Balancing LoRA Performance and Efficiency with Simple Shard Sharing

Jiale Kang [1] **Qingyu Yin** [2]

Abstract

Parameter-Efficient Fine-Tuning (PEFT) methods, particularly Low-Rank Adaptation (LoRA), effectively reduce the number of trainable parameters in Large Language Models (LLMs). However, as model scales continue to grow, the demand for computational resources remains a significant challenge. Existing LoRA variants often struggle to strike an optimal balance between adaptability (model performance and convergence speed) and efficiency (computational overhead, memory usage, and initialization time). This paper introduces MiSS(**M**atrix **S**hard **S**haring), a novel PEFT approach that addresses this trade-off through a simple shard-sharing mechanism. MiSS leverages the insight that a low-rank adaptation can be achieved by decomposing the weight matrix into multiple fragment matrices and utilizing a shared, trainable common fragment. This method constructs the low-rank update matrix through the replication of these shared, partitioned shards. We also propose a hardware-efficient and broadly applicable implementation for MiSS. Extensive experiments conducted on a range of tasks, alongside a systematic analysis of computational performance, demonstrate MiSS's superiority. The results show that MiSS significantly outperforms standard LoRA and its prominent variants in both model performance metrics and computational efficiency, including initialization speed and training throughput. By effectively balancing expressive power and resource utilization, MiSS offers a compelling solution for efficiently adapting large-scale models.

[1]RWKV OS [2]Zhejiang University, Hangzhou. Correspondence to: Jiale Kang and Qingyu Yin <jiale@rwkvos.com, qingyu.yin@zju.edu.cn>.

Efficient Systems for Foundation Models Workshop at the International Conference on Machine Learning (ICML) 2025.,

1. Introduction

Fine-tuning Large Language Models (LLMs) (Radford et al., 2019; Raffel et al., 2020; Yin et al., 2024) is a prevalent methodology for adapting these models to specific downstream tasks. However, the comprehensive fine-tuning of all parameters typically incurs substantial computational costs. Consequently, numerous Parameter-Efficient Fine-Tuning (PEFT) techniques (Xu et al., 2023) have been developed to mitigate the training expenditure associated with these large-scale models. Among such techniques, Low-Rank Adaptation (LoRA) (Hu et al., 2021) has distinguished itself as one of the most prominent PEFT methods. LoRA employs a low-rank approximation for the weight updates, a strategy that offers a markedly reduced number of tunable parameters, notable efficacy when compared to full fine-tuning, and the potential for zero inference overhead. LoRA constructs this low-rank adaptation matrix through an intuitive design, positing that the weight update $\Delta \boldsymbol{W}$ can be approximated by the product of two lower-rank matrices, $\boldsymbol{BA} \approx \Delta \boldsymbol{W}$. Evidently, this specific factorization is not necessarily the optimal low-rank approximation of the original $\Delta \boldsymbol{W}$.

The improvements of LoRA are mimicked in two streams: (1) *Adaptability* (Ding et al., 2023; Liu et al., 2024; Biderman et al., 2024): This refers to the celerity with which the method converges to an optimal or near-optimal state. The approximation must exhibit a representational capacity comparable to that of the original, full $\Delta \boldsymbol{W}$. Extensive experiments have shown that LoRA's convergence is significantly slower compared to full fine-tuning. To address this issue, researchers have proposed several LoRA variants (Hayou et al., 2024; Meng et al., 2024; Wang et al., 2024b). By adopting different initialization strategies to influence the model's training gradients, they have accelerated LoRA's convergence speed. Different initializations of LoRA variants accelerate convergence essentially by increasing the initial gradients during training or aligning them with the full-scale training gradients. Certain solutions may improve the performance of finetuning, but makes the initialization process significantly more complex or increase the training cost, leading to a worse complexity. (2) *Efficiency* (Kopiczko et al., 2024; Wang et al., 2024a; 2025): This encompasses expeditious initialization, modest memory consumption, and minimal computational overhead. Op-

timizing LoRA from an efficiency perspective can lead to reduced VRAM consumption and an accelerated training process. However, further simplification of the LoRA structure may result in an additional loss in performance. However, the critical balance between achieving highly expressive architectures and maintaining algorithmic efficiency has often received insufficient attention.

In this work, we systematically investigate the Pareto frontier of the trade-off between Adaptability and Efficiency in PEFT. We contend that an ideal PEFT method must adeptly reconcile these two crucial aspects. To this end, we first conduct a series of foundational experiments focusing on these dimensions, encompassing a simulated pre-training and fine-tuning pipeline, computational complexity analysis, and initialization time tests. Our analysis reveals that while LoRA-like methods leveraging shard sharing can achieve excellent computational efficiency, their expressive power may be constrained by such sharing, sometimes leading to performance below the standard LoRA baseline. Conversely, SVD-based modifications to the original LoRA often exhibit strong adaptability, yet their practical efficiency can be undermined by complex initialization procedures and more intricate forward and update mechanisms.

Considering the aforementioned trade-off, our investigation reveals that a surprisingly simple strategy—constructing a low-rank matrix through the replication of a shared, partitioned shard—can achieve a compelling balance across these criteria. This approach, which we introduce as MISS, exhibits notable strengths. In terms of adaptability, MISS demonstrates superior approximation fidelity compared to LoRA and several of its variants. Its relatively substantial gradient norm (as illustrated in Figure 4) promotes rapid convergence, a key characteristic of its high adaptability. Furthermore, concerning efficiency, MISS benefits from rapid initialization; its hardware-friendly, repetition-based initialization facilitates swift processing, particularly when scaling to large models with substantial parameter counts. The formulation of MISS can also be expressed in an efficient and mathematically equivalent form that exhibits favorable time and space complexity.

