pm.4}$	$63.6_{\pm.6}$	86.3
Bitfit (0.1 %)(Zaken et al., 2021)	84.7	93.7	87.3	91.0	89.5	88.1	69.8	54.0	82.2
Prefix (0.5%)	$86.3_{\pm.4}$	$94.0_{\pm.1}$	$89.3_{\pm.1}$	$90.8_{\pm.2}$	$88.8_{\pm.1}$	$85.9_{\pm.3}$	$70.9_{\pm.4}$	$59.9_{\pm.6}$	83.2
LoRA (0.5%)	$87.2_{\pm.4}$	$94.2_{\pm.2}$	$90.3_{\pm.2}$	$92.7_{\pm.4}$	$90.5_{\pm.1}$	$88.6_{\pm.3}$	$75.4_{\pm.2}$	$60.1_{\pm.5}$	84.8
Adapter (0.5%)	$87.2_{\pm.2}$	$94.2_{\pm.1}$	$90.1_{\pm.4}$	$92.2_{\pm.2}$	$89.3_{\pm.1}$	$88.2_{\pm.4}$	$75.5_{\pm.6}$	$60.2_{\pm.2}$	84.6
Adapter-FFN (0.5%)	$87.5_{\pm.1}$	$94.1_{\pm.4}$	$90.9_{\pm.2}$	$92.2_{\pm.1}$	$90.7_{\pm.4}$	$88.9_{\pm.3}$	$76.4_{\pm.1}$	$61.7_{\pm.6}$	85.3
MAM Adapter (0.5%)	$87.4_{\pm.3}$	$94.2_{\pm.3}$	$90.7_{\pm.2}$	$91.5_{\pm.3}$	$89.7_{\pm.2}$	$87.8_{\pm.6}$	$72.8_{\pm.3}$	$61.4_{\pm.6}$	84.4
AUTOPEFT (0.5%) (Zhou et al., 2024)	$85.5_{\pm.2}$	$93.2_{\pm.3}$	$90.3_{\pm.1}$	$91.2_{\pm.4}$	$89.9_{\pm.2}$	$87.1_{\pm.5}$	$73.1_{\pm.3}$	$62.5_{\pm.2}$	84.1
LoRaDA-A (0.5%)	$\mathbf{88.2}_{\pm.5}$	$\mathbf{95.9}_{\pm.4}$	$\mathbf{92.8}_{\pm.6}$	$\mathbf{93.9}_{\pm.3}$	$91.0_{\pm.2}$	$\mathbf{92.0}_{\pm.5}$	$\mathbf{83.1}_{\pm.3}$	$\mathbf{69.9}_{\pm.3}$	**88.4**
Adapter-FFN (0.06%)	$80.4_{\pm.6}$	$92.0_{\pm.9}$	$85.8_{\pm.5}$	$87.9_{\pm.6}$	$86.2_{\pm.6}$	$84.9_{\pm.4}$	$68.4_{\pm.5}$	$52.6_{\pm.5}$	79.8
LoRaDA-A (0.06%)	$\mathbf{86.3}_{\pm.5}$	$94.4_{\pm.5}$	$88.9_{\pm.7}$	$91.7_{\pm.8}$	$\mathbf{91.8}_{\pm.9}$	$\mathbf{90.7}_{\pm.3}$	$\mathbf{78.5}_{\pm.7}$	$\mathbf{65.2}_{\pm.7}$	**85.9**

Table 6: Accuracy comparison of LLaMA-7B using existing methods versus using LMAM as a plugin in these methods.

Model	Methods	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
LLaMA-7B	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	Parallel+LMAM	3.54	69.6	82.1	79.4	86.7	82.5	81.5	65.6	77.4	**78.1 (+5.9)**
	LoRA	0.10	2.3	46.1	18.3	19.7	55.2	65.4	51.9	57	39.5
	LoRA+LMAM	0.10	50.3	78.1	77.6	56.1	56.3	79.8	62.9	76.4	**67.2 (+27.7)**
	QuanTA	0.041	71.6	83.0	79.7	91.8	81.8	84.0	68.3	82.1	80.3
	QuanTA+LMAM	0.044	72.9	84.2	80.9	92.0	83.2	86.0	70.8	81.9	**81.5 (+1.2)**

provement of 3.8%. Besides, our method outperforms OwLore-Full and OwLore by 18.2% and 17.9% in average accuracy, respectively. Except for QuanTA, DoRA achieved the highest performance among other methods with 80.5% accuracy, but it still falls 1.6% short of our method, despite having 5.4 times more trainable parameters.

For LLaMA3-8B, our method LoRaDA-LA outperforms the models LORA, DoRA, OwLore-Full, OwLore, IST(Yao et al., 2024) and OMoE-DoRA by 4.7%, 0.3%, 19.9%, 19.7%, 0.9% and 1.2% in accuracy, respectively, when using only 0.05% of tunable parameters. The comparison between the fine-tuning methods of LLaMA-7B and LLaMA-8B reveals that as the parameters of the larger model increase, the parameter amount required for fine-tuning decreases. Specifically, LoRA's trained parameter amount dropped from 0.83% to 0.70%, DoRA's from 0.84% to 0.71%, and our method's from 0.09% to 0.05%. Our method exhibits a more significant reduction in the trained parameter ratio.

