

# HD-PiSSA: High-Rank Distributed Orthogonal Adaptation

Yiding Wang<sup>\*1,2</sup>, Fanxu Meng<sup>\*1,3</sup>, Xuefeng Zhang<sup>1</sup>, Fan Jiang<sup>1,2</sup>,  
Pingzhi Tang<sup>1,2</sup>, Muhan Zhang<sup>1,3</sup>

<sup>1</sup>Institute for Artificial Intelligence, Peking University    <sup>2</sup>Yuanpei College, Peking University

<sup>3</sup>State Key Laboratory of General Artificial Intelligence, BIGAI

\*Equal contribution    ✉ Correspondence to [muhan@pku.edu.cn](mailto:muhan@pku.edu.cn)

## Abstract

Existing parameter-efficient fine-tuning (PEFT) methods for large language models (LLMs), such as LoRA and PiSSA, constrain model updates to low-rank subspaces, limiting their expressiveness and leading to suboptimal performance on complex tasks. To address this, we introduce **High-rank Distributed PiSSA (HD-PiSSA<sup>1</sup>)**, a distributed PEFT approach that initializes **orthogonal adapters** across different devices and aggregates their delta updates collectively on  $W$  for fine-tuning. Unlike Data Parallel LoRA or PiSSA, which maintain identical adapters across all devices, HD-PiSSA assigns different principal components of the pre-trained weights to each GPU, significantly expanding the range of update directions. This results in over  $16\times$  higher effective updated ranks than data-parallel LoRA or PiSSA when fine-tuning on 8 GPUs with the same per-device adapter rank. Empirically, we evaluate HD-PiSSA across various challenging downstream tasks, including mathematics, code generation, and multi-task learning. In the multi-task setting, HD-PiSSA achieves average gains of 10.0 absolute points (14.63%) over LoRA and 4.98 points (6.60%) over PiSSA across 12 benchmarks, demonstrating its benefits from the extra optimization flexibility.

## 1 Introduction

Large Language Models (LLMs) trained on general corpora have shown strong language modeling capabilities and broad generalization across various downstream applications (Taori et al., 2023; Luo et al., 2023). However, adapting these general models to specific application scenarios often requires fine-tuning or additional training on domain-specific data. While fine-tuning has proven effective, Full Fine-Tuning (FFT) LLMs are usually prohibitively expensive in terms of both computation and memory (Dettmers et al., 2024). The

<sup>1</sup>The code is available at [MuLabPKU/HD-PiSSA](https://github.com/MuLabPKU/HD-PiSSA)

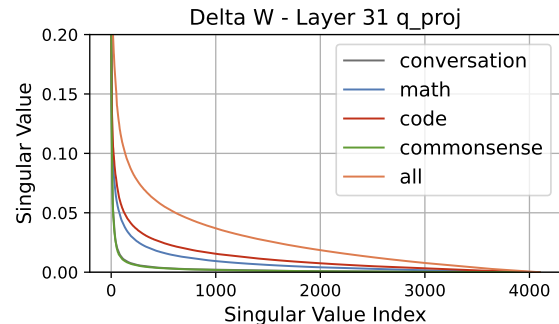


Figure 1: The rank analysis of  $\Delta W$  after Full Fine-Tuning (FFT) on different datasets. The magnitudes of the singular values in  $S$  provide a measure of the effective updated rank. Details of this analysis are provided in Appendix A.2.

cost of updating all model parameters becomes particularly challenging when working with models containing billions of parameters. To address this challenge, Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019) methods have been developed as more practical alternatives. These methods significantly reduce the number of trainable parameters while maintaining competitive performance, making them both efficient and scalable.

Among PEFT techniques, Low-Rank Adaptation (LoRA) (Hu et al., 2021) is widely adopted for its simplicity and efficiency. It approximates weight updates  $\Delta W$  with a low-rank decomposition  $\Delta W = AB$ , based on the assumption that adaptation occurs in a low-rank subspace. This design reduces training cost and parameter count without increasing inference latency. PiSSA (Meng et al., 2024) improves LoRA by initializing adapters with the top singular vectors of pre-trained weights, encouraging updates to follow more informative directions. Despite these improvements, both methods share a fundamental limitation: they operate under a fixed low-rank constraint that restricts update capacity. This limitation

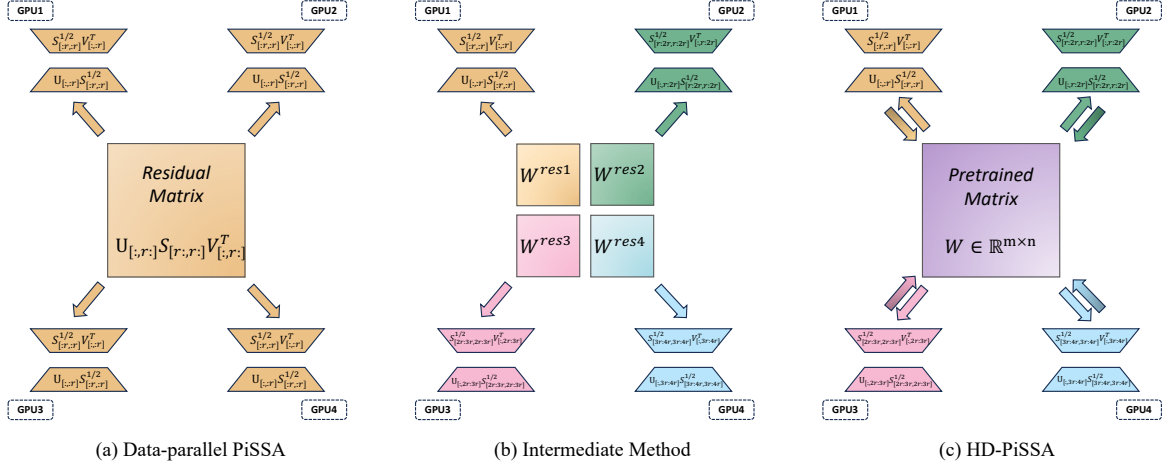


Figure 2: An illustration of the transition from Data-Parallel PiSSA to HD-PiSSA. PiSSA initializes and updates identical adapters across GPUs using only the top- $r$  components, leading to a low-rank update limitation. A natural extension is the Intermediate Method, which increases the updated rank by distributing the top- $Kr$  components across GPUs to construct orthogonal adapters. However, this approach introduces two key challenges: (1) How can single-step updates be performed on heterogeneous adapters? (2) Is it necessary to maintain separate copies of the residual matrix on different GPUs? HD-PiSSA addresses these challenges by introducing Direct Weight Update and a Muting Mechanism, enabling efficient, high-rank updates without the need for separate residual matrices.

becomes evident when adapting to complex tasks. As shown in Fig. 1, Full Fine-Tuning (FFT) yields much higher update ranks on complex tasks such as code generation and math than on simpler ones like commonsense or conversation. These observations suggest that **effective adaptation to complex tasks demands higher update ranks**, highlighting the need for more expressive PEFT approaches.