We perform scaling experiments to verify our preliminary conclusions. Our extensive evaluation shows that MISS demonstrates excellent performance on understanding and generation tasks, significantly surpassing other baseline method. Moreover, we present the training dynamics *i.e.,* loss curves, which MISS also showcase a significantly faster convergence rate compared to other baselines. In generation tasks, MISS demonstrates superior performance, with evaluation metrics surpassing even the strong LoRA variants across the board. This demonstrates the feasibility of the framework. Furthermore, we validate the efficiency of MISS which enables it to significantly outperform LoRA and its variants in both computational efficiency and memory usage.

Our contributions can be summarized as threefolds:

1. In Section 3, we discuss the balancing Between Adaptability and Efficiency in LoRA-like methods with highly artificial controlled dataset to make the preliminary verification transparently and easy for replication.
2. In Section 4, we proposed MISS, an efficient and adaptable structure with shard sharing mechanism.
3. In Section 5, we scale the experiments across a wide range of datasets and models, validated its effectiveness through extensive experiments.

2. Preliminaries and Related Works

Low-Rank Adaptation (LoRA). Parameter-Efficient Fine-Tuning (PEFT) refers to a family of techniques designed to adapt large pre-trained models to downstream tasks while minimizing the number of trainable parameters, thereby reducing computational and memory overhead. Among diverse methods, Low-Rank Adaptation (LoRA) has gained significant prominence. It operates on the principle that the change in weights during model adaptation often possesses a low intrinsic rank. Instead of fine-tuning the entire pre-trained weight matrix $\boldsymbol{W}_0 \in \mathbb{R}^{d\times k}$, LoRA introduces a low-rank decomposition to represent the update. Consider a simple linear projection with input $x \in \mathbb{R}^d$ and output $y \in \mathbb{R}^k$, LoRA adapts the following forward pass:

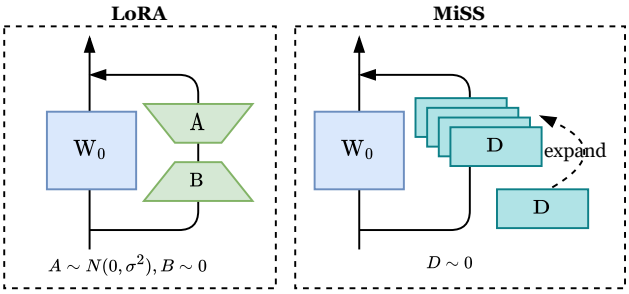
$$y = (\boldsymbol{W}_0 + \boldsymbol{\Delta W})x \approx \boldsymbol{W}_0 x + \boldsymbol{B}\boldsymbol{A}x,$$
$$\text{where } \boldsymbol{B} \in \mathbb{R}^{d\times r},\ \boldsymbol{A} \in \mathbb{R}^{r\times k}.$$

Here, $\boldsymbol{A}$ and $\boldsymbol{B}$ are low-rank matrices, with the rank r being significantly smaller than the original dimensions *i.e.,* $r \ll \min(d, k)$. During the fine-tuning process, the original weights $\boldsymbol{W}_0$ are kept frozen, and only the parameters within matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are trained. Specifically, LoRA initializes $\boldsymbol{A}$ with Gaussian noise $A \sim N(0, \sigma^2)$ with small σ and $\boldsymbol{B}$ with zeros, ensuring that $\boldsymbol{B}\boldsymbol{A} = 0$ at the start, preserving the pre-trained model's output.

Improvements of LoRA. LoRA is the low rank adaptation towards full-param finetuning, and intuitively it downperforms than it. Several works propose diverse methods towards a better convergence and adaptability of LoRA. One compelling venue is to change the form of LoRA. PiSSA (Meng et al., 2024) optimizes the compact parameter space by representing the matrices in the model as the product of two trainable matrices, augmented with a residual

Table 1. A variety of LoRA variants are listed, each with its specific update formulation and initialization strategy for the low-rank matrices. The differences between these methods are compared in a clear and intuitive manner. ^{e} denotes efficient form.

Method	Forward	Initialization
LoRA (Hu et al., 2021)	$y = \boldsymbol{W}_0 x + \boldsymbol{B}\boldsymbol{A}x$	$\boldsymbol{A} \sim N(0, \sigma^2)$ $\boldsymbol{B} \sim 0$
PiSSA	$y = \boldsymbol{W}_0 x + \boldsymbol{B}\boldsymbol{A}x$	$\boldsymbol{A} = U_{[:,:r]} S^{1/2}_{[:r,:r]}$, $\boldsymbol{B} = S^{1/2}_{[:r,:r]} V^{\top}_{[:,:r]}$
AdaLoRA	$y = \boldsymbol{W}^{(0)} x + \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{Q}x$	$\boldsymbol{\Lambda} \sim 0$, $\boldsymbol{P}, \boldsymbol{Q} \sim N(0, \sigma^2)$
DoRA	$y = \boldsymbol{m}(\ \boldsymbol{W}_0 x + \boldsymbol{B}\boldsymbol{A}x\ /\ \|\boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A}\|_c)$	$\boldsymbol{A} \sim \text{Rect.KaimingUnif}$, $\boldsymbol{B} \sim 0$
ProLoRA	$y = \boldsymbol{W}_0 x + (\boldsymbol{B}_{\boldsymbol{u}} \oplus_h \dots)\,(\boldsymbol{A}_{\boldsymbol{u}} \oplus_v \dots)x$	$\boldsymbol{A}_{\boldsymbol{u}} \sim \text{KaimingUnif}$, $\boldsymbol{B}_{\boldsymbol{u}} \sim 0$
MoS	$y = \boldsymbol{W}_0 x + \boldsymbol{B}^s \boldsymbol{A}^s x$	$\boldsymbol{A}^{\text{pub/pri}}, \boldsymbol{B}^{\text{pub/pri}} \sim 0$
MISS(Ours)	$y = \boldsymbol{W_0} x + \text{expand}(\boldsymbol{D})x$	$\boldsymbol{D} \sim 0$
MISS^{e}(Ours)	$y = \boldsymbol{W_0} x + \boldsymbol{D}^{\top}\boldsymbol{S}$	$\boldsymbol{D} \sim 0$




```
def init(in_feats: int, out_feats: int):
    weight = nn.Parameter(
        torch.empty((out_features, in_features)) )
    D = nn.Parameter(torch.zeros(self.r, out_features))
def forward(self, x):
    result = F.linear(x, self.weight) # x: [B, T, C]
    y = result + torch.sum(x.reshape
        (*x.shape[:-1], x.size(-1)//self.r, self.r)\\
        , dim=-2) @ self.miss
    return y
```