4.3 Ablation Study.

The proposed LoRaDA-LA contains multiple key components (i.e., Negative information, Low-rank attention maps, and the Position embedding). In this section, we perform an ablation study on the Commonsense Reasoning benchmark, and compare variants of LoRaDA-LA with respect to the following aspects to demonstrate the effectiveness of LoRaDA-LA:

- **LoRaDA-LA$\neg N$:** $Attn_s$ is incorporated into Eq.1 as $Attn = softmax(\frac{QK^T}{\sqrt{d}} + Attn_s)$
- **LoRaDA-LA$\neg S$:** Characterized by $Attn =$ softmax$(\frac{QK^T}{\sqrt{d}})$ + softmax$(Attn_s)$
- **LoRaDA-LA$\neg L$:** A variant of LoRaDA-LA, in which W_a and W_b in LMAM are removed and the full-rank $Attn_s \in \mathbb{R}^{h\times n\times n}$ is directly fine-tuned, is referred to as FMAM.
- **LoRaDA-LA$\neg P$:** A variant of LoRaDA-LA with the Position embedding being removed.

As shown in Table 9, the removal of negative information results in LoRaDA-LA$\neg N$ achieving 4.8% lower accuracy compared to LoRaDA-LA. It indicates that negative information plays a crucial role in downstream tasks, which is also demonstrated by the performance comparison between LoRaDA-LA$\neg S$ (37.1%) and LoRaDA-LA (80.0%). LoRaDA-LA$\neg L$ falls 7.8% behind LoRaDA-LA in accuracy, requiring 3.6 times more parameters. In addition, we compare the additional memory consumption between LMAM and FMAM within a single LLaMA-7B layer, as shown in Table

Table 7: Perplexity evaluation on proof-pile(Rae et al., 2019) test split. The result of LongLoRA is sourced from (Chen et al., 2023).

Model	Methods	# Params (%)	Eval Context Length (2048,PPL ↓)	Eval Context Length (4096,PPL ↓)	Eval Context Length (8192,PPL ↓)
LLaMA2-7B	LongLoRA	2.1%	3.20	2.91	2.72
	LongLoRA+LMAM	2.1%	**3.15**	**2.88**	**2.70**

Table 8: Performance Comparison on Arithmetic Reasoning benchmark.The results for the LoRA and QuantTA methods are adopted from (Chen et al., 2024).

Model	Methods	# Params (%)	AQuA	GSM8K	MAWPS	SVAMP	Avg. w/o AQuA
LLaMA2-7B	LoRA	0.83%	**17.5**	65.7	91.2	80.8	79.6
	QuantTA	0.19%	16.7	67.0	94.3	80.3	80.5
	LoRaDA-Q (ours)	0.19%	16.8	**67.3**	**94.6**	**82.5**	**81.5**

Table 9: Ablation study of LoRaDA-LA on the LLaMA-7B model, focusing on its performance on the Commonsense Reasoning benchmark. Detailed experimental results can be found in supplementary materials

Methods	Negative	Low-rank	Position	Avg. Acc
LoRaDA-LA (0.09%)	✓	✓	✓	**80.0**
LoRaDA-LA$\neg N$ (0.09%)	×	✓	✓	75.2
LoRaDA-LA$\neg S$ (0.09%)	×	✓	✓	37.1
LoRaDA-LA$\neg L$ (0.32%)	✓	×	✓	72.2
LoRaDA-LA $\neg P$ (0.09%)	✓	✓	×	77.1

3. During training, FMAM incurs 5.04 times the memory cost of LMAM. This indicates that directly tuning full-rank static attention map $Attn_s$ not only results in significant memory consumption but also leads to a decline in model performance. Compared to LoRaDA-LA$\neg P$, LoRaDA-LA achieves an additional gain of 2.9%. This indicates that position embeddings enable the model to capture token position information, thereby enhancing model performance. Overall, negative information, low-rank attention maps, and positional embeddings all play a significant role in enhancing model performance.

5 Conclusion

In this work, we propose a Low-Rank Direct Attention Adaptation (LoRaDA) method for efficient LLM fine-tuning. To address the issue of low-rank limitations, we introduce a novel Low-rank Multi-head Attention Map Module (LMAM), which can bring negative attention to self-attention modules and learn low-rank attention weights directly, capturing the characteristics of downstream tasks. To bridge the gap between full fine-tuning and existing methods, LMAM can serve as a plug-in method to existing methods, providing higher performance than full fine-tuning. Extensive experimental results on six standard benchmarks demonstrate that our model outperforms state-of-the-art approaches.

Limitations

Our research primarily focuses on applying language models to downstream tasks; we plan to extend this work to visual and multimodal models.

