META-LORA: Memory-Efficient Sample Reweighting for Fine-Tuning Large Language Models

Weicheng Li[1], Lixin Zou[1*], Min Tang[2], Qing Yu[1], Wanli Li[3], Chenliang Li[1]

[1]Wuhan University, Wuhan, China, [2]Monash University, Melbourne, Australia
[3]Huazhong Agricultural University, Wuhan, China
{liweicheng, zoulixin, yu_qing, cllee}@whu.edu.cn, min.tang@monash.edu,
liwanli@mail.hzau.edu.cn

Abstract

Supervised fine-tuning (SFT) is widely adopted for tailoring large language models (LLMs) to specific downstream tasks. However, the substantial computational demands of LLMs hinder iterative exploration of fine-tuning datasets and accurate evaluation of individual sample importance. To address this challenge, we introduce META-LORA, a memory-efficient method for automatic sample reweighting. META-LORA learns to reweight fine-tuning samples by minimizing the loss on a small, high-quality validation set through an end-to-end bi-level optimization framework based on meta-learning. To reduce memory usage associated with computing second derivatives, we approximate the bi-level optimization using gradient similarity between training and validation datasets, replacing bi-dimensional gradient similarity with the product of one-dimensional activation states and their corresponding gradients. Further memory optimization is achieved by refining gradient computations, selectively applying them to the low-rank layers of LoRA, which results in as little as 4% additional memory usage. Comprehensive evaluations across benchmark datasets in mathematics, coding, and medical domains demonstrate META-LORA's superior efficacy and efficiency. The source code is available at https://github.com/liweicheng-ai/meta-lora.

1 Introduction

Supervised fine-tuning (SFT) adapts pre-trained large language models (LLMs) to specific downstream tasks by further training them on labeled datasets (Chung et al., 2024; Taori et al., 2023; Wang et al., 2023; Tang et al., 2025; Zhang et al., 2024). However, as the number of model parameters and the size of training corpora increase, the cost of fine-tuning LLMs rises significantly. Moreover, developing or annotating a comprehensive instruction-tuning dataset comparable to those offered by leading companies is extremely costly and challenging, especially given the closed-source nature of LLM training procedures (Achiam et al., 2023). Consequently, iteratively selecting training samples by directly training on available data is less effective, as biased or noisy data are often encountered in practice (Khan et al., 2017; Zhang et al., 2021).

Recent efforts have addressed this issue from two perspectives. The first approach primarily utilizes data similarity for selection. For instance, GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2023) both train a binary classifier to select examples that are similar to formal text from Wikipedia. Xie et al. (2023) introduce DSIR, a variant of Moore-Lewis Selection (Moore and Lewis, 2010), which selects data with hashed n-gram features similar to those of validation samples. Similarly, Yao et al. (2022) propose a retrieval-based method (BM25) for efficient task-relevant data selection. Although these methods can be applied before training without adding any computational burden during the training phase, their reliance on textual similarity does not guarantee accurate control over data distribution and fails to precisely assign weights to the training data.

An alternative approach is to optimize sample weights through end-to-end gradient descent. The core idea is to assess each training sample's contribution by comparing the model's performance with and without that sample, leveraging the automatic optimization capabilities of deep learning tools. Consequently, a series of representative gradient-based data valuation methods, such as influence functions (Koh and Liang, 2017; Park et al., 2023; Schioppa et al., 2022; Pruthi et al., 2020; Hanawa et al., 2021), have been proposed. Although these methods can precisely calculate sample weights in an end-to-end manner, the need to compute second derivatives introduces significant memory over-

*Corresponding author.

Proceedings of the 31st International Conference on Computational Linguistics, pages 8504–8517
January 19–24, 2025.

head, limiting their applications in LLMs.

To address this, we propose META-LORA, which reweights fine-tuning samples in an end-to-end meta-learning manner while optimizing memory utilization to closely match that of naive training methods. Specifically, we weight the training samples by minimizing their post-update validation loss on a small, high-quality validation set, which is essentially a bi-level optimization process. To avoid computing second derivatives, we approximate the bi-level optimization by manually deriving sample weights, leveraging the gradient similarity between training and validation samples. To reduce memory usage, we replace the gradient similarity of bi-dimensional parameters with the product of one-dimensional activation states and their corresponding gradients. Additionally, to further decrease memory consumption, we approximated the weight by only selecting the low-rank layers of LoRA modules (Hu et al., 2022), which use low-rank approximation on specific weight matrices to enable efficient fine-tuning. Finally, with only a 4% increase in memory usage, we can effectively select highly representative samples for fine-tuning. Extensive experiments conducted on downstream benchmark tasks in mathematics, coding, and medical domains validate the effectiveness of the proposed method. In summary, the contributions are as follows:

- We propose META-LORA, an approximate end-to-end learning to reweight strategy based on approximated gradient similarity, which automatically reweights training samples.
- By manually deriving sample weights, we approximate the weights with gradient similarity on the low-rank layers of LoRA between the train and validation. To our knowledge, this is the first work to decompose gradient similarity for sample weighting. This approach results in just a 4% increase in memory consumption while achieving up to a 5% improvement in performance.
- We conducted a series of experiments across various downstream tasks, including mathematical, coding, and medical domains. The results show that our method can up-weight beneficial training samples, leading to superior downstream performance compared to direct fine-tuning and other baseline methods.

2 Preliminaries

This section covers supervised fine-tuning for large language models, the meta-learning approach to sample reweighting, and the use of Low-Rank Adaptation (LoRA) to enhance memory efficiency.

2.1 Supervised Fine-Tuning

Our research focuses on supervised fine-tuning of large language models to adapt pre-trained models for specific tasks or domains. Given a training dataset $\{(X_i, Y_i)\}_{i=1}^{N}$, where each X_i is an input sequence (e.g., instructions or text) and Y_i is the corresponding target output, both represented as token sequences $x_{1:v} = (x_1, \ldots, x_v)$ and $y_{1:k} = (y_1, \ldots, y_k)$. The model predicts the probability $P(y_k \mid x_{1:v}, y_{1:k-1})$ for each token y_k. We train the model by minimizing the cross-entropy loss:

$$\ell_i(\theta) = \ell(X_i, Y_i, \theta) = -\frac{1}{K}\sum_{k=1}^{K} \log P(y_k \mid x_{1:v}, y_{1:k-1}; \theta),$$

where K is the sequence length of the output and θ represents the model parameters.