Figure 1. **Left.** Schematic diagram of the forward computation and initialization methods in LoRA and MISS. **Right.** PyTorch-style pseudocode illustrating the implementation of MISS.

matrix for error correction. Using Singular Value Decomposition (SVD), OLoRA (Büyükakyüz, 2024) leverages QR decomposition to initialize the adaptation matrices during the fine-tuning process, ensuring that these matrices are orthogonal. This orthogonal initialization helps maintain the stability of the parameter space during optimization. LoRA-GA and PiSSA are similar in form, but they differ in that LoRA-GA initializes $\boldsymbol{A}$ and $\boldsymbol{B}$ by computing the initial gradient, thereby closely approximating full fine-tuning. LoRA+ extended this method by introducing independent learning rates for matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ with a fixed ratio, improving the method's efficiency. DoRA (Liu et al., 2024) decomposes the weight matrix into two parts: magnitude and direction, which are optimized separately. This approach allows for more precise control over the learning rate, making LoRA updates closer to the effect of full fine-tuning. The improvements brought by these LoRA variants validate that the updates to the weights exhibit a low intrinsic rank during adaptation and hold greater potential. However, they also introduce more complex initialization steps and increase preprocessing time.

3. No Free Launch: Balancing Between Adaptability and Efficiency

This section elucidates the fundamental trade-off inherent in LoRA-style PEFT techniques: the delicate balance between their *adaptability* and *efficiency*. Adaptability, in this context, refers to the capacity of a given method to emulate the performance benchmarks set by full-parameter fine-tuning. Conversely, efficiency encompasses the method's judicious use of computational resources, specifically time and memory. We utilize highly artificial controlled dataset and model with relatively small parameter count to make the verification transparently and easy for replication.

We considered diverse methods [1]: (1) Full-parameter fine-tuning (Lv et al., 2024). (2) LoRA (Hu et al., 2021). (3) Alternatives to LoRA *w/* different architectures, including: PiSSA (Meng et al., 2024), VeRA (Kopiczko et al., 2024), DoRA (Liu et al., 2024) and MoRA (Jiang et al., 2024). (4) Efficent LoRA Design that keeps the LoRA $\boldsymbol{BA}$ structure: PROLORA (Wang et al., 2024a), MoS (Wang et al., 2025). (1) An overview of their forward form, initialization method can be found at Table 1.

[1]We have not included methods such as LoRA-GA (Wang et al., 2024c) or LoRA+ (Hayou et al., 2024) in our current analysis. While these approaches aim to more closely approximate the performance of full-parameter fine-tuning, we consider MISS to be largely orthogonal to them. Consequently, the analytical techniques employed in their study may still offer valuable insights for MISS.

3.1. Empirically Benchmarking the Adaptability of LoRA Variants

Experimental Setup. Parameter-efficient adaptation methods, particularly those leveraging low-rank principles, typically constrain trainable parameters by applying low-rank decompositions either to newly introduced adapter matrices or to the updates of pre-existing model weights. To rigorously evaluate such strategies, we selected a deliberately minimalistic base model: a single-layer MLP designed to process a series of features and yield outputs. This model is initially pre-trained to fit some sinusoidal functions using a constrained set of data points. Following this pre-training, the target function is subtly altered, and an additional dataset sampled from this modified function is employed for training to assess the adaptation performance of various fine-tuning techniques. Comprehensive details regarding the experimental settings are elaborated in Appendix B.

Results. Figure 2 illustrates the comparative adaptability of different methods. We utilize the minimum validation loss achieved by each approach as an indicator of its expressive capacity when approximating the performance of full-parameter fine-tuning. The results clearly demonstrate that methods leveraging singular value decomposition (SVD), such as PiSSA, attain a relatively low loss. Conversely, efficiency-focused techniques like MoS exhibit higher losses. A plausible explanation for this discrepancy is that such methods further decompose LoRA matrices into shared components which may inherently constrain their expressive power. Our method MISS reaches a relatively advanced performance comparing to other variants.

3.2. Efficiency Analysis of LoRA Variants

Metrics. We evaluate the efficiency of LoRA-like variants from two primary perspectives: (1) *Space and Time Complexity in Training*. Space and time complexity during training are generally considered crucial criteria for evaluating PEFT methods. To benchmark these aspects, we employ the model architecture detailed in Section 3.1. We also test the real cost in our experiment section *i.e.,* Section 5.5. (2) *Initialization*. Initialization time is often overlooked in theoretical complexity analyses. This oversight typically stems from the assumption that common initialization techniques (e.g., Kaiming Initialization) are computationally inexpensive and represent a one-time cost within the entire training pipeline. However, several recent advancements in LoRA and its variants incorporate matrix operations (e.g., Singular Value Decomposition - SVD) that are not inherently hardware-friendly and can pose challenges for efficient optimization and computation. Consequently, we explicitly include initialization time as a distinct evaluation metric in our experimental framework. We then progressively scale the trainable parameter count of various approaches to meticulously measure their respective time and space costs.