Acknowledgments

This work was supported in part by National Key R&D Program of China (No. 2025ZD0122000), Beijing Natural Science Foundation (L244046), the Science and Technology Major Special Program of Jiangsu (No.BG2024028), Jiangsu Key R&D Program for Industry Prospect and Core Technological Innovations Project (No. BE2023016), and Natural Science Foundation of Jiangsu Province under Grant BK20243051.

References

Zhangir Azerbayev, Edward Ayers, and Bartosz Piotrowski. 2022. Proof-pile, 2022. *URL https://github.com/zhangir-azerbayev/proof-pile*.

Seyedarmin Azizi, Souvik Kundu, and Massoud Pedram. 2024. Lamda: Large model fine-tuning via spectrally decomposed low-dimensional adaptation. *arXiv preprint arXiv:2406.12832*.

Daniel Bershatsky, Daria Cherniuk, Talgat Daulbaev, Aleksandr Mikhalev, and Ivan Oseledets. 2024. Lotr: Low tensor rank weight adaptation. *arXiv preprint arXiv:2402.01376*.

Kartikeya Bhardwaj, Nilesh Prasad Pandey, Sweta Priyadarshi, Viswanath Ganapathy, Rafael Esteves, Shreya Kadambi, Shubhankar Borse, Paul Whatmough, Risheek Garrepalli, Mart Van Baalen, et al. 2024. Rapid switching and multi-adapter fusion via sparse high rank adapters. *arXiv preprint arXiv:2407.16712*.

Dan Biderman, Jose Gonzalez Ortiz, Jacob Portes, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan

Frankle, et al. 2024. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, page 7432–7439.

Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. 2016. Findings of the 2016 conference on machine translation (wmt16). In *First conference on machine translation*, pages 131–198. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*.

Zhuo Chen, Rumen Dangovski, Charlotte Loh, Owen Dugan, Di Luo, and Marin Soljačić. 2024. Quanta: Efficient high-rank fine-tuning of llms with quantum-informed tensor adaptation. *arXiv preprint arXiv:2406.00132*.

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. In *Proceedings of the 2019 Conference of the North*.

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*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. *arXiv preprint arXiv:2311.11696*.

Jinyuan Feng, Zhiqiang Pu, Tianyi Hu, Dongmin Li, Xiaolin Ai, and Huimu Wang. 2025. Omoe: Diversifying mixture of low-rank adaptation by orthogonal finetuning. *arXiv preprint arXiv:2501.10062*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. Warp: Word-level adversarial reprogramming. *arXiv preprint arXiv:2101.00121*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021b. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

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, 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*.

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 *International Conference on Learning Representations*.

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*.

Greg Kamradt. 2023. Needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main`.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157.

Philip A Laplante, Robin Cravey, Lawrence P Dunleavy, James L Antonakos, Rodney LeRoy, Jack East, Nicholas E Buris, Christopher J Conant, Lawrence Fryda, Robert William Boyd, et al. 2018. *Comprehensive dictionary of electrical engineering*. CRC Press.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural

language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Dongyue Li and Hongyang Zhang. 2021. Improved regularization and robustness for fine-tuning in neural networks. *Advances in Neural Information Processing Systems*, 34:27249–27262.

Kunyang Li, Jean-Charles Noirot Ferrand, Ryan Sheatsley, Blaine Hoak, Yohan Beugin, Eric Pauley, and Patrick McDaniel. 2025. On the robustness tradeoff in fine-tuning. *arXiv preprint arXiv:2503.14836*.

Pengxiang Li, Lu Yin, Xiaowei Gao, and Shiwei Liu. 2024. Owlore: Outlier-weighed layerwise sampled low-rank projection for memory-efficient llm fine-tuning. *arXiv preprint arXiv:2405.18380*.

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

Cheng Lin, Lujun Li, Dezhi Li, Jie Zou, Wenhan Luo, Wei Xue, and Yike Guo. 2024. Nora: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv preprint arXiv:2408.10280*.

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

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

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Y Liu. 2020. Multilingual denoising pre-training for neural machine translation. *arXiv preprint arXiv:2001.08210*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Weikang Meng, Yadan Luo, Xin Li, Dongmei Jiang, and Zheng Zhang. 2025. Polaformer: Polarity-aware linear attention for vision transformers. *arXiv preprint arXiv:2501.15061*.

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. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Alireza Naderi, Thiziri Nait Saada, and Jared Tanner. 2024. Mind the gap: a spectral analysis of rank collapse and signal propagation in transformers. *arXiv preprint arXiv:2410.07799*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Samet Oymak and Mahdi Soltanolkotabi. 2019. Overparameterized nonlinear learning: Gradient descent takes the shortest path? In *International Conference on Machine Learning*, pages 4951–4960. PMLR.

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.

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

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, page 8732–8740.

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

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*.

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*.

Qibin Wang, Xiaolin Hu, Weikai Xu, Wei Liu, Jian Luan, and Bin Wang. 2024. Pmss: Pretrained matrices skeleton selection for llm fine-tuning. *arXiv preprint arXiv:2409.16722*.

Maurice Weber, Dan Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. 2024. Redpajama: an open dataset for training large language models. *Advances in neural information processing systems*, 37:116462–116492.