To address this, we propose **HD-PiSSA**, which leverages data parallelism to break through the low-rank bottleneck. Unlike prior methods that replicate identical adapters across devices, HD-PiSSA assigns orthogonal adapters initialized from disjoint principal components to each GPU, thereby expanding the effective update subspace. Instead of updating the adapters directly, we compute and aggregate their induced updates and **apply them directly to the shared pretrained weights  $W$** . Furthermore, to avoid maintaining separate residual matrices across devices, we introduce a **Muting Mechanism**, which enables all devices to share a single full-rank model while still benefiting from heterogeneous adapter updates. Fig. 2 illustrates the transition from standard PiSSA to HD-PiSSA.

As illustrated in Fig. 3, HD-PiSSA overcomes the rank limitation of PiSSA and closely resembles FFT with smooth full-rank updates. We formalize this improvement through a rank analysis in Section 7. In addition, we evaluate the performance of HD-PiSSA on a wide range of challenging fine-tuning tasks, including mathematics,

code generation, and multi-task learning. The results show that HD-PiSSA **outperforms LoRA and PiSSA while maintaining a similar number of optimizer parameters**. We further compare HD-PiSSA with other recent PEFT methods, like DoRA (Liu et al., 2024) and high-rank updating methods MoRA (Jiang et al., 2024) and HiRA (Anonymous, 2025) on commonsense reasoning tasks. HD-PiSSA outperforms these baselines with a significant margin (+2.0). HD-PiSSA exhibited key characteristics of FFT, including the ability to perform high-rank or theoretically full-rank updates, underscoring its potential as an efficient and scalable alternative to Full Fine-Tuning for complex tasks.

A summary of our contributions is as follows:

- We propose HD-PiSSA, a novel distributed PEFT method that enables high-rank updates under data parallel fine-tuning, achieving a learning capacity resembling FFT without additional inference latency over LoRA.
- HD-PiSSA consistently outperforms existing PEFT methods, including LoRA, PiSSA and other high-rank updating methods, across single-task and multi-task fine-tuning settings.
- We provide empirical analysis of update ranks of FFT, LoRA, PiSSA and HD-PiSSA, revealing key insights into the learning patterns and limitations of different PEFT methods.

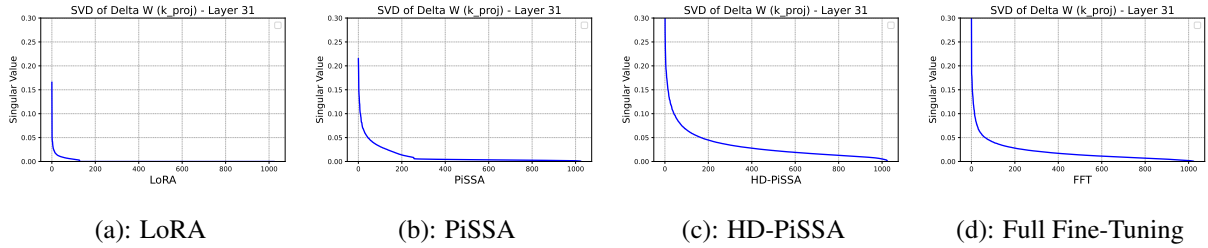


Figure 3: SVD analysis of  $\Delta W$  after fine-tuning with different methods on the same training set. The rank of  $\Delta W$  is bounded by  $r$  or  $2r$  for LoRA and PiSSA. (The effective updated rank of PiSSA is also  $r$ , and the appearance of  $2r$  is due to its operation of first subtracting the adapter from the pretrained weight and then merging the updates after fine-tuning.) HD-PiSSA demonstrates a similar smooth full-rank update pattern to Full Fine-Tuning. More details are included in Appendix A.3.

## 2 Related Works

### 2.1 Parameter-Efficient Fine-Tuning (PEFT)

PEFT (Han et al., 2024) methods are designed to reduce the computational or memory cost of fine-tuning large-scale models. Broadly, PEFT methods can be categorized into four groups. The first category is *partial fine-tuning* (Zaken et al., 2021; Xu et al., 2021), which involves updating only a subset of the original model’s parameters based on a predefined selection strategy. The second category is *soft prompt fine-tuning*, which introduces additional soft prompt tokens or vectors into the initial input and fine-tunes only these vectors (Lester et al., 2021; Li and Liang, 2021). However, these methods are sensitive to input variations and initialization, which limits their effectiveness in downstream tasks. The third category is *adapter-based* methods. These methods (Houlsby et al., 2019; Pfeiffer et al., 2020; He et al., 2021) incorporate additional trainable modules into original frozen models. While these methods enhance performance, they come at the cost of increased inference-time latency.