Results. The efficacy (See Figure 2) of MISS is evident: its strategic combination of parameter sharing and an efficient computational design culminates in rapid, scalable performance across both initialization and training stages. In contrast, while techniques like PiSSA demonstrate commendable adaptability, as shown in prior experiments, their reliance on computationally intensive Singular Value Decomposition for initialization significantly hampers their overall speed. Other approaches, such as VeRA and AdaLoRA, offer efficient initialization and computation; however, as previously discussed, they often achieve this at the cost of comparatively reduced adaptability.

4. MISS: Shard Sharing for the Performance and Efficiency Tradeoff

4.1. Method Overview

In traditional low-rank adaptation methods *e.g.,* LoRA, the weight update $\Delta \boldsymbol{W}$ is approximated as a low-rank matrix, e.g., $\Delta \boldsymbol{W} = \boldsymbol{A}\boldsymbol{B}^T$, where $\boldsymbol{A} \in \mathbb{R}^{d\times r}$, $\boldsymbol{B} \in \mathbb{R}^{k\times r}$, and the rank $r \ll \min(d, k)$. This approach achieves efficiency by limiting the number of parameters. However, we observe that a repeating matrix—where a small matrix is replicated to form a larger one—can also be viewed as a low-rank structure. For instance, if a matrix's rows or shards are constructed by repeating a limited set of independent elements, its effective rank is often much smaller than its full dimensions.

Based on this insight, we propose MISS, which defines the weight update $\Delta \boldsymbol{W}$ as a large matrix generated from a small trainable matrix $\boldsymbol{D}$ through an expansion operation. The updating of $\boldsymbol{W}$ and the forward pass can be expressed as:

$$\begin{aligned} \boldsymbol{W} &= \boldsymbol{W_0} + \Delta \boldsymbol{W} = \boldsymbol{W_0} + \text{expand}(\boldsymbol{D}), \\ y &= \boldsymbol{W_0}x + \text{expand}(\boldsymbol{D})x. \end{aligned}$$

Here, $x \in \mathbb{R}^{b\times l\times k}$, $y \in \mathbb{R}^{b\times l\times d}$, $\boldsymbol{W_0} \in \mathbb{R}^{d\times k}$ is the pre-trained weight matrix, $\boldsymbol{D} \in \mathbb{R}^{r_1\times r_2}$ is a small trainable matrix with $(r_1, r_2) \ll \min(d, k)$, and $\text{expand}(\boldsymbol{D})$ is a function that extends $\boldsymbol{D}$ to $\mathbb{R}^{d\times k}$. This structure inherently exhibits low-rank properties. Since the rows within each shard are identical, the rank of $\text{expand}(\boldsymbol{D})$ is at most N. When $N \ll d$, ΔW is a low-rank matrix, reducing the parameter count from $d \times k$ to $N \times k$.

Regarding the expansion method, we partition the output dimension d of $\boldsymbol{W_0}$ into N shards of sizes $\{s_1, s_2, \ldots, s_N\}$, where $\sum_{i=1}^{N} s_i = d$. Let $\boldsymbol{D} \in \mathbb{R}^{N\times k}$, where N is the number of shards. For each shard i, its update is determined by the i-th row of $\boldsymbol{D}$, denoted $\boldsymbol{D_i} \in \mathbb{R}^{1\times k}$, repeated s_i

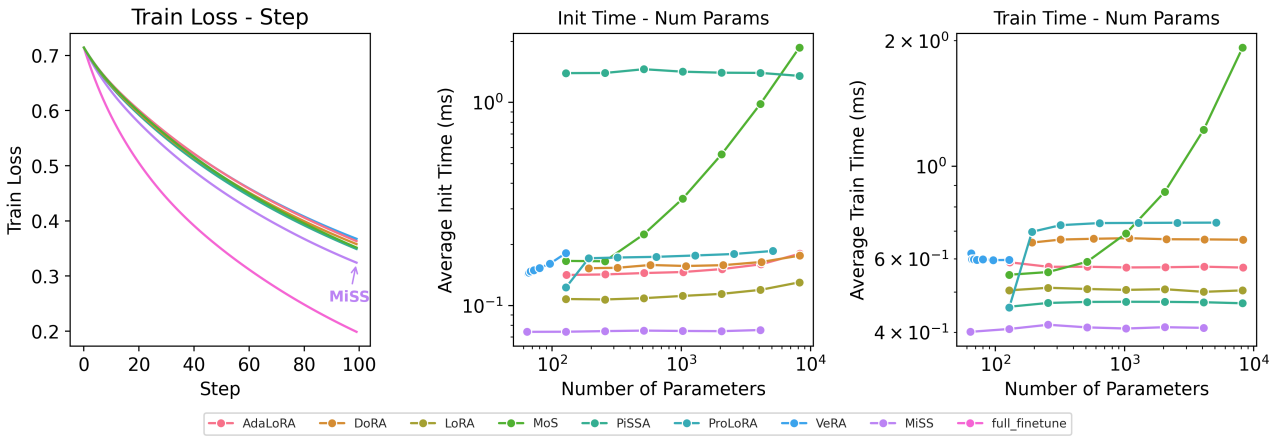


Figure 2. No Free Launch Experiment. **Left.** The training loss curves of all methods. **Middle.** Initialization time *w/* parameters. **Right.** Training time *w/* parameters.

times to form the shard's update matrix. Formally:

$$(\text{expand}(\boldsymbol{D}))^\top = \left[(\mathbf{1}_{s_1}\boldsymbol{D_1})^\top\ (\mathbf{1}_{s_2}\boldsymbol{D_2})^\top\ \dots\ (\mathbf{1}_{s_N}\boldsymbol{D_N})^\top\right] \tag{1}$$

Here, $\mathbf{1}_{s_i} \in \mathbb{R}^{s_i \times 1}$ is an all-ones vector, and $\mathbf{1}_{s_i}\boldsymbol{D_i}$ denotes $\boldsymbol{D_i}$ repeated s_i times vertically. The shards are vertically concatenated to match the dimensions of $\boldsymbol{W_0}$.