Yifan Yang, Jiajun Zhou, Ngai Wong, and Zheng Zhang. 2024. Loretta: Low-rank economic tensor-train adaptation for ultra-low-parameter fine-tuning of large language models. *arXiv preprint arXiv:2402.11417*.

Kai Yao, Penlei Gao, Lichun Li, Yuan Zhao, Xiaofeng Wang, Wei Wang, and Jianke Zhu. 2024. Layer-wise importance matters: Less memory for better performance in parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2410.11772*.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. 2024. Differential transformer. *arXiv preprint arXiv:2410.05258*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2024. Autopeft: Automatic configuration search for parameter-efficient fine-tuning. *Transactions of the Association for Computational Linguistics*, 12:525–542.

A Appendix

This supplementary material provides additional details and results to complement the main paper. It is organized as follows: Appendix A.1 provides additional implementation details for LMAM. In Appendix A.2, we present an in-depth analysis of the experimental results, including a detailed comparison of performance on medium-sized language models (LMs), evaluations of training stability, extensive ablation studies and hyperparameters analysis for LoRaDA. Appendix A.3 outlines the core implementation details, ensuring reproducibility of our method. Appendix A.4 showcases representative generation examples that further illustrate the efficacy of our approach. Appendix A.5 specifies the environment and configurations used to conduct all experiments, facilitating a clear understanding of the setup. Finally, Appendix A.6 presents Theoretical Analysis of LMAM.

A.1 More Details in LMAM

In the above discussion of LMAM, the result of $Attn_s$ is further added to the original Attention Map, which will be categorized into the following two cases, when $n \leqslant N$:

$$\hat{Attn}[:, :n, :n] = Attn[:, :n, :n] + Attn_s \quad (14)$$

when $n > N$:

$$\hat{Attn} = Attn + Attn_s[:, :N, :N] \quad (15)$$

where the $+$ is an in-place operation and N represents the input length. For long-sequence benchmarks—i.e., when $n \ll N$—we add $Attn_s$ to the model's attention scores for the final n tokens, specifically as:

$$\hat{Attn}[:, N-n:, N-n:] = Attn[:, N-n:, N-n:] + Attn_s\,. \quad (16)$$

From the viewpoint of attention computation, each later-generated token already encapsulates all the information from the tokens that came before it. In our method we therefore keep the trailing tokens when clipping, so the model can still learn the relevant expressions they carry. Moreover, in token-pruning or prompt-pruning studies, many papers determine attention importance precisely by inspecting the last tokens in a sequence—underscoring just how critical the final n tokens are.

Furthermore, to reduce the number of trainable parameters, the number of heads in LMAM is often set to be fewer than the number of heads in the attention map. In this scenario, calculations can proceed as follows:

$$\hat{Attn} = Attn + repeat(Attn_s, [c:, 1, 1]), c = \frac{h}{h_s} \quad (17)$$

where $repeat$ represents the repeat operation in Pytorch. h_s and h indicate the number of heads in the LMAM and the original Attention Map, respectively. In general, h_s is divisible by h. For instance, if $h = 32$, then the h_s can be chosen from $1, 2, 4, 8, 16, 32$.

A.2 Experiments

Table 10: Comparison of various parameter-efficient tuning methods on XSum and MT benchmark. "†" are results copied from MAM Adapter.

Method	# params	XSum (R-1/2/L)	MT (BLEU)
Full fine-tuning†	100%	44.81/21.94/36.83	37.3
Pfeiffer adapter, r=600†	7.2%	$44.03/20.89/35.89_{\pm.13/.10/.08}$	$36.9_{\pm.1}$
LoRA (ffn), r=102†	7.2%	$44.53/21.29/36.28_{\pm.14/.07/.10}$	$36.8_{\pm.3}$
PA (attn, r=30) + PA (ffn, r=512)†	6.7%	$44.29/21.06/36.12_{\pm.31/.19/.18}$	$37.2_{\pm.1}$
Prefix tuning (attn, l=30) + LoRA (ffn, r=102)†	6.7%	$44.84/21.71/36.77_{\pm.07/.05/.03}$	$37.0_{\pm.1}$
MAM Adapter (l=30, r=512)†	6.7%	$45.06/21.90/36.87_{\pm.08/.01/.04}$	$37.5_{\pm.1}$
AUTOPEFT	6.7%	$44.97/21.78/36.92_{\pm.11/.06/.08}$	$36.9_{\pm.2}$
LoRaDA-A	6.7%	**46.08/22.57/38.11**$_{\pm.09/.05/.08}$	**38.0**$_{\pm.2}$

A.2.1 Comparison of Performance on Medium-size LMs.

