NLoRA: Nyström-Initiated Low-Rank Adaptation for Large Language Models

Anonymous ACL submission

Abstract

Parameter-efficient fine-tuning (PEFT) is essential for adapting large language models (LLMs), with low rank adaptation (LoRA) being the most popular approach. However, LoRA suffers from slow convergence, and some recent LoRA variants, such as PiSSA, primarily rely on Singular Value Decomposition (SVD) for initialization, leading to expensive computation. To mitigate these problems, we resort to Nyström method, which follows a three-matrix manipulation. Therefore, we first introduce Structured**LoRA** (SLoRA), investigating to introduce a small intermediate matrix between the low-rank matrices A and B. Secondly, we propose Nyström**LoRA** (NLoRA), which leverages Nyström-based initialization for SLoRA to improve its effectiveness and efficiency. Finally, we propose **Int**ermediate**Tune** (IntTune) to explore fine-tuning exclusively the intermediate matrix of NLoRA to furthermore boost LLMs' efficiency. We evaluate our methods on 5 natural language generation (NLG) tasks and 8 natural language understanding (NLU) tasks. On GSM8K, SLoRA and NLoRA achieve accuracies of 56.48% and 57.70%, surpassing LoRA by 33.52% and 36.41% with only 3.67M additional trainable parameters. IntTune boosts average NLG performance over LoRA by 7.45% while using only 1.25% of its parameters. These results demonstrate the efficiency and effectiveness of our approach in enhancing model performance with minimal parameter overhead.

1 Introduction

Fine-tuning large language models (LLMs) has emerged as a fundamental approach to enhancing model capabilities (Yu et al., 2023; Li et al., 2023; Xia et al., 2024) and aligning models with specific application requirements (Zheng et al., 2023; Ouyang et al., 2022). However, the growing scale of LLMs introduces significant challenges to LLM

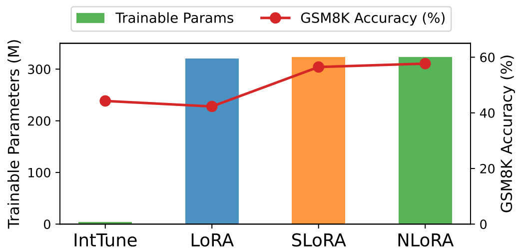


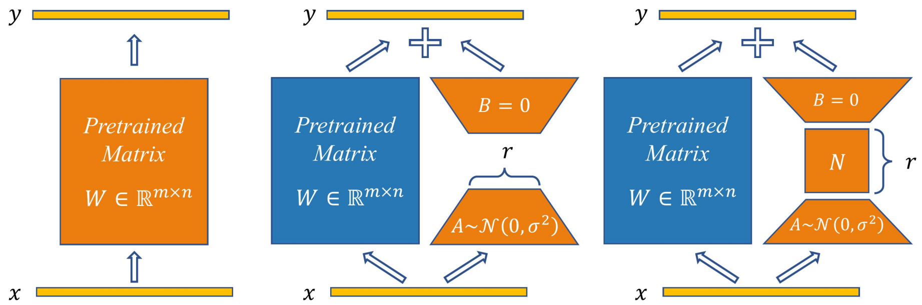
Figure 1: The comparison among LoRA and our models

development, with fine-tuning requiring substantial computational and memory resources (Hu et al., 2021; Chang et al., 2024). For example, fine-tuning a LLaMA-65B model requires more than 780 GB of GPU memory (Dettmers et al., 2023), while training GPT-3 175B requires 1.2 TB of VRAM (Hu et al., 2021). Such resource-intensive processes are infeasible for many researchers and institutions, driving the development of parameter-efficient fine-tuning (PEFT) methods. Among these methods, Low-Rank Adaptation (LoRA) (Hu et al., 2021) has received widespread attention due to its ability to achieve competitive performance compared to full parameter fine-tuning, while significantly reducing memory consumption and avoiding additional inference latency.