### 2.2 Low-rank Adaptation Methods

Low-rank adaptation methods, such as LoRA (Hu et al., 2021) and its variants, fall into the fourth category. These methods use low-rank matrices to approximate weight changes during fine-tuning and can be merged with pre-trained weights before inference. They have gained popularity due to their ability to retain the model’s original architecture while enabling efficient fine-tuning. Building on LoRA, O-LoRA (Wang et al., 2023) introduces orthogonal regularization to mitigate catastrophic forgetting, DeltaLoRA (Zi et al., 2023) injects adapter-informed updates into the original weights, and DoRA (Liu et al., 2024) decouples weight mag-

nitude and direction. PiSSA (Meng et al., 2024) initializes low-rank adapters using the principal components of the original weights and fine-tunes these components. Despite their efficiency, these methods are inherently constrained by fixed low-rank updates, which can limit their expressiveness on complex tasks. To address this, recent work has explored high-rank adaptation. Hao et al. (2024) interpret LoRA as a gradient compressor and propose Flora to enable high-rank updates with reduced memory. MoRA (Jiang et al., 2024) replaces the low-rank structure with square matrices to enable full-rank updates, while HiRA (Anonymous, 2025) introduces a Hadamard product mechanism to enrich the update space. Unlike previous high-rank PEFT methods that increase capacity by modifying adapter structures on each device, HD-PiSSA leverages data-parallel training to achieve high-rank adaptation via distributed and orthogonal updates. This design retains the efficiency of low-rank methods while substantially expanding the update space.

## 3 Background and Motivation

### 3.1 LoRA Basics

In this paper, we denote the original model’s weight matrix as  $W \in \mathbb{R}^{m \times n}$ , where  $m$  and  $n$  represent the input and output dimensions, respectively. LoRA approximates the weight update  $\Delta W$  using a low-rank decomposition:  $\Delta W = AB$ , where  $A \in \mathbb{R}^{m \times r}$  and  $B \in \mathbb{R}^{r \times n}$  with  $r \ll \min(m, n)$ . The original weights  $W$  are frozen during fine-tuning, and only the low-rank matrices  $A$  and  $B$  are trainable. Typically,  $A$  is initialized with Kaiming Uniform initialization, while  $B$  is initialized to zero, resulting in an initial  $\Delta W = 0$ .

### 3.2 PiSSA Basics

PiSSA builds upon the low-rank adaptation idea but introduces a more informed initialization strategy. Specifically, it first performs Singular Value Decomposition (SVD) on the original weight matrix  $W$ , obtaining:  $W = U\Sigma V^\top$ , where  $U \in \mathbb{R}^{m \times d}$ ,  $\Sigma \in \mathbb{R}^{d \times d}$ , and  $V \in \mathbb{R}^{n \times d}$ , with  $d = \min(m, n)$ . PiSSA then selects the top  $r$  principal components from  $U$  and  $V$  to initialize the low-rank matrices  $B$  and  $A$ , respectively. That is,

$$A = U_r \Sigma_r^{1/2}, \quad B = \Sigma_r^{1/2} V_r^\top,$$

The original weight is then replaced by a residual form  $W_{res} = W - AB$  to preserve the model’s output consistency. This initialization captures the key directions of  $W$ , offering better fine-tuning performance than random initialization.

### 3.3 Distributed High-Rank Adaptation

While data parallelism is a widely used paradigm for large language model fine-tuning, existing parameter-efficient tuning methods, including PiSSA, do not fully exploit its potential. In conventional PiSSA (as shown in Fig. 2a), all devices fine-tune the same adapters initialized from top singular components, leading to limited update rank and underutilization of model capacity.

To address this, we observe that data parallel training naturally allows each GPU to maintain a local copy of the model and its adapters. This opens up the opportunity to assign *heterogeneous*, non-overlapping adapter modules to different GPUs. By distributing distinct subsets of principal components across devices, we effectively enlarge the overall update rank without increasing memory overhead on any single GPU. This simple yet powerful insight lays the foundation for our method, HD-PiSSA, which leverages the structure of data parallelism to scale up the expressiveness of low-rank updates through adapter heterogeneity, while keeping computational and memory costs comparable to conventional approaches.

## 4 Methodology

Building on the insight that data parallelism enables heterogeneous adapters across devices, we present **HD-PiSSA (High-rank Distributed PiSSA)**, which enhances update capacity while maintaining efficiency. Specifically, HD-PiSSA introduces three key modifications to standard PiSSA. First, we adopt *orthogonal adapter initialization*: each

GPU is assigned a distinct subset of principal components, enabling it to fine-tune a unique set of adapters. This increases the effective update rank without incurring extra memory costs. Second, we employ a *direct weight update* strategy: instead of updating  $A$  and  $B$ , gradients are applied directly to the original weights  $W$ , while keeping the adapters fixed. Finally, we design a *muting mechanism* using a learnable scalar that suppresses adapter outputs during the forward pass, enabling gradient flow without modifying model predictions or requiring subtraction of  $AB$ . Together, these designs make HD-PiSSA a simple, scalable, and efficient solution for high-rank adaptation in data-parallel training.

### 4.1 Orthogonal Adapters Initialization

Assuming we are fine-tuning on  $K$  devices, HD-PiSSA begins by applying SVD to each weight matrix  $W \in \mathbb{R}^{m \times n}$ :

$$W = USV^T,$$

where  $U$  and  $V$  are orthogonal matrices and  $S$  contains the singular values. Instead of selecting only the top- $r$  components as in PiSSA, we select the top- $Kr$  components and partition them across devices. For each device  $i \in \{0, \dots, K - 1\}$ , we construct the adapter pair as:

$$A_i = U_{[:,r_i:r_{i+1}]} S_{[r_i:r_{i+1}]}^{1/2}, \quad B_i = S_{[r_i:r_{i+1}]}^{1/2} V_{[:,r_i:r_{i+1}]}^\top,$$

where  $r_i = i \cdot r$ , and  $r$  denotes the predefined rank that each device handles locally. Compared to data-parallel PiSSA, which updates only the top-rank components using the entire batch of data, HD-PiSSA initializes adapters in a way that can distribute the batch data evenly across a larger set of principal components, thereby exploring more directions and increasing the fine-tuned parameter space. Naively, different adapters imply different residual weights  $W_{res,i} = W - A_i B_i$  to maintain consistent outputs across devices. However, as we show in the Section 4.3, our muting mechanism allows us to avoid this by sharing a single global weight matrix  $W$  across all devices without compromising correctness. This enables efficient distributed fine-tuning with heterogeneous adapters while preserving model consistency.