In order to assess the efficacy of our method LoRaDA-A on other NLP tasks, we conduct experiments on XSum and MT benchmarks, as shown in Table 10. Compared to full fine-tuning, LoRaDA-A achieves improvements of 1.27, 0.63, and 1.28 in R-1, R-2, and R-L scores, respectively, on the XSum dataset, as well as a 0.7 increase in BLEU score on the MT dataset. This indicates that LoRaDA-A can outperform full fine-tuning while using 14.9 times fewer trainable parameters. For the state-of-the-art MAM Adapter method, LoRaDA-A achieves 1.02, 0.67, and 1.24 higher R-1, R-2, and R-L scores, respectively, on the XSum benchmark, and a 0.5 increase in BLEU score on the MT benchmark.

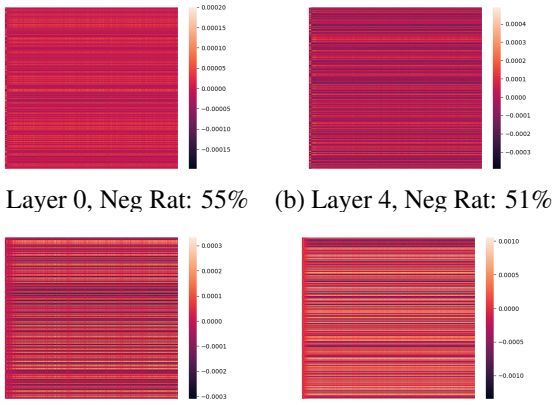
To assess the effectiveness of LoRaDA-A on another backbone, we applied it (with LMAM's rank and length set to 2 and 128, respectively) to the DeBERTaV3-base model (He et al., 2021b). As shown in Table 11, our method outperforms SoRA (Ding et al., 2023) and AdaLoRA (Zhang et al., 2023) on the GLUE benchmark while using fewer trainable parameters.

Table 11: Performance Comparison on the GLUE Benchmark using DeBERTaV3-base. The results for the SoRA and AdaLoRA are sourced from (Ding et al., 2023).

Method	# Params	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg.
AdaLoRA	1.27M	70.86	95.95	90.22	92.13/88.41	91.39	90.27/90.30	94.28	87.36	88.83
SoRA	0.91M	71.48	95.64	91.98	92.39/89.87	92.22	**90.35/90.38**	94.28	**87.77**	89.36
LoRaDA-A (r=2)	0.29M	**78.56**	**96.74**	**93.29**	**93.91/91.24**	**92.98**	90.12/90.16	**94.96**	86.58	**90.89**

Table 12: Training Stability of LoRaDA-LA on Commonsense Reasoning datasets (mean ± std over 3 seeds following (He et al., 2021a)).

Model	# Params (%)	Avg. (%)
LLaMA-7B	0.009	80.0 ± 0.1
LLaMA2-7B	0.008	82.1 ± 0.2
LLaMA3-8B	0.05	85.5 ± 0.1
LLaMA-13B	0.029	83.4 ± 0.1



(a) Layer 0, Neg Rat: 55% (b) Layer 4, Neg Rat: 51%

(c)Layer 7,Neg Rat: 53% (d)Layer 10,Neg Rat: 51%

Figure 3: Visualization of the feature information Negative Ratio across different layers of the Attention Map in Ours (0.06%) on the SST2 test dataset using $\text{RoBERTa}_{\text{BASE}}$, with both the number of heads and rank set to 1.

A.2.2 More Experiment Results of LMAM as Plug-in.

As shown in Table 18, the Adapter approach achieves an average accuracy of 46.5% with only 0.008% of trainable parameters (rank=2). However, incorporating the LMAM method into this method yields a 21.1% improvement in accuracy. Similarly, LoRA, utilizing 0.10% of trainable parameters (rank=4), achieves only 39.5% accuracy in downstream tasks, indicating significant performance degradation. By integrating LMAM with LoRA, the accuracy improves by 27.7%, highlighting the efficacy of LMAM in enhancing task performance. Furthermore, DoRA, with 0.11% of trainable parameters (rank=4), attains 61.9% accuracy, while the addition of LMAM leads to a 15.8% accuracy gain. These experimental results significant demonstrate that LMAM, as a versatile plug-in module, can significantly enhance model performance with extreme low rank setting.

A.2.3 Training stability.

In order to verify the training stability of LoRaDA-A under the same low-rank setting used in Table 4, we ran the method with three different random seeds following (He et al., 2021a) on Commonsense Reasoning datasets. As shown in Table 12, across four backbone sizes (LLaMA-7B, LLaMA2-7B, LLaMA3-8B, and LLaMA-13B), LoRaDA-A achieves consistently high performance with very low variance (e.g.,85.5 ± 0.1 on LLaMA3-8B), confirming that our approach not only boosts accuracy but also yields stable training behavior.

A.2.4 More Experiment Results of LoRaDA in Ablation Studies

As shown in Table 14, we provide the detailed ablation study results in Table 9 of the main paper. Adhering to the experimental settings outlined in Table 9 of the main paper, we conducted additional ablation studies on the GLUE benchmark, which are presented in Table 13. We also present a visualization of the Attention Map in LMAM, as shown in Figure 3. Negative information predominantly occupies more than half of the Attention Map across various layers. It indicates that negative information plays a crucial role in downstream tasks The results of these experiments can also help validate the efficacy of key components within the LoRaDA-A approach, including: Negative information, Low-rank attention maps, and the Position embedding.