LoRA enables the indirect training of dense layers in a neural network by optimizing low-rank decomposition matrices that represent changes in the dense layers during adaptation, all while keeping the pre-trained weights fixed. For a pre-trained weight matrix $W \in \mathbb{R}^{m\times n}$, LoRA introduces a low-rank decomposition $\Delta W = AB$, where $A \in \mathbb{R}^{m\times r}$, $B \in \mathbb{R}^{r\times n}$, and the rank $r \ll \min(m, n)$. This modifies the forward pass of a layer as follows:

$$Y = X(W + \Delta W) = X(W + AB), \quad (1)$$

where $X \in \mathbb{R}^{b\times s\times m}$, $Y \in \mathbb{R}^{b\times s\times n}$, and b represents the batch size, s represents the sequence



(a) Full Fine-tuning (b) LoRA (c) SLoRA

Figure 2: The comparison among Full Fine-tuning, LoRA, and SLoRA

length. For initialization, A is randomly initialized with Gaussian values and B is set to zero, ensuring that injection of the low-rank adaptation does not alter the model predictions at the start of training. Unlike traditional fine-tuning methods that require updating and storing gradients for the full weight matrix W, LoRA optimizes only the smaller matrices A and B, significantly reducing the number of trainable parameters and memory usage. Furthermore, LoRA often achieves performance comparable or superior to full fine-tuning, demonstrating that adapting only a small subset of parameters suffices for many downstream tasks.

Despite the above benefits, LoRA suffers from slow convergence (Ding et al., 2023). To address this issue, some recent LoRA variants, such as PiSSA (Meng et al., 2024), choose to conduct initialization of the low rank matrices by using Singular Value Decomposition (SVD). However, SVD-based initialization is computationally expensive and requires a long time. To mitigate this issue, we investigate using Nyström method, which approximates a matrix as a product of three matrices, to approximate SVD. To fit the three-matrix structure, we first propose Structured**LoRA** (SLoRA), where an additional $r \times r$ matrix is inserted between the low-rank matrices A and B, as shown in Figure 2. Furthermore, we explore whether an extra matrix can influence the language model's performance, experimental results indicate that SLoRA effectively enhances performance with only a minor increase in the number of parameters, demonstrating the potential of the three-matrix structure for PEFT.

Secondly, inspired by NyströmFormer (Xiong et al., 2021), we proposed Nyström**LoRA** (NLoRA) to leverage Nyström method, which conducts SVD approximation by sampling a subset of rows and columns of the pre-trained parameter matrix to reduce the computational cost, for weight initialization. NLoRA is supposed to bypass the computational cost of SVD's eigenvalue decomposition, reducing time complexity to $O(mr + r^2 + rn)$ compared to the $O(mn^2)$ complexity of SVD-based methods.

Finally, to explore whether we can further compress the trainable parameters of NLoRA, we propose **Int**ermediate**Tune** (IntTune), which exclusively adjusts the intermediate matrix of NLoRA. This method significantly reduces the number of trainable parameters. Specifically, on the evaluation of LLaMA 2-7B across five NLG benchmarks, LoRA uses 320M parameters, while our IntTune method only requires tuning 4M parameters. In the meantime, IntTune outperforms LoRA by 7.45% on average across NLG benchmarks. The comparison of our proposed methods with LoRA in terms of performance and trainable parameters is illustrated in Figure 1.

In summary, our contributions are as follows:

1. We propose SLoRA, an extension to the LoRA framework, incorporating an additional intermediate matrix to enhance model expressiveness, achieving improved performance with minimal parameter overhead.

2. We introduce NLoRA, leveraging Nyström approximation for efficient and effective initialization, particularly excelling in natural language generation (NLG) and natural language understanding (NLU) tasks.

3. We propose IntTune to fulfill supervised fine-tuning (SFT) LLaMA 2-7B by tuning 4M parameters, achieving superior performance compared to LoRA on average, offering a

lightweight and efficient alternative for SFT LLMs in resource-constrained scenarios.

2 Related Works

2.1 LoRA's variants and related extensions

With the introduction of LoRA (Hu et al., 2021), many derivative methods have emerged. AdaLoRA (Zhang et al., 2023) proposes an adaptive rank allocation strategy based on parameter importance to improve fine-tuning efficiency. DoRA (Liu et al., 2024) introduces a decomposation of weight matrices into magnitude and direction components, leveraging LoRA to update only the directional component. ReLoRA (Lialin et al., 2023) achieves high-rank training through iterative low-rank updates, periodically merging parameters into the main model. LoRA+ (Hayou et al., 2024) further improves efficiency by applying different learning rates to the two matrices in LoRA. Other works have focused on improving the initialization of the AB matrix, such as PiSSA (Meng et al., 2024), which suggests initializing A and B by performing SVD on the pre-trained matrix W to accelerate the convergence speed. LoRA-GA (Wang et al., 2024) initializes A and B using the eigenvectors of the full-gradient matrix, aligning the gradient direction of the low-rank product BA with the gradient direction of the pretrained weight matrix W. A related work is LaMDA (Azizi et al., 2024), which also introduces an intermediate matrix. However, LaMDA relies on SVD-based initialization and primarily focuses on memory efficiency. In contrast, our method adopts Nystrom-based initialization, which not only significantly shortens the initialization time but also achieves strong performance with fewer parameters, offering advantages in both computational and memory efficiency.