### 4.2 Direct Weight Update

With orthogonal adapters distributed across devices, their gradients lie in distinct subspaces and cannot be directly averaged, unlike in standard data-



parallel LoRA or PiSSA. To address this, HD-PiSSA keeps  $A_i, B_i$  unchanged and instead translates their updates into an equivalent update on the shared weight matrix  $W$ .

Specifically, we assume that each device performs forward and backward propagation and obtains the gradients  $g_{A_i}^t \in \mathbb{R}^{m \times r}$  and  $g_{B_i}^t \in \mathbb{R}^{r \times n}$  on its mini-batch data at step  $t$ . Then, we are able to compute the delta update  $\Delta A_i^t \in \mathbb{R}^{m \times r}$  and  $\Delta B_i^t \in \mathbb{R}^{r \times n}$  correspondingly, where we temporarily ignore the discussion of optimizers. If we simply update  $A$  and  $B$  at each step, the process could be formalized as follows:

$$A_i^{t+1} = A_i^t + \Delta A_i^t, \quad B_i^{t+1} = B_i^t + \Delta B_i^t.$$

This leads to the following expression for the delta update  $U_i^t \in \mathbb{R}^{m \times n}$  to the product of low-rank matrices  $A_i$  and  $B_i$  at step  $t + 1$ :

$$\Delta U_i^t = \Delta A_i^t B_i^t + A_i^t \Delta B_i^t + \Delta A_i^t \Delta B_i^t,$$

These deltas are aggregated and then applied to  $W$ :

$$W^{t+1} = W^t + \frac{1}{K} \sum_{i=0}^{K-1} \Delta U_i^t.$$

where  $K$  is the total number of devices.

We refer to this strategy as **Direct Weight Update**. Through Direct Weight Update, we achieve joint fine-tuning of the top- $Kr$  principal components across different orthogonal directions of the original weight matrix, significantly increasing the effective updated rank. Additionally, since the  $\Delta A$  and  $\Delta B$  terms allow arbitrary rotations and translations according to data, HD-PiSSA not only updates the top- $Kr$  components, but also enables a theoretically full-rank update. (More details are included in Section 7.)

### 4.3 Muting Mechanism

Since we initialize orthogonal adapters on different devices, and the dot product between  $A$  and  $B$  in these adapters is nonzero, following PiSSA’s forward propagation paradigm

$$Y = X(W_{res,i} + A_i B_i) \quad (1)$$

would require subtracting the dot product  $AB$  from the original matrix on each device and maintaining a separate  $W_{res}$  for each GPU. This introduces significant operational and computational overhead, and makes it difficult to scale HD-PiSSA to larger

models, as each device would need to store an independent copy of the full model. Therefore, we introduce the **muting mechanism**, which exploits the automatic differentiation mechanism and makes it feasible to store only one shared model  $W$  across devices.

Specifically, we introduce a small constant (1e-16 in our main experiments), **the Mute Scalar**  $\gamma$ . During forward propagation, we multiply this scalar to eliminate the impact of different adapters on the output:

$$Y = X(W + \gamma A_i B_i).$$

This operation ensures that 1) the differences among  $A_i B_i$  are muted in the forward propagation so that all devices behave the same, and 2) we can maintain a single model  $W$  shared across devices instead of maintaining  $W_{res,i}$  for each device, thus apply the averaged update only to  $W$ .

Although  $A_i B_i$  are muted during forward propagation, their gradients still provide directional information for update to the combined model  $W$ . Therefore, during backward propagation, we rescale the gradients on  $A$  and  $B$  to get:

$$g_{A_i} \leftarrow \frac{1}{\gamma} \frac{\partial \mathcal{L}}{\partial A_i}, \quad g_{B_i} \leftarrow \frac{1}{\gamma} \frac{\partial \mathcal{L}}{\partial B_i},$$

where  $\frac{\partial \mathcal{L}}{\partial A_i}$  and  $\frac{\partial \mathcal{L}}{\partial B_i}$  are the gradients of the loss  $\mathcal{L}$  with respect to the adapters  $A_i$  and  $B_i$ , respectively. It is easy to prove that the obtained  $g_{A_i}$  and  $g_{B_i}$  **are equal to** the gradients of  $A_i$  and  $B_i$  in the normal forward propagation as in Equation 1. Therefore, we have recovered the actual gradients while also saving the need of different  $W_{res,i}$  at the same time through this smart Muting Mechanism.

## 5 Experiments

In this section, we comprehensively evaluate HD-PiSSA. We begin with single-task fine-tuning (Section 5.1) on math and code datasets using LLaMA-2-7B, LLaMA3.1-8B, and Mistral-7B-v0.1, comparing against LoRA and PiSSA. In Section 5.2, we extend to a more challenging multi-task setting that includes math, code, conversation, and commonsense reasoning. We also benchmark HD-PiSSA under the popular commonsense evaluation setup from Liu et al. (2024), comparing it to high-rank PEFT baselines like MoRA, HiRA, and DoRA. Lastly, we examine HD-PiSSA’s robustness across different numbers of devices (Section 6.1) and ranks (Section 6.2).

Table 1: Accuracy comparison of LLaMA-2-7B, LLaMA-3.1-8B, and Mistral-7B-v0.1 with LoRA, PiSSA, and HD-PiSSA on mathematics and code generation datasets. “Params.” denotes the per-GPU optimizer parameters, which remain the same across LoRA, PiSSA, and HD-PiSSA.