A.2.5 Hyperparameters Analysis.

In this section, we conduct an independent study on the hyperparameters within LMAM, including

Table 13: Detailed experimental results of the ablation study on LoRaDA-A using RoBERTa$_{\text{BASE}}$ for evaluating efficient fine-tuning performance on the GLUE benchmark.

Method (# params)	MNLI	SST2	QQP	QNLI	STS-B	MRPC	RTE	CoLA	mean
LoRaDA-A (0.5%)	**88.2**$_{\pm.5}$	**95.9**$_{\pm.4}$	**92.8**$_{\pm.6}$	**93.9**$_{\pm.3}$	**91.0**$_{\pm.2}$	**92.0**$_{\pm.5}$	**83.1**$_{\pm.3}$	**69.9**$_{\pm.3}$	**88.4**
LoRaDA-A$\neg N$ (0.5%)	86.5$_{\pm.4}$	94.9$_{\pm.2}$	90.3$_{\pm.4}$	92.4$_{\pm.3}$	90.2$_{\pm.6}$	89.9$_{\pm.3}$	78.3$_{\pm.5}$	62.1$_{\pm.7}$	85.6
LoRaDA-A$\neg S$ (0.5%)	63.8$_{\pm.4}$	86.6$_{\pm.7}$	83.6$_{\pm.2}$	80.9$_{\pm.6}$	45.2$_{\pm.9}$	72.7$_{\pm.3}$	58.9$_{\pm.3}$	64.1$_{\pm.3}$	69.5
LoRaDA-A$\neg L$ (23.5%)	85.0$_{\pm.5}$	85.5$_{\pm.5}$	87.0$_{\pm.7}$	87.3$_{\pm.8}$	79.8$_{\pm.9}$	82.1$_{\pm.3}$	53.2$_{\pm.7}$	61.2$_{\pm.7}$	77.6
LoRaDA-A$\neg P$ (0.5%)	87.9$_{\pm.1}$	95.5$_{\pm.2}$	91.1$_{\pm.2}$	92.8$_{\pm.3}$	91.1$_{\pm.2}$	91.1$_{\pm.5}$	80.4$_{\pm.3}$	69.1$_{\pm.3}$	87.3

Table 14: Additional ablation studies on the LLaMA-7B model, focusing on its performance on the Commonsense Reasoning benchmark.

Model	Methods	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
LLaMA-7B	LoRaDA-LA	0.09	71.3	84.4	80.9	88.5	83.2	83.5	68.8	79.4	**80.0**
	LoRaDA-LA$\neg N$	0.09	67.0	81.1	77.4	78.4	80.0	79.2	60.2	78.4	75.2
	LoRaDA-LA$\neg S$	0.09	58.2	49.9	33.1	25.7	50.7	26.3	27.3	25.6	37.1
	LoRaDA-LA$\neg L$	0.32	68.4	80.1	77.4	53.3	78.1	80.9	64.3	75.2	72.2
	LoRaDA-LA$\neg P$	0.09	68.3	81.1	78.0	84.3	81.4	80.9	65.5	77.0	77.1

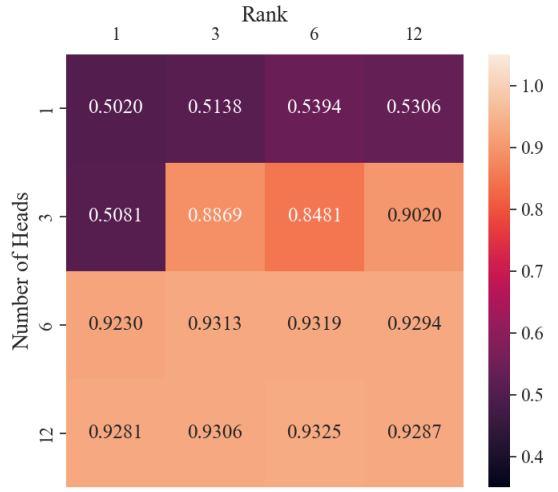


Figure 4: Results on SST2 using RoBERTa$_{\text{BASE}}$ across varying attention heads and ranks within the LMAM framework.

the number of heads and the ranks. As shown in Figure 4, when the number of heads is consistent, particularly when the number of heads is 1, 6, or 12, increasing the rank does not enhance the model's performance significantly. Yet, when the rank number is fixed, increasing the number of attention heads can boost the model's performance to over 90%, even when rank=1. This indicates that the number of heads has a significant impact on the model's performance. Moreover, as the rank increases, fewer additional heads are required to enhance the model's performance to over 90%.

A.3 Core Code

We represent the core code of LMAM as follows:

```
class Negtaive_attention(nn.Module):
```

Table 15: Hyperparameter configurations of LoRADA-A for fine-tuning BART$_{\text{LARGE}}$ and mBART$_{\text{LARGE}}$ with XSum and MT benchmarks. In LMAM, r, n, h_s and s represent the low-rank dimension, fixed token length, number of heads, and scale, respectively. The underline indicates the optimal choices.