2.2 Sample Reweighting with Meta Learning

Meta-learning, or *learning to learn*, aims to develop models that quickly adapt to new tasks using limited high-quality data (Finn et al., 2017). To select high-quality samples, we employ a meta-learning approach that optimizes the sample distribution that best minimizes the validation loss. Suppose we have a small, high-quality validation set $\{(X_j^v, Y_j^v)\}_{j=1}^{M}$, where $M \ll N$, which we use to reweight each training sample in a meta-learning manner. Specifically, we aim to minimize the weighted loss as

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} w_i \ell_i(\theta), \tag{1}$$

where $w_i i = 1^N$ are the weights assigned to each training instance. Viewing these weights as hyper-parameters, we optimize them by minimizing the validation loss:

$$w^* = \arg\min_{w, w\geq 0} \frac{1}{M}\sum_{j=1}^{M} \ell_j^v(\theta^*(w)) \tag{2}$$

$$= \arg\min_{w, w\geq 0} \frac{1}{M}\sum_{j=1}^{M} \ell(X_j^v, Y_j^v, \theta^*(w)). \tag{3}$$

Optimization with Automatic Differentiation To alleviate the computational burden of nested optimization loops, we adopt a single-step online

approximation. At each step t, we sample mini-batch of training samples $\{(X_i, Y_i)\}_{i=1}^{n}$ and validation samples $\{(X_j^v, Y_j^v)\}_{j=1}^{m}$, where $n \ll N$ and $m \ll M$. We perturb its training loss by ϵ_i, aiming to reweight each training sample before calculating the parameter change:

$$\ell_{i,\epsilon}(\theta) = \epsilon_i \ell_i(\theta). \quad (4)$$

Using vanilla SGD, the parameters are updated as:

$$\theta_{t+1}(\epsilon) = \theta_t - \alpha \nabla \sum_{i=1}^{n} \ell_{i,\epsilon}(\theta)\Big|_{\theta=\theta_t}, \quad (5)$$

where α denotes the step size. We then perform a gradient descent step *w.r.t.* ϵ_i using the mini-batch of validation samples as:

$$u_{i,t} = -\beta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^{m} \ell_j^v(\theta_{t+1}(\epsilon))\Big|_{\epsilon_{i,t}=0}, \quad (6)$$

with β as the descent step size for ϵ. The final non-negative, normalized weights are:

$$\tilde{w}_{i,t} = \max(u_{i,t}, 0), \quad (7)$$

$$w_{i,t} = \frac{\tilde{w}_{i,t}}{(\sum_j \tilde{w}_{j,t}) + \delta(\sum_j \tilde{w}_{j,t})}, \quad (8)$$

where $\delta(a) = 1$ if $a = 0$, otherwise it is set to 0. $w_{i,t}$ is the final weight assigned to the i-th training sample at step t. Further details and memory usage analysis are provided in Appendix B.

2.3 Low-Rank Adaptation in SFT

To reduce memory usage during fine-tuning, Low-Rank Adaptation (LoRA) (Hu et al., 2022) has been widely adopted. LoRA significantly reduces the number of trainable parameters, by factors ranging from 100 to 10,000, through learning pairs of rank-decomposition matrices while keeping the original weights frozen. Specifically, LoRA approximates the changes of parameters $\Delta W \in \mathbb{R}^{d_1 \times d_2}$ as the product of two low-rank matrices, $A \in \mathbb{R}^{r \times d_2}$ and $B \in \mathbb{R}^{d_1 \times r}$, where $r \ll \min(d_1, d_2)$. After fine-tuning, the matrices A and B are multiplied, and the resulting product is incorporated into the original parameters W as following:

$$W \leftarrow W + \Delta W = W + BA.$$

3 Methodology

To optimize memory utilization, this section first provides a detailed analysis of the meta-learning process and our strategy for avoiding the computation of second derivatives. Subsequently, we compute approximate gradient similarity using the low-rank layers of the LoRA modules. The overall architecture of META-LORA is shown in Figure 1.

3.1 Analysis of Meta Learning

This section derives the weights of the training samples $\{w_i\}_{i=1}^{n}$ by analysing the gradient computation in the meta-learning context.

For clarity and simplicity, we first decompose the gradient concerning the l-th layer parameters θ_l as follows. Let z_l and $\tilde{z}_l$ denote the input and output associated with the parameters θ_l, respectively, such that $\tilde{z}_l = z_l \theta_l^\top$. It is important to note that z_l and $\tilde{z}_l$ are not modeled as the input and output of an entire Feed-Forward Network (FFN) block or a Self-Attention block in a transformer. Instead, they correspond to the input and output related to each individual trainable weight matrix. During back-propagation, let g_l represent the gradients of the loss with respect to $\tilde{z}_l$. The gradients with respect to θ_l can then be obtained by following the chain rule for gradient calculation as

$$\frac{\partial \ell_i(\theta)}{\partial \theta_l} \propto z_{i,l} g_{i,l}^\top. \quad (9)$$

Detailed derivations are provided in Appendix A.

With the gradient decomposition, the gradient of validation loss *w.r.t.* ϵ_i at step t (denoted as $\epsilon_{i,t}$) can be derivated as follows:

$$\begin{aligned}
&\frac{\partial}{\partial \epsilon_{i,t}} \mathbb{E}\Big[\ell^v(\theta_{t+1}(\epsilon))|_{\epsilon_{i,t}=0}\Big] \\
&= \frac{1}{m} \sum_{j=1}^{m} \frac{\partial}{\partial \epsilon_{i,t}} \ell_j^v(\theta_{t+1}(\epsilon))\Big|_{\epsilon_{i,t}=0} && (10)\\
&= \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial \theta_{t+1}(\epsilon_{i,t})}{\partial \epsilon_{i,t}}\Big|_{\epsilon_{i,t}=0} && (11)\\
&\propto -\frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial \ell_i(\theta)}{\partial \theta}\Big|_{\theta=\theta_t} && (12)\\
&= -\frac{1}{m} \sum_{j=1}^{m} \sum_{l=1}^{L} \frac{\partial \ell_j^v(\theta)}{\partial \theta_l}\Big|_{\theta_l=\theta_{l,t}}^{\top} \frac{\partial \ell_i(\theta)}{\partial \theta_l}\Big|_{\theta_l=\theta_{l,t}} && (13)\\
&\propto -\frac{1}{m} \sum_{j=1}^{m} \sum_{l=1}^{L} \mathrm{vec}(z_{j,l}^v g_{j,l}^{v\top})^\top \mathrm{vec}(z_{i,l} g_{i,l}^\top), && (14)
\end{aligned}$$

where ℓ^v denotes the validation loss, m denotes the mini-batch size of validation samples, and L denotes the total number of trainable modules. We derive Equation 11 following the chain rule. The derivation of $\frac{\partial \theta_{t+1}(\epsilon_{i,t})}{\partial \epsilon_{i,t}}\Big|_{\epsilon_{i,t}=0} \propto \frac{\partial \ell_i(\theta)}{\partial \theta}|_{\theta=\theta_t}$ in Equation 12 is based on the approximate computation of second-order partial derivatives, which's detailed in Appendix B. The Equation 14 follows the rule in Equation 9.