MODEL	METHOD	PARAMS.	GSM8K	MATH	HUMANEVAL	MBPP	AVG.
LLAMA 2-7B	LoRA	0.59%	44.58	6.16	15.20	33.60	24.89
	PiSSA	0.59%	49.13	7.46	18.90	37.10	28.15
	<b>HD-PiSSA</b>	0.59%	<b>52.92</b>	<b>8.38</b>	<b>21.30</b>	<b>37.30</b>	<b>29.98</b>
LLAMA 3.1-8B	LoRA	0.52%	75.66	25.92	50.60	66.10	54.57
	PiSSA	0.52%	75.74	27.06	49.40	66.40	54.65
	<b>HD-PiSSA</b>	0.52%	<b>76.12</b>	<b>28.52</b>	<b>52.40</b>	<b>69.30</b>	<b>56.59</b>
MISTRAL-7B-v0.1	LoRA	0.58%	65.81	18.10	41.50	<b>58.70</b>	46.03
	PiSSA	0.58%	68.46	18.46	41.50	58.30	46.68
	<b>HD-PiSSA</b>	0.58%	<b>68.68</b>	<b>19.34</b>	<b>42.10</b>	57.90	<b>47.01</b>

## 5.1 Performance for Single-Task Fine-Tuning

As shown in Fig. 1, we use math and code datasets to represent complex single-task datasets. Specifically, we use MetaMathQA (Yu et al., 2023) to fine-tune mathematical problem-solving capability and then use GSM8K (Cobbe et al., 2021) and MATH datasets for evaluation. We use CodeFeedback (Zheng et al., 2024) to fine-tune and evaluate coding proficiency using the HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) datasets. We evaluate the fine-tuning performance of HD-PiSSA, compared with LoRA and PiSSA on three models: LLaMA-2-7B, LLaMA-3.1-8B, and Mistral-7B-v0.1. These three models cover common model architectures, with LLaMA-3.1-8B and Mistral-7B-v0.1 utilizing Group Query Attention (GQA) (Ainslie et al., 2023). To provide a fair comparison, we uniformly set the rank  $r$  to 16 and keep the hyper-parameters consistent. More details are included in Appendix A.4. For each model, we fine-tune with each method on two training datasets separately for one epoch and then evaluate their performance on the corresponding test sets. Results are reported in Table 1.

HD-PiSSA consistently outperforms LoRA and PiSSA on complex tasks that demand higher adaptability and rank-update, such as mathematics and code generation. As shown in Table 1, HD-PiSSA achieves the best results across all models on the GSM8K, MATH, and HumanEval datasets. On the GSM8K test set for LLaMA-2-7B, HD-PiSSA surpasses PiSSA and LoRA by 3.79% and 8.34%, respectively. Similarly, on HumanEval and MBPP for LLaMA-3.1-8B, HD-PiSSA exceeds PiSSA by nearly 3% in both cases. These improvements demonstrate that, compared to PiSSA and LoRA, HD-PiSSA effectively utilizes a similar number of

optimizer parameters per device while expanding the set of update directions. Its higher updated rank and flexible parameter space significantly enhance learning capacity, particularly for challenging fine-tuning scenarios.

## 5.2 Performance for Multi-Task Fine-Tuning

Comparison of LoRA, PiSSA, and HD-PiSSA Across 10 Tasks

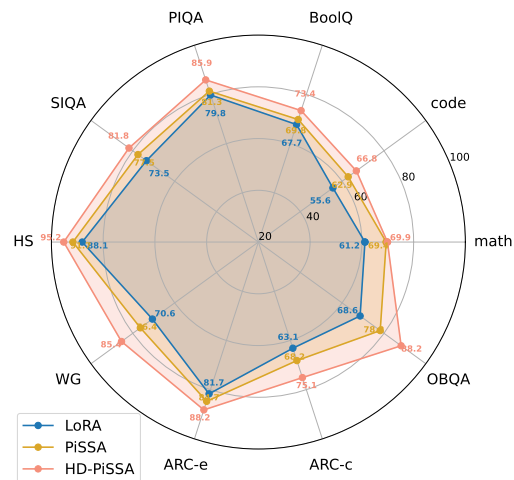


Figure 4: HD-PiSSA outperforms LoRA and PiSSA in multi-task fine-tuning setting. The absolute values of two math evaluation datasets (GSM8K, MATH) and two code datasets (MBPP, HumanEval) are summed up to keep value range consistency.

Based on our earlier analysis, we hypothesized that HD-PiSSA would demonstrate even stronger performance in more challenging multi-task fine-tuning settings due to its ability to overcome the rank limitations of LoRA and PiSSA and update parameters in a higher-dimensional space. To evaluate this hypothesis, we construct a multi-task fine-tuning dataset by combining data from four domains: mathematics, code generation, common-

Table 2: Accuracy comparison of LLaMA-2 7B with various PEFT methods on eight commonsense reasoning datasets. Results for ChatGPT, LoRA, and DoRA are sourced from Liu et al. (2024), and results for MoRA and HiRA are sourced from Anonymous (2025).

MODEL	METHOD	PARAMS	BOOLQ	PIQA	SIQA	HELLA SWAG	WINO GRANDE	ARC-E	ARC-C	OBQA	AVG.
CHATGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-2-7B	LoRA	0.83%	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA	0.84%	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
	MoRA	0.82%	72.2	80.8	79.5	29.0	80.2	85.3	71.4	81.2	72.5
	HiRA	0.83%	71.2	83.4	79.5	88.1	84.0	86.7	73.8	84.6	81.4
	HD-PiSSA	0.83%	73.7	84.6	81.1	94.0	84.7	87.5	72.4	84.2	83.4

sense reasoning, and conversation. Specifically, we use MetaMathQA, CodeFeedback, WizardLM-Evol-Instruct (Xu et al., 2024), and eight widely used commonsense reasoning datasets (Liu et al., 2024) to build a cross-domain dataset (see Appendix B). This combined dataset represents a substantially more complex fine-tuning scenario, with a diverse range of tasks requiring broad adaptability and higher learning capacity. For all PEFT methods, we use adapters with 128 ranks to fit the increased difficulty of the multi-task setting. The models are fine-tuned on LLaMA-2-7B for one epoch, and evaluations are conducted using the same benchmarks as the single-task experiments in Section 5.1, as well as the test sets of the eight commonsense reasoning datasets.

As shown in Fig. 4, HD-PiSSA consistently demonstrates significant improvements over both PiSSA and LoRA across mathematics, programming, and commonsense reasoning tasks. Especially on some commonsense datasets, such as OpenBookQA (78.2%/88.2%) and WinoGrande (76.4%/85.4%), it achieves nearly a 10% performance improvement. This indicates that HD-PiSSA’s enhanced updated rank flexibility allows it to capture and retain a greater volume of task-specific knowledge, which is essential for adapting to the diverse and complex data distributions present in multi-task fine-tuning.