4.2. Efficient Implementation of MISS

The above formulation is effective in the initialization process, as it only needs to initialize a small $\boldsymbol{D}$. However, directly computing $\text{expand}(\boldsymbol{D})x$ has a time complexity of $O(bldk)$ and memory complexity of $O(dk)$, which can be computationally intensive. It is obvious that MISS can be transformed into an efficient form that leverages the block structure of the input to avoid explicitly forming the large matrix, by redefining $\boldsymbol{D} \in \mathbb{R}^{r \times d}$, where r is a tunable rank parameter. Instead of partitioning the output dimension d, we divide the input dimension k into r blocks, each of size $g = \lfloor k/r \rfloor$ (for simplicity, assume k is divisible by r). For an input $\boldsymbol{x} \in \mathbb{R}^{b \times l \times k}$, partition it along the k-dimension, and sum each block along the k-dimension:

$$\boldsymbol{x} = [\boldsymbol{x^{(1)}}, \boldsymbol{x^{(2)}}, \dots, \boldsymbol{x^{(r)}}], \quad \boldsymbol{x^{(i)}} \in \mathbb{R}^{b \times l \times g}, \tag{2}$$

$$\boldsymbol{S} = \left[\sum_{j=1}^{g} \boldsymbol{x^{(1)}}_{[:,:,j]}, \sum_{j=1}^{g} \boldsymbol{x^{(2)}}_{[:,:,j]}, \dots, \sum_{j=1}^{g} \boldsymbol{x^{(r)}}_{[:,:,j]}\right] \tag{3}$$

This enjoys the following updating term and forward pass:

$$\Delta \boldsymbol{W}\boldsymbol{x} = \boldsymbol{D}^\top \boldsymbol{S},\ \boldsymbol{y} = \boldsymbol{W_0}\boldsymbol{x} + \boldsymbol{D}^\top \boldsymbol{S}, \text{ where } \boldsymbol{D}^\top \in \mathbb{R}^{d \times r}. \tag{4}$$

Here $\boldsymbol{S} \in \mathbb{R}^{b \times l \times r}$, and $\boldsymbol{D}^\top \boldsymbol{S} \in \mathbb{R}^{b \times l \times d}$, matching the dimensions of $W_0 x$.

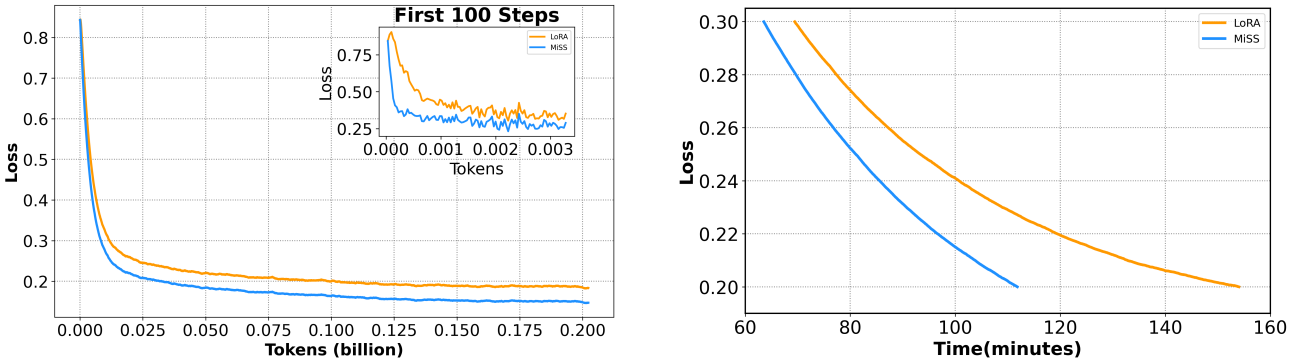
This efficient form implicitly defines $\text{expand}(\boldsymbol{D})$, such that $\text{expand}(\boldsymbol{D})x = D^T S$. Specifically, $\text{expand}(\boldsymbol{D}) \in \mathbb{R}^{d \times k}$ has rows corresponding to rows of $\boldsymbol{D}$, repeated across blocks in the k-dimension. *E.g.,* if $k = 6$, $r = 3$, and $g = 2$, the i-th row of $\text{expand}(\boldsymbol{D})$ takes values $\boldsymbol{D}_{j,i}$ in block $j = \lceil j'/g \rceil$, where j' is the column index. This structure avoids storing the $d \times k$ matrix explicitly, requiring only $\boldsymbol{D} \in \mathbb{R}^{r \times d}$, significantly reducing memory usage.

The efficient implementation of MISS relies on an innovative input aggregation mechanism, namely blockwise input summation. We highlight its advantages through the following steps: (1) *Input Partitioning and Aggregation*: The aggregation exploits local redundancy in the input, preserving critical information while reducing the computational dimensionality. (2) *Fast Computation*: The cost of computing the efficient form is significantly lower than the original complexity. (3) *Resource Savings*: Memory usage drops comparing to original form. For example, with $k = 1024$ and $r = 16$, memory is reduced by about 64 times. An overall analysis of space and time complexity is analyzed in Table 5.

5. Experiments

In this section, we evaluate the performance of MISS on various benchmark datasets. Initially, we assess Natural Language Understanding (NLU) capabilities using a subset of the GLUE dataset with the robert-base model. Subsequently, we evaluated the Natural Language Generation (NLG[2]) capabilities by fine-tuning the LLM. To ensure fair comparisons, we select methods officially included in the PEFT repository to avoid discrepancies caused by inconsistent tuning implementations.

[2]Evaluations were performed with the OpenCompass repository, and the Math dataset was evaluated using a 5-shot prompt configuration.



(a) Loss-Token

(b) Loss-Time

Figure 3. **Left.** MISS consumes less GPU memory and achieves higher throughput(# tokens per second). GPU memory usage refers to the amount of memory occupied by an RTX 4090 card. **Right.** The results of fine-tuning LLaMA2-7B on MetaMathQA.