Memory-usage Analysis According to the above derivation, we only need to compute $\mathrm{vec}(z^v_{j,l} {g^v_{j,l}}^\top)$ and $\mathrm{vec}(z_{i,l} g_{i,l}^\top)$ for each trainable module, which can then be multiplied to obtain the gradient similarity, without actually calculating second derivatives. However, the magnitudes of these vectors are comparable to the sizes of the corresponding bi-dimensional parameters, resulting in twice the memory usage of computing a gradient, thus making it rather expensive to be adopted in LLMs.

3.2 Reweighting as Gradient Similarity

We approximate gradient similarity by decomposing the original gradient into one-dimensional activation and gradient products. Additionally, to compute this similarity with minimal overhead, we utilize the low-rank layers in LoRA modules instead of dense layers.

Gradient Decomposition Considering the high cost of computing and saving the gradient similarity, we perform further derivations based on Equation 14 to convert the calculation of gradient similarity into product of one-dimensional activation states and their corresponding gradients (Equation 17), which is derived according to the commutative property of multiplication:

$$\frac{\partial}{\partial \epsilon_{i,t}} \mathbb{E}\Big[\ell^v(\theta_{t+1}(\epsilon))|_{\epsilon_{i,t}=0}\Big] \tag{15}$$

$$\propto -\frac{1}{m}\sum_{j=1}^{m}\sum_{l=1}^{L} \mathrm{vec}(z^v_{j,l} {g^v_{j,l}}^\top)^\top \mathrm{vec}(z_{i,l} g_{i,l}^\top) \tag{16}$$

$$= -\frac{1}{m}\sum_{j=1}^{m}\sum_{l=1}^{L} ({z^v_{j,l}}^\top z_{i,l})({g^v_{j,l}}^\top g_{i,l}). \tag{17}$$

Through this conversion, we only need to save the one-dimensional activations (i.e., $z^v_{j,l}$ and $z_{i,l}$) and gradients (i.e., $g^v_{j,l}$ and $g_{i,l}$) in memory for gradient similarity's calculation, instead of the bi-dimensional gradients, leading to a significant reduction in memory utilization.

Memory-usage analysis The above decomposition of parameter gradients leads to substantial memory savings in model training. Consider a trainable parameter with dimensions $d_1 \times d_2$. Calculating its bi-dimensional gradient similarity incurs a memory overhead proportional to $2 \times d_1 \times d_2$. By decomposing it into the product of one-dimensional activation and gradient, the memory overhead is reduced to being proportional to $2 \times (d_1 + d_2)$, achieving an approximate $\frac{d_1 d_2}{d_1 + d_2}$-fold reduction in memory usage. Similarly, the memory usage when applying LoRA is reduced from $2 \times r \times (d_1 + d_2)$ to $2 \times (2r + d_1 + d_2)$, resulting in an approximate r-fold reduction in memory usage. It should be noted that the actual training parameters used in META-LORA are the same as those used in naive fine-tuning, and the additional memory overhead arises from the one-dimensional activations and gradients of training and validation samples used to approximate sample weights.

Low-Rank Approximation of Gradients The proposed approximated gradient similarity still presents a critical issue: LLMs consist of extensive trainable modules, making it overloaded to store all activations and gradients even when applying LoRA. Additionally, the activations on these modules' dense layers may become unstable or redundant during training, particularly in the lower layers. Therefore, we approximates the gradient similarity by utilizing the activations and gradients on low-rank layers (with dimensions of r) of LoRA modules, instead of those on the dense layers:

$$\begin{aligned} &\frac{\partial}{\partial \epsilon_{i,t}} \mathbb{E}\Big[\ell^v(\theta_{t+1}(\epsilon))|_{\epsilon_{i,t}=0}\Big] \\ &\propto -\frac{1}{m}\sum_{j=1}^{m}\sum_{l=1}^{L} ({z^v_{j,l}}^\top z_{i,l})({g^v_{j,l}}^\top g_{i,l}) \\ &\approx -\frac{1}{m}\sum_{j=1}^{m}\sum_{\hat{l}=1}^{\hat{L}} ({z^v_{j,\hat{l}}}^\top z_{i,\hat{l}})({g^v_{j,\hat{l}}}^\top g_{i,\hat{l}}), \end{aligned} \tag{18}$$

where $\hat{L}$ denotes the total number of LoRA modules' low-rank layers in the model.

Memory-usage analysis Through low-rank approximation, we achieve a memory utilization that is closely match that of the naive training. When applying LoRA, calculating approximated gradient similarity on low-rank layers only results in additional memory usage proportional to $2 \times (r + r) = 4r$, leading to a nearly $(d_1 + d_2)$-fold memory reduction compared to calculating bi-dimensional gradient similarity.

4 Experiment

To verify META-LORA's effectiveness, this section presents a series of experiments and detailed analyses to guide its usage.

4.1 Experimental Setting

Dataset We employ a variety of datasets across mathematics, coding, and medical tasks to assess