Hyper \ Datasets	XSum	MT
LMAM h_s	1,4,16	1,4,16
LMAM r	22,24	32,64
LMAM n	128,256,512	128,256,512
LMAM s	1	1
Adapter Rank r	512,600	800,820
Adapter Scale	4	4
Optimizer	Adam	Adam
Batch size	64 sents	16384 tokens
LR	3e-5,5e-5	5e-5,6e-5
weight decay	0.1	0.01
warmup updates	0	0
label smoothing	0.1	0.1
max source length	512	150
max target length	128	150
weight decay	0.01	0.01
train steps	100K	50K
max grad norm	0.1	1
Where/Adapter	FFN	FFN
Where/LMAM	Encoder	Encoder

```
def __init__():
    super().__init__()
    self.Wa = nn.Parameter(attention_heads,
        num_tokens, attention_rank,
        requires_grad=True)
    self.Wb = nn.Parameter(attention_heads,
    attention_rank,
    num_tokens,requires_grad=True)
    pos = torch.arange(num_tokens)
    omega = torch.arange(num_tokens)
    omega = omega / (num_tokens / 2)
    omega = 1. / 10000**omega
    self.pos_a= omega.unsqueeze(0).unsqueeze
        (2)
    self.pos_b= omega.unsqueeze(0).unsqueeze
```

```
        (0)
    def forward(self, Attn):
        self.wa = self.wa + self.pos_a
        self.wb = self.wb + self.pos_b
        Attn_s = torch.matmul(self.wa, self.wb)*
            scale
        Attn = Attn + Attn_s
        return Attn
```

A.4 Generation Examples

The following showcases our proposed method, LoRaDA-LA, based on LLaMA-7B, generating examples for the BoolQ and OBQA datasets within the Commonsense Reasoning benchmark.

Instruction (BoolQ)

Below is an instruction that describes a task. Write a response that appropriately completes the request. Instruction: Please answer the following question with true or false, question: can a fly lay eggs in your skin? format: true/false Please answer the following question with true or false, question: can a fly lay eggs in your skin? Answer format: true/false

Response

Model	Response
LLaMA-7B	the correct answer is true

Instruction (OBQA)

Below is an instruction that describes a task. Write a response that appropriately completes the request. Instruction: Please choose the correct answer to the question: Some animals use a liquid coming from their skin to adjust to Answer1: cold Answer2: water Answer3: heat Answer4: humidity Answer format: answer1/answer2/answer3/answer4 Please choose the correct answer to the question: Some animals use a liquid coming from their skin to adjust to Answer1: cold Answer2: water Answer3: heat Answer4: humidity Answer format: answer1/answer2/answer3/answer4

Response

Model	Response
LLaMA-7B	the correct answer is answer3

A.5 Environment and Settings

Our method is fine-tuned and evaluated across six benchmarks: GLUE, XSum, MT, Arithmetic Reasoning, Long-sequence and Commonsense Reasoning. These benchmarks span a diverse spectrum of tasks, ranging from straightforward to highly complex, and include classification, generation, translation, and reasoning challenges. Conducting experiments on these tasks allows us to comprehensively demonstrate the effectiveness of our approach. We provide a detailed overview of the experimental configurations for our method through the following tables. Specifically, Table 16 lists the hyperparameters of LoRaDA-A with the $\text{RoBERTa}_{\text{BASE}}$ model on the GLUE benchmark; when using the $\text{DeBERTaV3}_{\text{base}}$ model, the only modification is setting the rank to 2. Huggingface transformers library is employed in our method. Table 15 presents the hyperparameter values for fine-tuning $\text{BART}_{\text{LARGE}}$ and $\text{mBART}_{\text{LARGE}}$ on the XSum and MT benchmarks. For the Commonsense Reasoning benchmark, the experimental settings for LoRaDA-LA and LoRaDA-Q are given in Tables 19 and 17, respectively. Table 17 also reports the configurations used in the Arithmetic Reasoning benchmark, while Table 19 likewise applies to the Long-Sequence benchmark (with sequence length set to 256). For these tasks, we conducted experiments with medium-sized language models (LMs) on NVIDIA RTX 3090 GPUs and large language models (LLMs) on NVIDIA A100 GPUs.

A.6 Theoretical Analysis of LMAM

We propose the following theoretical framework to demonstrate the effectiveness of the LMAM module by analyzing gradient propagation paths. As demonstrated in (Oymak and Soltanolkotabi, 2019), a shorter gradient propagation path results in more effective model optimization. In Proposition 1, we prove that LMAM has shorter gradient propagation path.

Proposition 1 For the original attention matrix $Attn$ and the low-rank attention matrix $Attn_s$, the model propagation depth satisfies: $\text{Depth}(W_Q/W_K) \geqslant \text{Depth}(W_a/W_b)$

Proof: It is evident that the gradient with respect to $\nabla_{W_Q} L = \nabla_{Attn} L \cdot \nabla_Q Attn \cdot \nabla_{W_Q} Q$ follows a longer propagation path compared to the gradient with respect to $\nabla_{W_a} L = \nabla_{Attn_s} L \cdot \nabla_{W_a} (Attn_s) \cdot$.