We further evaluate HD-PiSSA on a widely adopted multi-task commonsense reasoning setting (Liu et al., 2024), comparing it against high-rank PEFT methods such as MoRA, HiRA, and DoRA. Training is conducted on eight standard datasets (BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, ARC-c, ARC-e, and OBQA), and evaluated on their respective test sets. To ensure fairness, we match the number of trainable parameters by tuning only the  $q\_proj$ ,  $k\_proj$ ,  $v\_proj$  and  $up$ ,  $down$  matrices for three epochs. We report results from the final model without validation-based

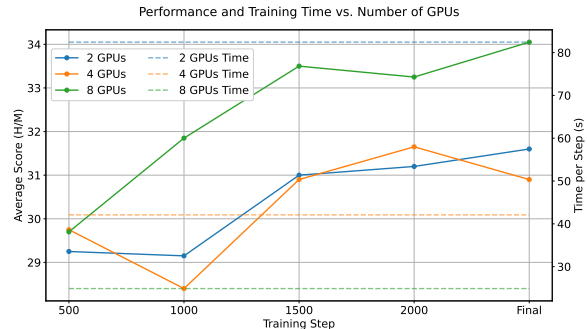


Figure 5: Average performance on code generation task and training time with different numbers of devices during distributed fine-tuning.

checkpointing. As shown in Table 2, HD-PiSSA achieves a 2.0% average gain over the previous state-of-the-art (HiRA), and consistently outperforms other high-rank approaches.

## 6 Ablation Study

### 6.1 Experiments on Different Numbers of Devices

To evaluate the distributed nature of HD-PiSSA, we conducted ablation experiments using different numbers of GPUs (2, 4, and 8) during training. All experiments were performed on the same code dataset for 3 epochs, with evaluations on HumanEval (H) and MBPP (M) conducted every 500 steps. The total batch size was fixed at 128 for all configurations to ensure comparability. Other hyperparameters remained unchanged.

The motivation behind this experiment is rooted in HD-PiSSA’s design: as the number of devices increases, so does the number of orthogonal adapters and their corresponding data partitions. This enhanced modularization is expected to improve both training efficiency and performance.

As shown in Fig. 5, increasing the number of GPUs significantly reduces training time per step (from 82.44s to 24.88s) while also leading to im-

proved final performance. With 8 devices, HD-PiSSA achieves the best average score (34.05), showing that the distributed modularity inherent in the method benefits both scalability and quality of learned representations. This confirms the strength of HD-PiSSA in leveraging increased computational resources effectively.

## 6.2 Experiments on Various Ranks

We study how different ranks affect HD-PiSSA’s performance using LLaMA-2-7B and 100K samples from MetaMathQA. Evaluations on GSM8K and MATH (Fig. 6) show that HD-PiSSA generally outperforms LoRA and PiSSA, especially at higher ranks. On MATH, HD-PiSSA approaches Full Fine-Tuning performance with only a 0.62% gap at rank 256.

When comparing HD-PiSSA with PiSSA, we observe distinct patterns across datasets. On GSM8K, HD-PiSSA performs better than PiSSA in the 2-32 rank range. On MATH, however, HD-PiSSA’s advantage continues to grow beyond rank 8. This aligns with the fact that MATH is a more complex dataset requiring more updates to learn effectively, while GSM8K is simpler and benefits from fewer updates. Consequently, the difference between the methods is less pronounced at lower ranks on MATH. These results show that HD-PiSSA can achieve near-FFT performance with much lower ranks than PiSSA, demonstrating its efficiency in handling complex tasks with fewer updates.

## 6.3 Performances with Different Mute Scalars

We also evaluate the impact of different mute scalar values on performance. We tested several mute scalar values, including  $1 \times 10^{-4}$ ,  $1 \times 10^{-8}$ ,  $1 \times 10^{-16}$ ,  $1 \times 10^{-2}$ , and  $1 \times 10^{-32}$ . Specifically, we finetune LLaMA-2-7B with these mute scalars on MetaMathQA for one epoch, then evaluate the performance on the GSM8K test set. As shown in Fig. 7, performance remained consistent for mute scalar values of  $1 \times 10^{-4}$ ,  $1 \times 10^{-8}$ , and  $1 \times 10^{-16}$ . However, when the mute scalar was set to  $1 \times 10^{-2}$  or  $1 \times 10^{-32}$ , performance decreased significantly.

The likely cause of this behavior is that if the mute scalar is relatively large, the adapter’s contributions during forward propagation are not fully eliminated, which affects the output  $y$ . On the other hand, setting the scalar too small might result in reaching the lower limit of float32 precision, causing inaccuracies in gradient updates.

## 6.4 More Ablation Study

We conducted future ablation studies to verify the effectiveness of Direct Weight Update (DWU) and Orthogonal Adapter Initialization (OAI). Specifically, we included: (1) **LoRA init. (with DWU)**: A variant using standard LoRA initialization with the Direct Weight Update technique. (2) **Top-Rank HD-PiSSA (w/o OAI)**: HD-PiSSA without orthogonal adapters, using PiSSA adapter instead. All variants were fine-tuned on the code dataset for 3 epochs. Evaluation was conducted on HumanEval and MBPP (abbreviated as H&M in the table) every 500 steps for detailed training-time insights. The total batch size and training configuration were kept consistent across methods.

Analyzing the results in Table 3, we conclude that HD-PiSSA’s superior performance is due to the synergy of DWU and OAI. First, the use of DWU clearly provides a measurable gain, as seen in the improved performance of LoRA+DWU compared to standard baselines. This aligns with findings in prior work such as AdaLoRA. However, even with DWU, both LoRA and PiSSA-based variants lag behind HD-PiSSA in terms of average loss and final evaluation metrics.

We also test HD-PiSSA’s training overhead compared with PiSSA, and found that HD-PiSSA’s time and memory costs slightly increase. Details in the Appendix C

## 7 Analysis of Effective Update Rank

We provide an intuitive explanation for why HD-PiSSA enables richer and higher-rank updates than traditional PEFT methods like LoRA and PiSSA.