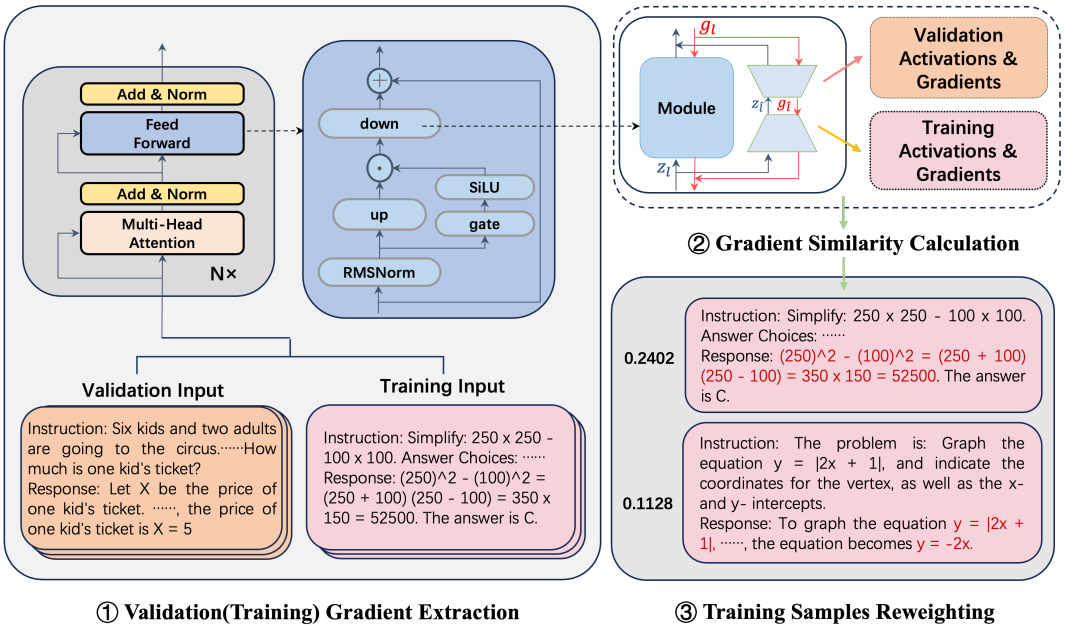

Figure 1: The META-LORA performs sample reweighting based on the approximated gradient similarity between training and validation samples. To do this, we first decompose the gradient of all trainable weight matrices (taking the down-projection module of FFN block as an example in the figure) into product of one-dimensional activations (z_l) and corresponding output gradient (g_l), and then use them to approximate the gradient similarity. To further reduce memory overhead, we only utilize the activations ($z_{\hat{l}}$) and gradients ($g_{\hat{l}}$) on LoRA's low-rank layers for calculation. An example of reweighting is shown in the bottom right, where important key terms are manually highlighted in red.

the effectiveness of META-LORA. In the mathematics domain, we utilize **MathInstruct**(Yue et al., 2023), an instruction tuning dataset covering various mathematical fields, and **GSM8K**(Cobbe et al., 2021), containing diverse grade-school level math word problems. For coding tasks, we use **OSS-Instruct**(Wei et al., 2023), comprising extensive coding problem-solution pairs, **Code-Alpaca**(Chaudhary, 2023), an instructional dataset focusing on Python samples, and **MBPP+** (Liu et al., 2024), consisting of Python programming problems solvable by entry-level programmers. In the medical field, we employ **PubMedQA**(Jin et al., 2019), a question-answering dataset with biomedical QA instances, **ICliniq-chatgpt**(Li et al., 2023), **MedQA** (Jin et al., 2021), a multiple-choice OpenQA dataset from professional medical board exams, and **MedMCQA**(Pal et al., 2022), replicating real-world medical exams. Table 1 presents the partitioning and statistics of all the datasets used for the experiments.

Evaluation Metric For all the benchmarks, we use the Accuracy (ACC) to measure the degree of precise matching between the model's outputs and standard answers.

Task	Dataset	Dataset partitioning			
		Train	Validation	Test	Total
Math	MathInstruct	262,000	0	0	262,000
	GSM8K	0	7,473	1,319	8,792
Coding	OSS-Instruct	75,200	0	0	752,000
	Code-Alpaca	0	20,000	0	20,000
	MBPP+	0	0	378	378
Medical	PubMedQA	273,000	0	0	273,000
	ICliniq-chatgpt	0	7,320	0	7,320
	MedQA	0	0	1,273	1,273
	MedMCQA	0	0	6,150	6,150

Table 1: The statistics and partition of all the datasets.

Baseline Methods To demonstrate the effectiveness of META-LORA, we compare it with both classical methods and recent state-of-the-art techniques: **(1)** The **Base Model** before fine-tuning. **(2) Fine-tuning** the model on $\mathcal{D}_{\text{train}}$. **(3) Binary Classification** (Brown et al., 2020) trains a classifier to select data sharing similar characteristics with validation samples. **(4) MIX** training and validation samples in a certain ratio. **(5)** Using **BM25** (Robertson et al., 2009; Yao et al., 2022) scores to assign weights to training samples based on textual statistical features (i.e., TF-IDF). **(6) DSIR** (Xie et al., 2023) reweights training samples based on the similarity of hashed n-gram features with validation samples.

Experimental Protocols We employ Baichuan2-7B-Base (Yang et al., 2023) as the backbone LLM for both math and coding experiments, while selecting OPT-1.3B (Zhang et al., 2022) as the backbone model for medical experiments. **The reason for this choice is that the Baichuan2-7B model already performs exceptionally well on the medical dataset, making further fine-tuning less impactful in enhancing its performance.** All fine-tuning processes are conducted using LoRA (Hu et al., 2022), with the rank r of the LoRA modules set to 32. To minimize GPU memory overhead, we employ mixed-precision with `bfloat16` (Wang and Kanwar, 2019) for both fine-tuning and inference. In the math and coding experiments, we set the maximum length of each sample to 400, as the input and output fields of related datasets are generally short. For the medical experiments, we increase the maximum length to 2048, since the context and response fields of PubMedQA (used as the training dataset) are typically long. During fine-tuning, we use a learning rate scheduler with cosine decay, setting the peak learning rate to 5×10^{-4}. All the models are optimized by the Adam optimizer (Kingma, 2014), with parameters β_1 and β_2 set to 0.9 and 0.95 respectively. To ensure generality, we perform fine-tuning with a batch size of 4 across all experiments on a single RTX3090 (24GB) GPU node.

4.2 Main Results

The main results of META-LORA across different downstream tasks and its comparison with baselines are presented in Table 2. From the table, we have following observations: **(1) META-LORA is the best practice in boosting downstream performance.** In Table 2, META-LORA shows an improvement over direct fine-tunning across all downstream tasks, with increases of 2.40%, 5.25%, 1.73% and 1.84% in accuracy respectively. Compared with the best baseline in each task, META-LORA achieves an improvement in accuracy of 1.60%, 0.75%, 1.41% and 0.19% respectively. **(2) Training a binary classifier is ineffective in improving performance.** In Table 2, Binary Classification realizes only a slight enhancement compared to direct fine-tuning in most cases ($< 0.5\%$), primarily due to the inadequate features it utilizes. **(3) Mixing training and validation samples can sometimes lead to better performance compared to direct fine-tuning.** In Table 2, Mixing training and validation samples in a 10:1 ratio achieves an improvement in accuracy of 4.25% and 1.39% compared to direct fine-tuning in MBPP+ and MedMCQA task respectively. However, it may sometimes lead to performance degradation (e.g., GSM8K and MedQA), probably due to the large discrepancy between training and validation samples leading to unstable model learning.