Table 16: Hyperparameter configurations for LoRaDA-A fine-tuning of RoBERTa$_{\text{BASE}}$ on the GLUE benchmark. In LMAM, r, n, h_s and s represent the low-rank dimension, fixed token length, number of heads, and scale, respectively.

Hyper \ Datasets	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
LMAM h_s	3	3	3	3	3	3	3	3
LMAM r	3	3	3	3	3	3	3	3
LMAM n	512	512	256	512	256	256	512	256
LMAM $1/s$	255	255	255	255	255	255	255	255
Adapter Rank r	16	16	16	16	16	16	16	16
Adapter Scale	2	2	2	2	2	2	2	2
Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Batch size	32	32	32	32	32	32	32	32
LR	3e-4	3e-4	3e-4	5e-4	1e-4	1e-4	7e-4	5e-4
weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
warmup updates	0	0	0	0	0	0	0	0
warmup ratio	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
label smoothing	0	0	0	0	0	0	0	0
maximum sequence length	512	512	512	512	512	512	512	512
weight decay	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Epochs	20,40	20,40	20,40	20,40	20,40	20,40	20,40	20,40
Where/Adapter	FFN	FFN	FFN	FFN	FFN	FFN	FFN	FFN

Table 17: Hyperparameter configurations for LoRaDA-Q on the Commonsense Reasoning benchmark and Arithmetic Reasoning benchmark. In LMAM, r, n, h_s and s represent the low-rank dimension, fixed token length, number of heads, and scale, respectively. The underline indicates the optimal choices. "†" are configurations copied from QuanTA.

Experiment	Hyperparameters	Values
LoRaDA-Q	Batch Size	8
	Optimizer †	AdamW
	Scheduler †	Linear Scheduler
	Learning Rate†	{5e-5, 1e-4}
	Weight Decay †	0
	Dropout†	0
	LMAM r	1,2,4,8
	LMAM n	128
	LMAM h_s	32,40
	N †	4
	d_1-d_2-$\cdots$-d_N †	[16-8-8-4, 16-8-8-5]
	Modules	{(q_proj v_proj), (o_proj up_proj), (o_proj down_proj)}

Table 18: Accuracy comparison of LLaMA-7B using the Adapter, LoRA, and DoRA methods, compared to incorporating LMAM as a plugin in these methods.

Model	Methods	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
LLaMA-7B	Adapter (rank = 2)	0.008	62.2	51.7	55.2	33.2	68.2	34.1	31.2	36.0	46.5
	Adapter+LMAM	0.008	37.8	78.9	76.1	65.5	71.7	79.2	59.6	71.6	**67.6 (+21.1)**
	LoRA (rank = 4)	0.10	2.3	46.1	18.3	19.7	55.2	65.4	51.9	57	39.5
	LoRA+LMAM	0.10	50.3	78.1	77.6	56.1	56.3	79.8	62.9	76.4	**67.2 (+27.7)**
	DoRA (rank = 4)	0.11	51.3	42.2	77.8	25.4	78.8	78.7	62.5	78.6	61.9
	DoRA+LMAM	0.11	69.6	82.1	79.7	85.6	82.7	79.9	62.8	79.9	**77.7(+15.8)**

Table 19: Hyperparameter configurations for LoRaDA-LA applied to LLaMA-7B, LLaMA2-7B, LLaMA3-8B, LLaMA-13B, and LLaMA2-70B models in Commonsense Reasoning benchmark. In LMAM, r, n, h_s and s represent the low-rank dimension, fixed token length, number of heads, and scale, respectively. The underline indicates the optimal choices.

Hyperparameters (LoRaDA)	LLaMA-7B	LLaMA2-7B	LLaMA3-8B	LLaMA-13B	LLaMA2-70B
LMAM h_s	32	32	32	40	64
LMAM r	1,2,4	1,2,4	1,2,4	1,2,4	1,2,4
LMAM n	128	128	128	128	128
LMAM $1/s$	4096*600	4096*600	4096*1200,4096*1500	5120*600	8192*1600, 8192*1200
LoRA Rank r	1,2	1,2	1,2	1,2	1
LoRA α	2,4	2,4	2,4	2,4	2,4
Adapter Rank r	10,14,16,18,20	10,14,15,17,20	2,4,8,12	4,12,14,18,20	2,4,7,10
Adapter Scale	4	4	4	4	4
Dropout	0.05	0.05	0.05	0.05	0.05
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW
LR	1e-4,2e-4	1e-4,2e-4	1e-4,2e-4	1e-4,2e-4	1e-4 ,2e-4
Batch size	4	4	4	3	2
Warmup Steps	100	100	100	100	100
Epochs	3	3	3	3	3
Where/LoRA	O,Down O,Up,Down	O,Down O,Up,Down	O,Down O,Up,Down	O,Down O,Up,Down	O,Down O,Up,Down
Where/Adapter	FFN	FFN	FFN	FFN	FFN