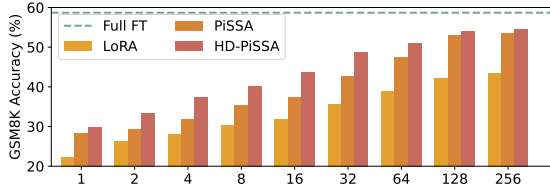
**Rank Expansion via Orthogonal Adapters.** Each device hosts an orthogonal adapter of rank  $r$ , and its update takes the form:

$$\Delta W^t = \frac{1}{K} \sum_{i=0}^{K-1} (\Delta A_i^t B_i + A_i \Delta B_i^t + \Delta A_i^t \Delta B_i^t).$$

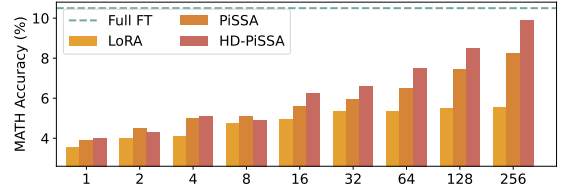
Assuming the adapter matrices  $A_i$  and  $B_i$  are orthogonal and uncorrelated, the first-order terms alone contribute an approximate update rank of  $2Kr$ , significantly higher than the single-device LoRA/PiSSA rank of  $r$ .

**Going Beyond 2Kr.** The second-order term  $\Delta A_i^t \Delta B_i^t$ , though small, introduces updates in additional directions not captured by the main components. Since  $\Delta A$  and  $\Delta B$  evolve during training, they can rotate the updates into less-explored





(a): Accuracy on GSM8K under various ranks.



(b): Accuracy on MATH under various ranks.

Figure 6: Performance comparison of LoRA, PiSSA, and HD-PiSSA across different adapter ranks (1 to 256). The dashed line represents the accuracy of Full Fine-Tuning (FFT) as a reference. The results for LoRA and PiSSA within 1-128 rank range are taken from Meng et al. (2024), with FFT performed using the Float32 data type rather than the BF16 data type used in their original FFT results.

Table 3: Ablation study on Direct Weight Update (DWU) and Orthogonal Adapter Initialization (OAI). Results shown as HumanEval / MBPP / Average (H/M/A).

Components / Steps	Avg. Loss	500	1000	1500	2000	Final (2323)
LoRA init. (w. DWU)	0.6035	21.3/37.3/29.3	23.2/34.9/29.05	25.6/40.2/32.9	27.4/39.4/33.4	26.2/39.9/33.05
Top-Rank HD-PiSSA (w/o OAI)	0.5984	20.1/37.3/28.7	23.8/37.0/30.4	25.6/41.0/33.3	26.8/37.6/32.2	28.0/38.4/33.2
HD-PiSSA	<b>0.5854</b>	19.5/39.9/29.7	25.6/38.1/31.85	26.8/40.2/33.5	26.8/39.7/33.25	27.4/40.7/34.05

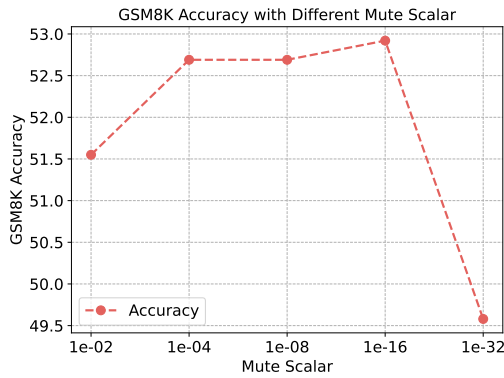


Figure 7: Performances on GSM8K with different mute scalars.

regions of the parameter space. As a result, HD-PiSSA’s updates extend beyond the top- $2Kr$  components. This is empirically validated in Fig. 8, where the singular values of HD-PiSSA remain 1–2 orders of magnitude larger than LoRA and PiSSA beyond the  $2Kr$  threshold—indicating stronger coverage of the parameter space and more expressive adaptation capability.

## 8 Conclusion

We propose **HD-PiSSA**, a parameter-efficient fine-tuning method that overcomes the rank limitations of LoRA and PiSSA while maintaining the same memory cost. By distributing orthogonal adapters across GPUs and applying Direct Weight Update, HD-PiSSA significantly expands the update rank and achieves performance close to Full

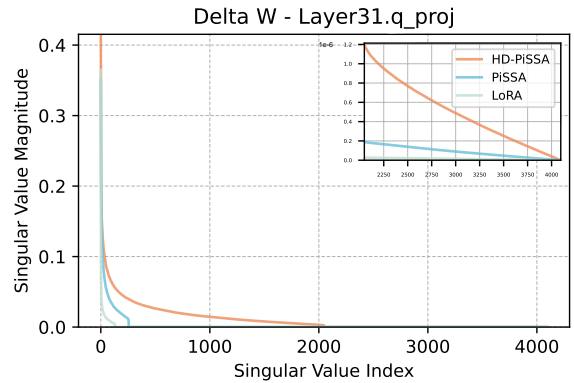


Figure 8: Comparison of singular value magnitudes for the update matrices of LoRA ( $r = 128$ ), PiSSA ( $r = 128$ ), and HD-PiSSA ( $r = 128$ ). The sub-figure in the top-right corner provides a magnified view of the singular values for indices beyond  $2Kr$ . This highlights that the singular values for HD-PiSSA remain 1 to 2 orders of magnitude larger than those of LoRA and PiSSA in this high-rank region, empirically supporting that HD-PiSSA achieves update rank beyond  $2Kr$ .

Fine-Tuning. The Muting Mechanism further improves our method by eliminating the need to maintain the residual matrices as PiSSA. Experiments on mathematics, code generation, and multi-task learning show consistent improvements over existing PEFT methods. Ablation studies further validate the effectiveness of our design choices. HD-PiSSA provides a scalable and efficient fine-tuning strategy for LLMs, making it a strong alternative to Full Fine-Tuning.