4.3 Analysis Experiments

We provide analyses of META-LORA from four perspectives. First, we analyze the influence of low-rank approximation, examining whether it would impair model's downstream performance. Second, we present a memory-usage analysis from both theoretical and experimental perspectives. Third, we conduct an ablation study to explore the influence of different modules used for gradient similarity computation. Lastly, we provide an analysis on the key hyperparameters involved in META-LORA.

Influence of Low-Rank Approximation To explore whether the low-rank approximation would impair downstream performance, we conduct experiments using activations and gradients on dense layers in META-LORA. Specifically, we adjust META-LORA with the following variants: **(1)** META-LORA (L): using the low-rank layers of LoRA modules for computing approximated gradient similarity; **(2)** META-LORA (D): using the dense layers of all trainable modules for computing gradient similarity. We report accuracy on all downstream tasks, as detailed in the previous evaluation setting. As shown in Figure 2, **the low-rank approximation of gradients leads to substantial memory savings while maintaining performance**. It even outperforms the results obtained by utilizing the dense-layer gradients in most cases in empirical results due to the stability of the low-rank layers.

Memory-Usage Analysis To explore the memory usage of META-LORA, we begin with an example quantitative analysis of one of the FFN down-projection modules in the Baichuan2-7B model (dimensions 4096×11008). Using LoRA, the weight matrix is decomposed into two low-rank matrices of sizes 4096×32 and 32×11008. For gradient similarity between training and validation samples, we apply a low-rank approximation by multiplying one-dimensional low-rank activations (32×1) and gradients (32×1), achieving a 7,552-fold reduction in memory usage.

Additionally, Table 3 reports GPU memory usage on an NVIDIA RTX 3090 with mixed precision

Downstream Task Metric	GSM8K ACC(%)	MBPP+ ACC(%)	MedQA ACC(%)	MedMCQA ACC(%)
Base Model	23.0	19.5	25.77	27.95
Fine-tuning	27.5	25.25	26.55	28.16
Binary Classification	27.6	25.5	26.30	28.55
MIX	26.4	29.5	26.39	29.55
BM25	28.1	29.75	26.87	29.81
DSIR	28.3	29.5	26.55	29.62
META-LORA	**29.9***	**30.5***	**28.28***	**30.00***

Table 2: Results of META-LORA across all the downstream tasks and its comparison with baselines. **Bold** numbers denote the best results. "*" indicates significant improvements over the best baselines with p-value < 0.05.

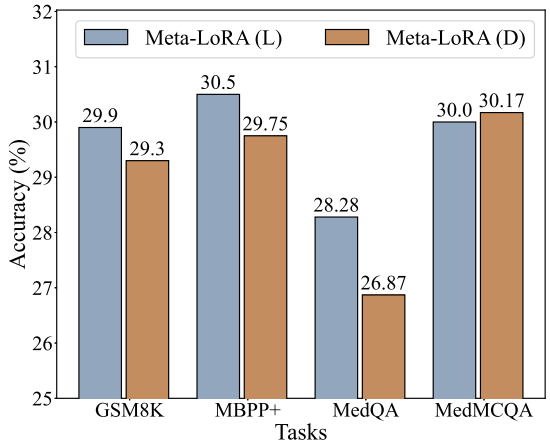

Figure 2: Results of META-LORA on all downstream tasks using gradients on low-rank layers (denoted as L in parentheses) or dense layers (denoted as D in parentheses).

(`bfloat16`), showing the maximum memory consumption during LoRA fine-tuning. The results indicate that **META-LORA is highly memory efficient with low-rank gradient approximation, increasing memory usage by only 4%.**

Ablation Study We explore META-LORA's performance against different modules used for computing gradient similarity. In particular, we adjust META-LORA with the following variants: **(1)** META-LORA (all): using the low-rank layers on all LoRA modules for computing approximated gradient similarity; **(2)** META-LORA (w/o MLP): using the low-rank layers on Attention LoRA modules for computing approximated gradient similarity; **(3)** META-LORA (w/o Attn.): using the low-rank layers on MLP LoRA modules for computing approximated gradient similarity. We report accuracy on all downstream tasks in Table 4. Our observation is that **both MLP and Attention modules are crucial for calculating gradient similarity as the variant of META-LORA (all) yields optimal results in all the scenario.** Due to this fact, we default to computing gradient similarity on all modules when applying META-LORA in the aforementioned experiments. Additionally, we find that the importance of MLP and Attention modules varies for different tasks. Specifically, the variant of META-LORA (w/o Attn.) performs better than META-LORA (w/o MLP) in both GSM8K and MBPP+ task, but in the MedQA and MedMCQA task, the latter variant performs better. This may be attributed to different base model adopted and the model's focus on learning different weight matrices for different tasks.

Hyperparameter Analysis The key hyperparameter involved in META-LORA is the ratio of training samples to validation samples. In the aforementioned experiment, we adopt a ratio of 10:1, and we further explore how its change will affect performance by shifting it to 5:1 and 50:1. Figure 3 presents the results, which indicate that **the ratio of training to validation samples does not significantly affect the META-LORA's performance.** Particularly, the outcomes are very similar when the ratio is set to 5:1 and 50:1, demonstrating the algorithm's robustness even with a small number of validation samples. Therefore, we adopt a default ratio of 10:1 in META-LORA unless otherwise specified, as this ratio most frequently yields optimal results.

Additional analyses We provide more analyses concerning bad cases and META-LORA's scalability to multi-GPU or multi-node environments in Appendix C. In brief, with respect to the bad case, an inappropriate choice of the validation set can degrade META-LORA's performance. As

Model	Dataset	LoRA Fine-tuning	META-LORA	Δ
Baichuan2-7B	GSM8K	20.30 GB	20.99 GB	↑ 3.4%
Baichuan2-7B	MBPP+	19.79 GB	20.19 GB	↑ 2.0%
OPT-1.3B	MedQA & MedMCQA	7,156 MB	7,435 MB	↑ 3.9%

Table 3: GPU memory usage analyses for META-LORA. Fine-tuning represents directly fine-tuning on the training set. META-LORA denotes fine-tuning using low-rank approximated gradient similarity for sample reweighting. Δ denotes the increase in total memory usage.

Downstream Task Metric	GSM8K ACC(%)	MBPP+ ACC(%)	MedQA ACC(%)	MedMCQA ACC(%)
META-LORA (all)	**29.9**	**30.5**	**28.28**	**30.00**
META-LORA (w/o MLP)	29.1	27.5	27.65	29.84
META-LORA (w/o Attn.)	29.7	29.25	27.49	29.79

Table 4: Performance comparison of META-LORA's different variants. **Bold** numbers denote the best results.