Table 2. The results of fine-tuning RoBERTa-base using MISS and various LoRA variants were compared on a subset of the GLUE benchmark.

Method	Trainable	MNLI	SST-2	CoLA	QNLI	MRPC
LoRA	0.236%	85.63±0.01	**94.03±0.02**	62.40±0.71	91.37±0.97	87.98±0.23
PiSSA	0.236%	**85.72±0.40**	93.64±0.13	67.28±0.59	91.40±0.54	88.11±0.24
MISS	0.236%	85.71±0.32	93.60±0.07	**72.86±3.13**	**91.43±0.76**	**88.14±0.60**

5.1. Experiments on Natural Language Understanding

We fine-tune the RoBERTa-base model on several datasets from the GLUE benchmark, including MNLI, SST-2, CoLA, QNLI, and MRPC. Performance is evaluated on the development set using accuracy as the primary metric. The experimental hyperparameter settings were aligned with those in the LoRA repository, but training was conducted using a single 4090 GPU. Each experiment is conducted with 3 different random seeds, and the average performance is reported. As shown in Table 2, MISS demonstrates outstanding performance, particularly on the CoLA dataset, where it exhibits significantly faster convergence and superior data-fitting capabilities, far surpassing LoRA and PiSSA.

5.2. Experiment on Natural Language Generation

Setup. To verify the generalizability of MISS, we conducted more comprehensive experiments on LLM. we conducted 3 more task finetuning experiments on LLM: *math* and *code*. (1) *Math*: We trained our model on a 395k subset of MetaMathQA (Yu et al., 2023), a dataset bootstrapped from other math instruction tuning datasets like GSM8K (Cobbe et al., 2021) and MATH (Yu et al., 2023), with higher complexity and diversity. (2) *Code*: We train our model on a 100k subset of CodeFeedback (Zheng et al., 2024), a high-quality code instruction dataset, removing explanations after code blocks. The model is tested on HumanEval (Chen et al., 2021) and Mbpp (Austin et al., 2021). The hyperparameter settings for this experiment were kept equal, while the train steps were adjusted according to the specific fine-tuning datasets used. It is worth noting that the attention-based architectures employed by models such as LLaMA, Qwen, and Mistral do not use fully symmetric weight structures, which makes it impossible to achieve exact alignment of trainable parameters when comparing MISS with LoRA. To address this, we set the rank r of LoRA to 36 and the rank r of MISS to 64, ensuring that MISS uses fewer parameters than LoRA to demonstrate its superiority. Each experiment is conducted with 2 different random seeds, and the average performance is reported.

Results. As shown in Table 6, MISS consistently achieves state-of-the-art performance across multiple models and evaluation metrics. Specifically, on LLaMA2-7B, MISS surpasses PiSSA on most metrics while using fewer trainable parameters than both LoRA and DoRA, showcasing its parameter efficiency and superior performance. On RWKV6-7B, where all methods share the same number of trainable parameters, MISS still achieves the best overall performance, indicating its strong adaptation capability to non-transformer architectures. Across all evaluated models, including Mistral-7B, LLaMA2-13B, and Qwen3-4B, MISS maintains its leading position on key benchmarks, often achieving the highest scores with significantly fewer parameters—further highlighting its effectiveness and scalability.

Table 3. We conducted fine-tuning of large language models using MISS and multiple LoRA variants, evaluating their performance on GSM8K, Math, HumanEval, and MBPP benchmarks. All reported results are averaged over three independent runs to ensure robustness. The first-place entry should be highlighted in **bold**, and the second-place entry should be underlined.

Model	Strategy	Trainable	GSM8K	Math	HumanEval	Mbpp
Llama2-7B (Touvron et al., 2023)	LoRA	89.9M	40.75	5.22	17.74	35.15
	DoRA	91.3M	42.93	6.51	21.95	36.53
	PiSSA	89.9M	43.89	6.92	22.15	**37.84**
	MISS	**87.0M**	**48.16**	**8.58**	**23.63**	36.81
RWKV 6-7B (Peng et al., 2024)	LoRA	88.1M	38.13	6.06	-	-
	PiSSA	88.1M	40.48	6.12	-	-
	MISS	88.1M	**41.73**	**6.52**	-	-
Mistral-7B (Jiang et al., 2023)	LoRA	94.4M	62.85	15.82	35.71	46.11
	DoRA	95.8M	63.68	13.60	38.41	48.73
	PiSSA	94.4M	67.01	18.13	41.28	51.37
	MISS	**87.0M**	**68.92**	**18.85**	**42.07**	**61.33**
Llama2-13B (Touvron et al., 2023)	LoRA	250M	56.18	12.60	31.79	37.82
	DoRA	252M	61.56	13.60	33.50	39.25
	PiSSA	250M	66.64	13.82	33.57	46.03
	MISS	255M	**68.64**	**15.74**	**38.15**	**47.91**
Qwen3-4B (Yang et al., 2025)	LoRA	74.3M	84.38	15.20	73.27	78.32
	DoRA	75.4M	85.11	21.73	74.20	**78.77**
	PiSSA	74.3M	**85.78**	26.00	**75.01**	78.04
	MISS	**70.1M**	85.52	**34.82**	74.48	78.05

5.3. Effect of Rank r

This subsection explores the upper limits of the MISS structure by varying the rank r in the MISS matrix. Comparative experiments were conducted by fine-tuning LLaMA2-7B [3] on the MetaMathQA dataset and validating on GSM8K and Math benchmarks. The test results, as shown in Table 4, demonstrate that the fine-tuning performance improves as the value of b increases. Notably, when $r = 16$, the MISS structure, with only one-quarter of the trainable parameters compared to PiSSA, surpasses PiSSA's performance on the GSM8k benchmark. However, its performance on the Math benchmark is only 3.73. The GSM8K score surpasses that of PiSSA, but the Math score is significantly lower, indicating The size of r impacts the model's ability to understand unseen data. Based on this observation, we hypothesize that when the rank is too small, it significantly limits the model's generalization ability.