## 9 Limitations

HD-PiSSA requires multi-GPU distributed setups to fully realize its benefits, which may restrict its applicability in resource-constrained environments. Besides, because HD-PiSSA aggregates all the gradient updates on the Pretrained Matrix W, its performance decreases when the base model is loaded at lower precision. Additionally, while demonstrating strong results on various tasks, its generalization to broader domains and modalities remains to be validated. We also plan to conduct more rigorous theoretical analysis on the effective update rank as part of future work.

## References

- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.
- Anonymous. 2025. [HiRA: Parameter-efficient hadamard high-rank adaptation for large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.
- Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient finetuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.
- Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. Flora: Low-rank adapters are secretly gradient compressors. *arXiv preprint arXiv:2402.03293*.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, and 1 others. 2024. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.
- Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

- Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.
- Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.
- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialliqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: an instruction-following llama model (2023). *URL [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca)*, 1(9).
- Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023. Orthogonal subspace learning for language model continual learning. *arXiv preprint arXiv:2310.14152*.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.
- Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. *arXiv preprint arXiv:2109.05687*.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.
- Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.
- Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhui Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.
- Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. 2023. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*.

## A Training Details

### A.1 Computing Infrastructure

All experiments were conducted on NVIDIA A100 GPUs with 40GB or 80GB memory configurations. Running the full set of main experiments, including all primary tables, required approximately 21 days using 8 GPUs in parallel.

### A.2 Implementation details of the experiment in Figure 1

**Model:** LLaMA-2-7B

**Optimizer:** AdamW, learning rate =  $2e-5$ , warm-up ratio = 0.03

**Epochs:** 1

**Batch size:** 128

**Visualized layer:** Layer 31, Query projection (q\_proj)

### A.3 Implementation details of the experiment in Figure 3

**Model:** LLaMA-3.1-8B

**Training set:** MetaMathQA

**Optimizer:** AdamW, learning rate =  $2e-5$ , warm-up ratio = 0.03

**Epochs:** 1

**Batch size:** 128

**Ranks Per Device For LoRA, PiSSA and HD-PiSSA:** 128

**Visualized layer:** Layer 31, Key projection (k\_proj)

### A.4 Training Setting in Main Experiments

Details are presented in Table 5. Results are obtained from a single time running. Our evaluation codebase follows PiSSA’s open-source evaluation codebase.

### A.5 Hyperparameters Configuration in Commonsense Reasoning Experiment

Table 6: Hyperparameters for HD-PiSSA

Hyperparameter	HD-PiSSA
Rank $r$	32
$\alpha$	64
Dropout	0.05
Optimizer	AdamW
Learning Rate (LR)	$2e-5$
LR Scheduler	Linear
Batch Size	16
Warmup Steps	100
Epochs	1
Matrices	Q, K, V, Up, Down

## B Multi-task Fine-Tuning Dataset

The details of the multi-task dataset are provided in Table 7.

## C Speed Comparison

To evaluate the training overhead of our proposed HD-PiSSA, we conducted a comparative analysis against PiSSA. The experiment was performed on the MetaMath dataset using a LLaMA-2-7B model, distributed across 8 GPUs with 16 ranks per GPU, and using bf16 precision without any advanced memory and acceleration optimizations (our implementation of parallelization and communication).

As detailed in **Table 8**, HD-PiSSA exhibits a slight increase in both peak memory usage per GPU (49,176 MiB vs. 47,290 MiB) and total training time for 100 steps (1995.8s vs. 1808.31s). We attribute this marginal overhead to the additional communication and computation required for updating the full-parameter matrix  $W$ . For instance, each GPU is required to gather the previous values and gradients of matrices  $A$  and  $B$  from all other GPUs, followed by computing the corresponding update to  $W$ . Nevertheless, we consider this overhead to be acceptable given the substantial gains in performance demonstrated by HD-PiSSA.



Table 4: Details of the datasets used for training and evaluation.

<b>Dataset</b>	<b>Description</b>
<b>math</b>	MetaMathQA
<b>code</b>	CodeFeedback
<b>commonsense</b>	BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, Arc-easy, Arc-challenge, OBQA
<b>conversation</b>	WizardLM-Evol-Instruct

Table 5: Hyperparameter configurations used in all experiments.

<b>Hyperparameter</b>	<b>Value</b>
<b>Optimizer</b>	AdamW
<b>Batch size</b>	128
<b>Learning rate (LLaMA models)</b>	$2 \times 10^{-5}$
<b>Learning rate (Mistral-7B-v0.1)</b>	$5 \times 10^{-6}$
<b>Scheduler</b>	Cosine annealing
<b>Warmup ratio</b>	0.03
<b>Weight decay</b>	0.0
<b>LoRA alpha</b>	Equal to LoRA rank
<b>LoRA dropout</b>	0
<b>Adapter insertion</b>	Applied to all linear layers
<b>Base Model Precision</b>	float32

Table 7: Details of the datasets used in multi-task fine-tuning.

<b>Dataset Name</b>	<b>Domain</b>	<b>Number of Training Samples</b>
MetaMathQA (Yu et al., 2023)	Mathematics	395K
CodeFeedback (Zheng et al., 2024)	Code Generation	105K
WizardLM-Evol-Instruct (Xu et al., 2024)	Conversation	143K
BoolQ (Clark et al., 2019)	yes/no QA	9.4K
PIQA (Bisk et al., 2020)	physical commonsense	16.1K
SIQA (Sap et al., 2019)	social reasoning	33.4K
HellaSwag (Zellers et al., 2019)	commonsense NLI	39.9K
WinoGrande (Sakaguchi et al., 2021)	fill-in-the-blank	40.4K
ARC_Challenge (Clark et al., 2018)	multiple-choice science questions	1.1K
ARC_Easy (Clark et al., 2018)	multiple-choice science questions	2.3K
OpenBookQA (Mihaylov et al., 2018)	multi-step reasoning	5.0K

Table 8: Training overhead comparison between PiSSA and HD-PiSSA on LLaMA-2-7B using 8 GPUs. Both methods are run for 100 steps on the MetaMath dataset without advanced memory or acceleration optimizations.

<b>Metric</b>	<b>PiSSA</b>	<b>HD-PiSSA</b>
Peak Mem.	47,290 MiB	49,176 MiB
Time (100 steps)	1808.31 s	1995.8 s