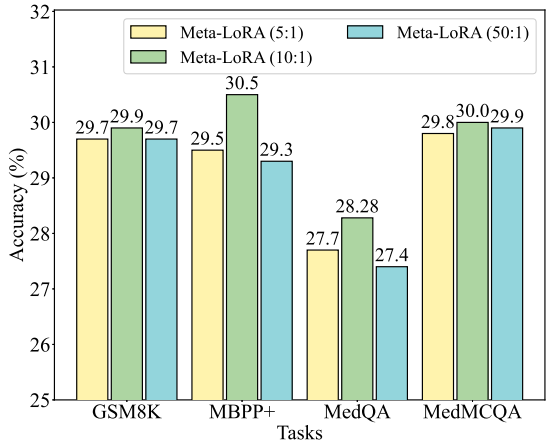

Figure 3: Performance comparison of META-LORA using different ratios. The ratios in the parentheses represent the proportion of training samples to validation samples used in META-LORA.

for its applicability in multi-GPU environments, META-LORA can be adapted to distributed settings through the incorporation of some distributed data processing techniques and synchronization mechanisms, like DeepSpeed (Rasley et al., 2020), ensuring its scalability to distributed systems.

5 Related Work

Traditional Data Selection Data selection or reweighting has been extensively studied in traditional machine learning tasks(Moore and Lewis, 2010; Ren et al., 2018; Guo et al., 2021; Yao et al., 2022; Luo et al., 2024; Xie et al., 2024b). Moore-Lewis Selection (Moore and Lewis, 2010; Axelrod, 2017; Feng et al., 2022) utilizes cross-entropy difference between LMs trained on training and target dataset to score training samples, and Xie et al. further proposes DSIR, a variant of Moore-Lewis Selection, to use hashed n-gram features to efficiently perform data weighting. In addition, Yao et al. utilize a retrieval-based method (BM25) to select task-relevant training data. These methods can be easily applied to LLMs for they do not use gradient information. Furthermore, Ren et al. proposes a gradient-based method to reweight training samples in a meta-learning manner, but the second derivative calculation involved hinders it from being applied to LLMs. In contrast, our proposed META-LORA convert gradient similarity to the low-rank layer-wise product of one-dimensional activation states and corresponding gradients, making it applicable to billion-scale LLMs.

Data Selection for LLM The selection of pretraining data is critical for improving LLM's performance (Brown et al., 2020; Gururangan et al., 2020; Zou et al., 2021; Hoffmann et al., 2022; Xie et al., 2024a, 2025). Many existing works rely on data similarity to assess training data's quality, e.g., GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2023) both train a binary classifier to select training samples that are similar to formal text from Wikipedia (Gao et al., 2020). However, training a classifier demands extensive data, and binary classification does not guarantee consistency with the target distribution (Xie et al., 2023), which makes it suboptimal in fine-tuning scenarios.

Data Valuation and Influence Functions Data Valuation, also referred to as Data Attribution (Park et al., 2023), indicates measuring the contribution

of training data on model's performance (Ghorbani and Zou, 2019; Jia et al., 2019). Most data valuation methods, like Data Shapley (Ghorbani and Zou, 2019; Kwon and Zou, 2021) or Datamodels (Ilyas et al., 2022), evaluate a training instance's value by comparing the model's performance with or without this sample in training set, which requires multiple repeated training, leading to significant scalability challenges. To alleviate this issue, a series of representative gradient-based data valuation methods, i.e., influence functions (Koh and Liang, 2017; Park et al., 2023; Schioppa et al., 2022; Pruthi et al., 2020; Hanawa et al., 2021), have been proposed. However, these works primarily focus on small models and can hardly be applied to LLMs due to the prohibitive computational and memory overhead caused by second derivatives. Indeed, we find that running these methods with billion-scale models all results in CUDA OOM errors on NVIDIA RTX 3090 GPUs. In contrast, our proposed META-LORA can be applied to billion-scale LLMs with minimal additional overhead.

6 Conclusion

We propose META-LORA, a memory efficient sample reweighting method for LLMs' fine-tuning. It is achieved by manually computing the gradient similarity between training and validation samples, without calculating second derivatives. Further, we decompose bi-dimensional gradients into product of one-dimensional activations and corresponding gradients, and calculating approximated gradient similarity on LoRA's low-rank layers. Experimental results show that our approach yield better downstream performance than direct fine-tuning on training set and other baselines, with minimal additional memory overhead.

7 Limitations

The primary limitation of our proposed method is its tight integration with LoRA, which may restrict its flexibility or general applicability to other architectures that do not use LoRA. Nevertheless, we can calculate the approximated gradient similarity without LoRA (see the first half of Section 3.2), since the decomposition can be applied to other architectures as well, and we simply combine it with LoRA to further conserve memory and enhance the model's applicability.

Acknowledgments

We express our sincere gratitude for the financial support provided by the National Natural Science Foundation of China (NO. 62302345 and NO. U23A20305), the Natural Science Foundation of Hubei Province (NO. 2023AFB192 and NO.2023BAB160), the CCF-ALIMAMA TECH Kangaroo Fund (NO. CCF-ALIMAMA OF 2024009), the Xiaomi Young Scholar Program, and the Natural Science Foundation of Wuhan (NO. 2024050702030136).

References

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

Amittai Axelrod. 2017. Cynical selection of language model training data. *arXiv preprint arXiv:1709.02279*.

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.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

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

Mathieu Dagréou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. 2022. A framework for bilevel optimization that enables stochastic and global variance reduction algorithms. *Advances in Neural Information Processing Systems*, 35:26698–26710.

Yukun Feng, Patrick Xia, Benjamin Van Durme, and João Sedoc. 2022. Automatic document selection for efficient encoder pretraining. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9522–9530.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Saeed Ghadimi and Mengdi Wang. 2018. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*.

Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pages 2242–2251. PMLR.

Siyuan Guo, Lixin Zou, Yiding Liu, Wenwen Ye, Suqi Cheng, Shuaiqiang Wang, Hechang Chen, Dawei Yin, and Yi Chang. 2021. Enhanced doubly robust learning for debiasing post-click conversion rate estimation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–284.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*.

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. 2021. Evaluation of similarity-based explanations. In *9th International Conference on Learning Representations, ICLR 2021*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35:30016–30030.

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*.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. 2022. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*.

Kaiyi Ji, Junjie Yang, and Yingbin Liang. 2021. Bilevel optimization: Convergence analysis and enhanced design. In *International conference on machine learning*, pages 4882–4892. PMLR.

Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. 2019. Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577.

Salman H Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous A Sohel, and Roberto Togneri. 2017. Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems*, 29(8):3573–3587.

DP Kingma. 2014. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Yongchan Kwon and James Zou. 2021. Beta shapley: a unified and noise-reduced data valuation framework for machine learning. *arXiv preprint arXiv:2110.14049*.

Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, Steve Jiang, and You Zhang. 2023. Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge. *Cureus*, 15(6).

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36.

Dan Luo, Lixin Zou, Qingyao Ai, Zhiyu Chen, Chenliang Li, Dawei Yin, and Brian D Davison. 2024. Unbiased learning-to-rank needs unconfounded propensity estimation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1535–1545.

Robert C Moore and William Lewis. 2010. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 conference short papers*, pages 220–224.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on health, inference, and learning*, pages 248–260. PMLR.

Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Mądry. 2023. Trak: attributing model behavior at scale. In *Proceedings of the 40th International Conference on Machine Learning*, pages 27074–27113.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pages 4334–4343. PMLR.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. 2022. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8179–8186.

Min Tang, Lixin Zou, Shiuan-ni Liang, She Jin, Weiqing Wang, and Shujie Cui. 2025. Chifraud: A long-term web text benchmark for chinese fraud detection. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025)*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7.

Shibo Wang and Pankaj Kanwar. 2019. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 4(1).

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. 2024a. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy S Liang. 2023. Data selection for language models via importance resampling. *Advances in Neural Information Processing Systems*, 36:34201–34227.

Tianchi Xie, Jiangning Zhu, Guozu Ma, Minzhi Lin, Wei Chen, Weikai Yang, and Shixia Liu. 2024b. Structural-entropy-based sample selection for efficient and effective learning. *arXiv preprint arXiv:2410.02268*.

Yunfan Xie, Lixin Zou, Dan Luo, Min Tang, Chenliang Li, Liming Dong, and Xiangyang Luo. 2025. Mitigating language confusion through inference-time intervention. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025)*.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

Xingcheng Yao, Yanan Zheng, Xiaocong Yang, and Zhilin Yang. 2022. Nlp from scratch without large-scale pretraining: A simple and efficient framework. In *International Conference on Machine Learning*, pages 25438–25451. PMLR.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Chaoran Zhang, Lixin Zou, Dan Luo, Xiangyang Luo, Zihao Li, Min Tang, and Chenliang Li. 2024. Efficient sparse attention needs adaptive token release. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 14081–14094.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.

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*.

Lixin Zou, Shengqiang Zhang, Hengyi Cai, Dehong Ma, Suqi Cheng, Shuaiqiang Wang, Daiting Shi, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained language model based ranking in baidu search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 4014–4022.

A The Derivation of Gradient Decomposition

In this section, we provide a detailed derivation of the gradient decomposition in backpropagation, utilizing the chain rule. The chain rule, initially proposed in calculus, is used to compute derivatives of composite functions. In the context of neural networks, the gradient of a weight matrix can be calculated by propagating derivatives layer by layer. Consider a neural network where the output of a certain layer is given by

$$\tilde{z} = Wz + b, \tag{19}$$

where W is the weight matrix, z is the input to the current layer (the activation from the previous layer), $\tilde{z}$ is the output of current layer, and b is the bias (which is typically set to be constantly zero in LLMs' fine-tuning). In backpropagation, the gradient of loss function ℓ with respect to the weight matrix W is computed as

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial \tilde{z}} \cdot \frac{\partial \tilde{z}}{\partial W} \tag{20}$$

according to the chain rule. In Equation 20, the term $\frac{\partial \ell}{\partial \tilde{z}}$ represents the gradient of ℓ with respect to the output $\tilde{z}$ of this layer, which is propagated from the next layer during backpropagation. The term $\frac{\partial \tilde{z}}{\partial W}$ is simply the input activation z:

$$\frac{\partial \tilde{z}}{\partial W} = z, \tag{21}$$

which is because the transformation is linear in W in Equation 19. Substituting the results from Equation 20 and Equation 21, we get:

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial \tilde{z}} \cdot z, \tag{22}$$

which shows that the gradient of weight matrix W is the input activation z multiplied by the gradient of the output $\frac{\partial \ell}{\partial \tilde{z}}$.

It's worth noting that Equation 19 also applies to the settings in Section 3.1 because we take each individual trainable weight matrix in LLMs into account. Given the weight matrix θ_l of a trainable module, and let $z_{i,l}$ and $\tilde{z}_{i,l}$ denote the input and output asscociated with θ_l for the i-th training sample, Equation 19 can be rewritten as

$$\tilde{z}_{i,l} = z_{i,l}\theta_l^\top. \tag{23}$$

Therefore, Equation 22 can be expressed as:

$$\frac{\partial \ell_i(\theta)}{\partial \theta_l} \propto z_{i,l} g_{i,l}^\top, \tag{24}$$

where ℓ_i denotes the loss of the i-th training sample, and θ denotes all weight matrices in the model. In this way, the gradients of θ_l is decomposed into product of one-dimensional input activation $z_{i,l}$ and corresponding output gradient $g_{i,l}^\top$.

B The Bi-level Optimization in Meta Learning

In this section, we provide a detailed derivation and memory-usage analysis of the bi-level optimization in meta learning.

In order to circumvent the two nested loops of optimization illustrated in Section 2.2, we adopt an online approximation, following the manner in (Ren et al., 2018). Specifically, we first sample a mini-batch of training samples $\{(X_i, Y_i)\}_{i=1}^n$ at every step t, where $n \ll N$. Then the parameters are adjusted based on the descent direction of the expected loss on this mini-batch. When using vanilla SGD, it can be expressed as follows:

$$\theta_{t+1} = \theta_t - \alpha \nabla \left(\frac{1}{n} \sum_{i=1}^{n} \ell_i(\theta_t) \right), \tag{25}$$

where α denotes the step size. To further assess the impact of each training sample to validation performance, we perturb its loss by ϵ_i:

$$\ell_{i,\epsilon}(\theta) = \epsilon_i \ell_i(\theta), \tag{26}$$

$$\theta_{t+1}(\epsilon) = \theta_t - \alpha \nabla \sum_{i=1}^{n} \ell_{i,\epsilon}(\theta)\Big|_{\theta=\theta_t}, \tag{27}$$

then look for the optimal ϵ^* that minimizes the validation loss ℓ^v at training step t:

$$\epsilon_t^* = \arg\min_{\epsilon} \frac{1}{M} \sum_{i=1}^{M} \ell_i^v(\theta_{t+1}(\epsilon)). \tag{28}$$