5.4. Gradient Norm Analysis

To further investigate the adaptive capabilities of MISS, we analyzed its initial gradient norms in comparison to Full Fine-tuning (FT), LoRA, and PiSSA. The magnitude of initial gradient norms is often correlated with faster convergence and the ability of a model to effectively adapt during fine-tuning. Our comparative analysis, illustrated in Figure 4 (Right), reveals distinct behaviors among the evaluated methods. Standard LoRA consistently exhibits the lowest initial gradient norms across the tested ranks. While PiSSA's gradient norms increase with higher ranks, they generally remain significantly lower than those achieved by full fine-tuning. Notably, MISS demonstrates substantially larger initial gradient norms compared to both LoRA and PiSSA across various ranks. More importantly, the gradient norm profile of MISS closely approximates that of Full Fine-tuning. This proximity to FT's gradient characteristics suggests that MISS is capable of inducing more significant initial updates, potentially leading to a training dynamic more akin to full fine-tuning.

5.5. Resource and efficiency

Table 5 compares the training resources and token throughput required for fine-tuning RWKV6 using LoRA and MISSon a single 4090 GPU. The specific fine-tuning settings are as follows: batch size = 1, context length (ctx_len) = 512. The results show that MISS has the highest computational efficiency, being nearly 10% faster than LoRA while also being more memory-efficient. At the end of the table, we provide the actual resource costs for fine-tuning RWKV6 on the MetaMathQA dataset using 4 NVIDIA 4090 GPUs, with checkpoint techniques applied.

[3]We use LLaMA2-7B instead of LLaMA3-8B because we found that the LLaMA3 series is over-optimized on math-related tasks.

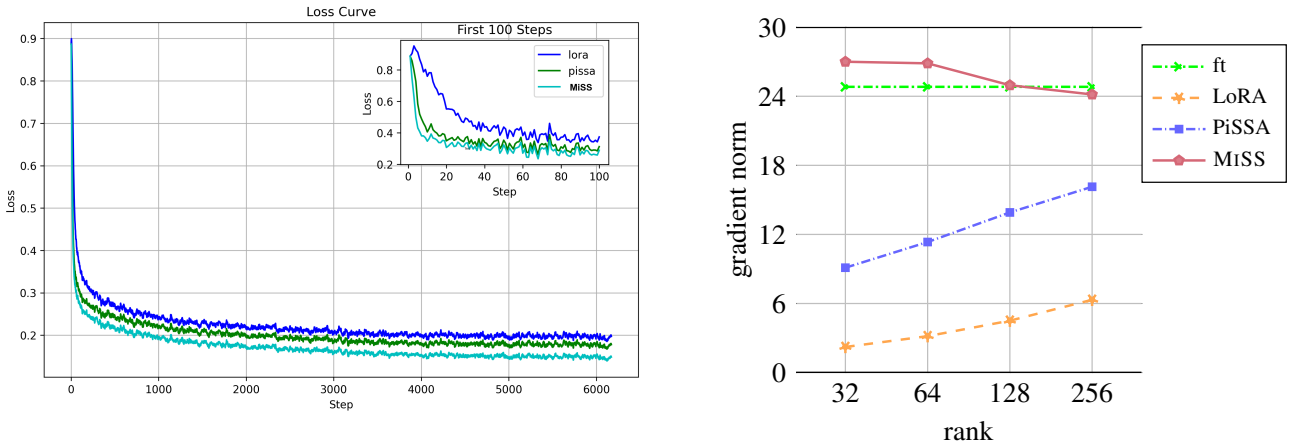


Figure 4. **Left.** The loss curve for LLaMA2-7B fine-tuned on MetaMathQA shows MISS's superior fitting ability across architectures and parameter settings, with a notably rapid loss reduction in the first 100 steps, underscoring its effectiveness. **Right.** The initial gradient norm of model training.

Table 4. Comparing different values of rank (r) on LLaMA2-7B with MISS.

Model	Rank	Trainable	GSM8K	Math
Llama2-7B	16	21.7M	45.90	3.77
	32	43.5M	46.18	7.43
	64	87.0M	48.16	8.58
	128	174.0M	53.49	10.08

Figure 5. Time and space complexity. ^{e} denotes efficient form.

Method	Space	Time
Full	$O(dk)$	$O(bld(d+k))$
LoRA	$O(dr+rk)$	$O(blr(d+k))$
MISS	$O(dr)$	$O(bldk)$
MISS^e	$O(dr)$	$O(blr(d+\frac{k}{r}))$

Table 5. Resource and efficiency

Devices	Strategy	Trainable	Memory	throughput
GPU × 1	LoRA	55.1M	12074 MB	3.62 kt/s
	MISS	55.1M	**11052** MB	**3.99** kt/s
GPU × 4	LoRA	55.1M	4×15328 MB	15.6 kt/s
	MISS	55.1M	**4**×**15304** MB	**16.0** kt/s

6. Conclusion

In this work, we proposed the Foissl framework, which divides pre-trained weights into multiple shards and updates them using a shared trainable matrix. This approach significantly reduces resource overhead and opens up new directions for PEFT techniques. Furthermore, we present a hardware-optimized version of MISS, achieving remarkable gains in computational efficiency. Extensive experiments demonstrated the superiority of MISS, which outperforms LoRA and its variants in evaluation metrics, computational efficiency, and resource usage.

Collectively, our framework redefines the design principles of efficient adaptation: MISS provides theoretical grounding for dimension-wise decomposition, MISS delivers practical efficiency parity with LoRA. This progression demonstrates that parameter efficiency need not come at the cost of expressivity, paving the way for adaptive fine-tuning in resource-constrained environments.