Beyond these LoRA variants, some works have explored integrating LoRA into more efficient model architectures. MixLoRA (Li et al., 2024) and MLoRA (Yang et al., 2024) extend LoRA to multi-task and multi-domain settings. MixLoRA builds a sparse MoE with multiple LoRA experts, while MLoRA uses domain-specific LoRA modules for CTR prediction, both improving performance with efficient resource use. Besides, m-LoRA (Ye et al., 2023) introduces a LoRA-aware pipeline parallelism scheme to efficiently fine-tune multiple LoRA tasks across GPUs and machines, significantly reducing fine-tuning time and improving GPU utilization.

2.2 Nyström-like methods

Nyström-like methods approximate matrices by sampling a subset of columns, a technique widely used in kernel matrix approximation (Baker and Taylor, 1979; Williams and Seeger, 2000). Numerous variants have been proposed to enhance the basic Nyström method, including Nyström with k-means clustering (Wang et al., 2019), Nyström with spectral problems (Vladymyrov and Carreira-Perpinan, 2016), randomized Nyström (Li et al., 2010; Persson et al., 2024), ensemble Nyström method (Kumar et al., 2009), fast-Nys (Si et al., 2016).

The Nyström method has also been extended to general matrix approximation beyond symmetric matrices (Nemtsov et al., 2016). Some methods (Wang and Zhang, 2013; Xiong et al., 2021) explicitly address general matrix approximation by sampling both rows and columns to reconstruct the full matrix. Inspired by such strategies, we propose NLoRA method by to optimize the approximation for efficient matrix reconstruction.

3 Method

The Nyström method (Baker and Taylor, 1979), originating from the field of integral equations, is a approach for discretizing integral equations using a quadrature technique. It is commonly employed for out-of-sample extension problems. Specifically, given an eigenfunction problem of the form:

$$\lambda f(x) = \int_a^b M(x, y) f(y)\, dy, \tag{2}$$

the Nyström method utilizes a set of s sample points $y_1, y_2, \ldots, y_s$ to approximate $f(x)$ as follows:

$$\lambda \tilde{f}(x) \triangleq \frac{b-a}{s} \sum_{j=1}^{s} M(x, y_j) f(y_j). \tag{3}$$

This approach effectively converts the continuous integral equation into a discrete summation, facilitating numerical computation and enabling out-of-sample extensions.

For the pre-trained matrix $W \in \mathbb{R}^{m \times n}$, we assume that it can be decomposed as follows:

$$W = \begin{bmatrix} A_W & B_W \\ F_W & C_W \end{bmatrix}, \tag{4}$$

where, $A_W \in \mathbb{R}^{r \times r}$ is designated to be our sample matrix, $B_W \in \mathbb{R}^{r \times (n-r)}$ and $F_W \in \mathbb{R}^{(m-r) \times r}$ represent the remaining sampled column and row components, respectively, and $C_W \in \mathbb{R}^{(m-r) \times (n-r)}$

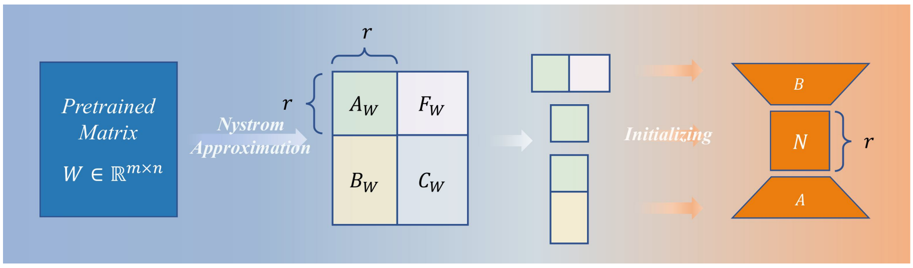


Figure 3: The diagram of the Nyström-based initialization

corresponds to the remainder of the matrix W. The matrix W can be efficiently approximated using the Nyström method's basic quadrature technique. Starting with the singular value decomposition (SVD) of the sample matrix A_W, represented as $A_W = U\Lambda H^T$, where $U, H \in \mathbb{R}^{r\times r}$ are unitary matrices and $\Lambda \in \mathbb{R}^{r\times r}$ is diagonal. The Nyström approximation reconstructs W based on the out-of-sample approximation strategy (Nemtsov et al., 2016). This strategy utilizes the entries of F_W and B_W as interpolation weights for extending the singular vector, resulting in the full approximations of the left and right singular vectors of W:

$$\hat{U} = \begin{bmatrix} U \\ F_W H \Lambda^{-1} \end{bmatrix}, \quad \hat{H} = \begin{bmatrix} H \\ B_W^T U \Lambda^{-1} \end{bmatrix}, \tag{5}$$

Using the Nyström method, the pretrained matrix W can be approximated as:

$$\begin{aligned} \widehat{W} = \hat{U}\Lambda\hat{H}^T &= \begin{bmatrix} A_W & B_W \\ F_W & F_W A_W^+ B_W \end{bmatrix} \\ &= \begin{bmatrix} A_W \\ F_W \end{bmatrix} A_W^+ \begin{bmatrix} A_W & B_W \end{bmatrix}, \end{aligned} \tag{6}$$

where A_W^+ is the Moore-Penrose pseudoinverse of the sampled core matrix A_W. The remaining block C_W is approximated as $F_W A_W^+ B_W$. This approximation demonstrates that W can be effectively reconstructed using only A_W, B_W, and F_W, significantly reducing computational complexity. For the detailed derivation, please refer to Appendix A.

In this way, the matrix W can be approximated as the product of three matrices. Based on this finding, we propose an improvement to LoRA by introducing an intermediate matrix, named as **S**tructured**LoRA** (SLoRA). Specifically, we introduce an intermediate matrix $N \in \mathbb{R}^{r\times r}$ between the low-rank matrices A and B, as illustrated in Figure 2. This modification transforms the weight update into:

$$\Delta W = ANB, \tag{7}$$

where $A \in \mathbb{R}^{m\times r}$, $B \in \mathbb{R}^{r\times n}$, $N \in \mathbb{R}^{r\times r}$, and $r \ll \min(m, n)$.

Building on the three-matrix structure, we further enhance SLoRA's effectiveness by employing a Nyström-based initialization. Specifically, by sampling r rows and r columns—corresponding to the rank of LoRA—we efficiently approximate W through matrix decomposition. The resulting submatrices are then directly utilized to initialize the three components of SLoRA, specifically:

- The component $\begin{bmatrix} A_W \\ F_W \end{bmatrix}$ is used to initialize the matrix A in SLoRA.
- The component A_W^+, representing the Moore-Penrose pseudoinverse of A_W, is used to initialize the matrix N in SLoRA.
- The component $\begin{bmatrix} A_W & B_W \end{bmatrix}$ is used to initialize the matrix B in SLoRA.

While the pseudoinverse can be computed using singular value decomposition (SVD), the process is computationally inefficient on GPUs. To overcome this challenge, we simplify the initialization by directly employing A_W instead of its pseudoinverse, thereby reducing computational overhead while preserving the effectiveness of the initialization. The diagram of the Nyström-based initialization is shown in Figure 3.

By employing this decomposition based on the Nyström approximation method, we propose an initialization strategy for SLoRA, which we term as Nyström**LoRA** (NLoRA). Additionally, we explore fine-tuning only the intermediate matrix while keeping the other two matrices fixed, which we term **Int**ermediate**Tune** (IntTune).

4 Experiments

The experiments were performed on NVIDIA L20 GPUs. For these experiments, we follow the experimental setting given by (Meng et al., 2024), we

Model	Strategy	Parameters	GSM8K	MATH	HumanEval	MBPP	MT-Bench
LLaMA 2-7B	Full FT	6738M	49.05	7.22	21.34	35.59	**4.91**
	LoRA	320M	42.30	5.50	18.29	35.34	4.58
	PiSSA	320M	53.07	7.44	21.95	37.09	4.87
	SLoRA	323M	56.48	**10.68**	23.78	42.32	4.85
	NLoRA	323M	**57.70**	9.94	**25.00**	**43.12**	4.82
Mistral-7B	Full FT	7242M	67.02	18.6	45.12	51.38	4.95
	LoRA	168M	67.70	19.68	43.90	58.39	4.90
	PiSSA	168M	72.86	21.54	46.95	**62.66**	**5.34**
	SLoRA	169M	73.01	21.88	**47.6**	60.3	5.12
	NLoRA	169M	**73.92**	**22.00**	44.5	60.3	5.21