To expedite the estimation of w_i at step t, we sample a mini-batch of validation samples $\{(X_i^v, Y_i^v)\}_{i=1}^m$, where $m \ll M$, then execute a single gradient descent step w.r.t. ϵ_t on this mini-batch. Finally, we adjust the output to ensure that they are non-negative and sum up to one:

$$u_{i,t} = -\beta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^{m} \ell_j^v(\theta_{t+1}(\epsilon))\Big|_{\epsilon_{i,t}=0}, \quad (29)$$

$$\tilde{w}_{i,t} = \max(u_{i,t}, 0), \quad (30)$$

$$w_{i,t} = \frac{\tilde{w}_{i,t}}{(\sum_j \tilde{w}_{j,t}) + \delta(\sum_j \tilde{w}_{j,t})}, \quad (31)$$

where β denotes the descent step size on ϵ, and $\delta(a) = 1$ if $a = 0$, otherwise it is set to 0. We proceed with further derivation of the Equation 29:

$$-\beta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^{m} \ell_j^v(\theta_{t+1}(\epsilon))\Big|_{\epsilon_{i,t}=0} \quad (32)$$

$$= -\beta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial \theta_{t+1}(\epsilon_{i,t})}{\partial \epsilon_{i,t}}\Big|_{\epsilon_{i,t}=0} \quad (33)$$

$$= \beta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial(\alpha \nabla \ell_{i,\epsilon}(\theta))}{\partial \epsilon_{i,t}}\Big|_{\theta=\theta_t, \epsilon_{i,t}=0} \quad (34)$$

$$= \alpha\beta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial^2(\epsilon_{i,t} \ell_i(\theta))}{\partial \epsilon_{i,t} \partial \theta}\Big|_{\theta=\theta_t, \epsilon_{i,t}=0} \quad (35)$$

$$\propto \alpha\beta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial \ell_j^v(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}^{\top} \frac{\partial \ell_i(\theta)}{\partial \theta}\Big|_{\theta=\theta_t}. \quad (36)$$

The above process involves calculating second-order mixed partial derivatives (Equation 35), and its result is proportional to the gradient similarity between training and validation samples (Equation 36).

Memory-usage Analysis The above process requires computing and storing Hessian matrix of the function ℓ relatively to θ and ϵ (Equation 35), which consists of all the second-order partial derivatives, resulting in a quadratic growth in memory overhead. In fact, assuming the number of trainable parameters is ϕ, the space and time complexity of calculating full Hessian matrix are $O(\phi^2)$ and $O(\phi^3)$ respectively, which is impractical in LLMs with more than 1 Billion parameters. Therefore, in order to circumvent explicit computation of full Hessian matrix, some works (Ghadimi and Wang, 2018; Ji et al., 2021; Dagréou et al., 2022) have explored computing Hessian-vector products (HVPS) to evaluate the directional variation of gradients along a specific direction, instead of the full Hessian matrix. In the meta-learning schema, we need to compute HVPs in a reverse-over-reverse mode, i.e., conduct backpropagation in the computational graph of gradients, leading to no less than twice the complexity of calculating the gradient. In practice, computing an HVP requires two or three times more memory than computing a gradient, which is quite resource-intensive to be applied to LLMs.

C Additional analyses of META-LORA

In this section, we first take the mathematical task as an example to illustrate the bad case of META-LORA, and explain its underlying cause. Then, we demonstrate the effectiveness of META-LORA when extended to multi-GPU environments.

Bad cases. One of the key factors that influences META-LORA's effectiveness is the choice of validation set, and we find that the inappropriate selection of validation set (or inappropriate choice of input fields in the validation set) would lead to performance degradation. Taking the mathematical task as an example, we replace the validation set for mathematical training with **Ape210K** [1], while other settings are consistent with those in Section 4.1. **Ape210K** is a large-scale dataset of math word problems where each problem includes a question, an equation (the calculation formula), and the final answer, as shown in Table 5.

Question	A school has 75 basketballs and 35 volleyballs. If these balls are distributed evenly among 5 classes, how many balls does each class receive?
Equation	x = (75 + 35) / 5
Answer	22

Table 5: Data fields of Ape210K.

We first choose 'Question' and 'Equation' as input fields, where the former serves as instructions and the latter as responses in the instruction tuning scenarios, and then select 'Question' and 'Answer' respectively for comparison. Table 6 shows the performance comparison of META-LORA using different datasets as the validation set. It can be seen that using Ape210K-ans as the validation set degrades META-LORA's effectiveness, resulting in performance even worse than direct fine-tuning, which may be because the selected 'Answer' field in Ape210K contains only a single answer without any reasoning information, hindering the model's learning, and thus leads to bad results. On the other hand, the performance using Ape210K-equ as the validation set is better than direct fine-tuning, but worse than using gsm8k as the validation set, which may be because the "Equation" field in Ape210K contains insufficient reasoning processes and lacks textual explanations.

[1] https://github.com/Chenny0808/ape210k

Downstream Task Metric	GSM8K ACC(%)
Fine-tuning	27.5
META-LORA (gsm8k)	**29.9**
META-LORA (Ape210K-equ)	28.5
META-LORA (Ape210K-ans)	*26.3*

Table 6: Performance comparison of META-LORA using different datasets (shown in the parentheses) as the validation set. Ape201K-equ denotes using 'Equation' as the response field, and Ape210K-ans denotes using 'Answer' as the response field. **Bold** number denotes the best result, and *italic* number denotes the worst result.

Scalability to multi-GPU environments. By utilizing some distributed data processing and synchronization mechanisms, like DeepSpeed (Rasley et al., 2020), we can extend META-LORA to multi-node or multi-GPU settings. For better comparison, we modify the single-GPU setup to a dual-GPU environment using RTX 3090 (24GB), while other settings are consistent with those in Section 4.1. Table 7 presents the performance superiority of META-LORA over direct fine-tuning under this setup, which demonstrates META-LORA's effectiveness in multi-GPU or multi-node environments, ensuring its applicability in large-scale distributed systems.

Task Metric	GSM8K ACC(%)	MBPP+ ACC(%)	MedQA ACC(%)	MedMCQA ACC(%)
Fine-tuning	28.0	25.5	25.8	28.5
META-LORA	**30.3**	**30.5**	**27.57**	**29.5**

Table 7: Performance comparison between direct fine-tuning and META-LORA in the dual-GPU environment. **Bold** number denotes the best results.