7. Limitations and Future work

Due to hardware constraints, we were unable to conduct and report results from the full-scale training that this method ideally requires. Nonetheless, through extensive evaluations and thorough comparisons with LoRA and its variants, we have validated the effectiveness of MISS. The strong performance achieved on large language models gives us substantial confidence in the potential of this approach. We believe that MISS can be readily applied to a wide range of multimodal tasks.

As a pioneering approach, MISS still leaves several aspects open for deeper exploration. We hope that future research will conduct broader and more in-depth studies to further refine PEFT techniques and identify the most effective strategies for large language models.

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Biderman, D., Ortiz, J. G., Portes, J., Paul, M., Greengard, P., Jennings, C., King, D., Havens, S., Chiley, V., Frankle, J., et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024.

Büyükakyüz, K. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.-M., Chen, W., et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

Hayou, S., Ghosh, N., and Yu, B. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Jiang, T., Huang, S., Luo, S., Zhang, Z., Huang, H., Wei, F., Deng, W., Sun, F., Zhang, Q., Wang, D., and Zhuang, F. Mora: High-rank updating for parameter-efficient fine-tuning, 2024. URL https://arxiv.org/abs/2405.12130.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation, 2024. URL https://arxiv.org/abs/2310.11454.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Lv, K., Yang, Y., Liu, T., Gao, Q., Guo, Q., and Qiu, X. Full parameter fine-tuning for large language models with limited resources, 2024. URL https://arxiv.org/abs/2306.09782.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Ferdinan, T., Hou, H., Kazienko, P., et al. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 2024.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wang, S., Xue, B., Ye, J., Jiang, J., Chen, L., Kong, L., and Wu, C. Prolora: Partial rotation empowers more parameter-efficient lora, 2024a. URL https://arxiv.org/abs/2402.16902.

Wang, S., Yu, L., and Li, J. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*, 2024b.

Wang, S., Yu, L., and Li, J. Lora-ga: Low-rank adaptation with gradient approximation, 2024c. URL https://arxiv.org/abs/2407.05000.

Wang, S., Chen, L., Chen, P., Dong, J., Xue, B., Jiang, J., Kong, L., and Wu, C. Mos: Unleashing parameter efficiency of low-rank adaptation with mixture of shards, 2025. URL https://arxiv.org/abs/2410.00938.

Xu, L., Xie, H., Qin, S.-Z. J., Tao, X., and Wang, F. L. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*, 2023.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yin, Q., He, X., Zhuang, X., Zhao, Y., Yao, J., Shen, X., and Zhang, Q. Stablemask: Refining causal masking in decoder-only transformer. *arXiv preprint arXiv:2402.04779*, 2024.

Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J. T., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Zheng, T., Zhang, G., Shen, T., Liu, X., Lin, B. Y., Fu, J., Chen, W., and Yue, X. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

A. Appendix

A.1. RWKV7

Table 6. We fine-tuned LLMs using MISS and various LoRA variants, and evaluated performance on GSM8k, Math, HumanEval, and MT-Bench.

Model	Strategy	Trainable	GSM8K	Math	HumanEval	MT-Bench
RWKV7-3B	Base	0M	44.35	-	-	-
	LoRA	47.2M	55.64	-	-	-
	PiSSA	47.2M	57.16	-	-	
	MISS	47.2M	**58.22**	-	-	-

Table 7. Hyperparameter settings for fine-tuning llama2-7B,Mistral-7B,RWKV6-7B,Qwen3-4B on NLG tasks

Hyperparameters	LoRA	DoRA	PiSSA	MISS
Rank r	36	36	36	64
α	72	72	36	-
Dropout	0.0			
Optimizer	AdamW			
LR	2e-5			
LR Scheduler	Cosine decay			
Batch size	64			
Warmup ratio	0.0			
Epochs	1			
Where	Q,K,V,O,Up,Down,Gate			

Table 8. Hyperparameter settings for fine-tuning llama2-13B on NLG tasks

Hyperparameters	LoRA	DoRA	PiSSA	MISS
Rank r	64	64	64	128
α	128	128	64	-
Dropout	0.0			
Optimizer	AdamW			
LR	2e-5			
LR Scheduler	Cosine decay			
Batch size	128			
Warmup ratio	0.0			
Epochs	1			
Where	Q,K,V,O,Up,Down,Gate			

B. Settings of Experiments in No Free Lunch

Table 9. Experimental Setup: Datasets and Hyperparameters

Parameter	Value
General Configuration	
Random Seed (SEED)	43
Device (DEVICE)	CUDA (if available, else CPU)
Base Model Architecture (MLP)	
Input Dimension	64
Hidden Dimension	64
Output Dimension	64
Synthetic Dataset Generation	
Base Function	$\sin(2\pi x)$
Modified Function	$\sin(2\pi x) + 0.3\cos(3\pi x)$
Input x Range	$[-1, 1]$
Training Samples (N_TRAIN)	50
Validation Samples (N_VALID)	100
Training Noise Std. Dev. (NOISE_STD)	0.05
Validation Noise Std. Dev.	0.0
Training Parameters	
Base Model LR (BASE_LR)	0.001
Adaptation LR (ADAPT_LR)	0.001
Base Model Epochs (BASE_EPOCHS)	250
Adaptation Epochs (ADAPT_EPOCHS)	100
Evaluation Interval (EVAL_INTERVAL)	10
Adapter-Specific Ranks	
LoRA Rank	2
VeRA Rank	64
MISSRank	4
PiSSA Rank	2
DoRA Rank	1
ProLoRA Rank	2
AdaLoRA Rank	2
MoS Rank	2

Note: Other adapter-specific hyperparameters (e.g., LoRA scale, VeRA d_init_val, DoRA lora_alpha, ProLoRA unshared_rank_u, MoS shard_dim_ratio) primarily use their default values as defined in the respective adapter class implementations or are derived based on the rank within benchmark functions. Refer to the provided Python code for their specific configurations during experiments.