Table 1: Experimental results on NLG tasks

Strategy	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
DeBERTa-v3-base								
Full FT	89.90	95.63	89.46	69.19	94.03	**92.40**	83.75	91.60
LoRA	90.65	94.95	89.95	69.82	93.87	91.99	85.20	91.60
PiSSA	90.43	95.87	91.67	72.64	94.29	92.26	87.00	91.88
SLoRA	90.43	96.10	**91.91**	70.82	93.94	92.11	**88.09**	91.86
NLoRA	**90.74**	**96.22**	**91.91**	**73.41**	**94.45**	92.03	**88.09**	**92.14**
RoBERTa-large								
Full FT	90.2	96.4	90.9	68.0	94.7	**92.2**	86.6	91.5
LoRA	90.6	96.2	90.9	68.2	94.9	91.6	87.4	92.6
PiSSA	90.7	96.7	**91.9**	69.0	95.1	91.6	**91.0**	**92.9**
SLoRA	**90.8**	**96.8**	91.7	68.5	94.9	91.6	90.3	92.7
NLoRA	90.7	96.6	**91.9**	**69.7**	**95.2**	91.6	90.3	92.7

Table 2: Experimental results on NLU tasks

employ the AdamW optimizer with a batch size of 4, a learning rate of 2E-4, and a cosine annealing schedule with a warmup ratio of 0.03, all while avoiding weight decay. The parameter lora_alpha is consistently set equal to lora_r, with lora_dropout fixed at 0. Adapters are integrated into all linear layers of the base model, and both the base model and adapters utilized Float32 precision for computation. We take the convenience to directly cite the baseline performance values from (Meng et al., 2024).

In this section, we evaluate the performance of SLoRA and NLoRA across various benchmark datasets. We compare them with the following baselines: (1) Full Fine-tune, which updates all model parameters; (2) LoRA (Hu et al., 2021), which approximates weight updates with low-rank matrices while freezing the base model; and (3) PiSSA (Meng et al., 2024), which initializes adapters using principal singular components and freezes residuals while retain LoRA's architecture.

We evaluate the capabilities of natural language generation (NLG) using the LLaMA 2-7B (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023) models through mathematical reasoning, coding proficiency, and dialogue tasks. Additionally, natural language understanding (NLU) tasks were evaluated using the GLUE dataset (Wang, 2018) with DeBERTa-v3-base (He et al., 2021) and RoBERTa-large (Liu, 2019). Finally, we analyze the empirical effects of exclusively fine-tuning the intermediate matrix on both NLU and NLG tasks.

4.1 Experiments on Natural Language Generation

We conduct experiments using LLaMA 2-7B and Mistral-7B-v0.1. To evaluate mathematical reasoning abilities, we perform fine-tuning using the MetaMathQA dataset and evaluated their performance on GSM8K (Cobbe et al., 2021) and MATH (Yu et al., 2023). In terms of coding capability, we perform fine-tuning on the CodeFeedback dataset (Zheng et al., 2024) and evaluated them using the HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) benchmarks. To measure session capabilities, the model is fine-tuned on the WizardLM-Evol-Instruct dataset (Xu et al., 2024) and tested using the MT-Bench dataset (Zheng et al., 2023).

Strategy	Parameters	GSM8K	MATH	HumanEval	MBPP	MT-Bench
LoRA	320M	42.30	5.50	18.29	35.34	4.58
IntTune	4M	44.28	6.86	20.70	34.40	4.46

Table 3: IntTune performance on NLG tasks

Strategy	Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
LoRA	1.33M	90.65	94.95	89.95	69.82	93.87	91.99	85.20	91.60
IntTune	3.07K	81.93	92.20	85.29	65.38	89.13	85.18	76.90	88.37

Table 4: IntTune performance on NLU tasks

All experiments use a subset of 100K data points.

As shown in Table 1, SLoRA consistently outperforms LoRA, which is labeled with a blue background in Table 1, and even outperforms PiSSA in most tasks. In most cases, NLoRA further enhances the performance of SLoRA. Both methods maintain high parameter efficiency, with only slight increases in trainable parameters (1.15% for LLaMA 2-7B and 0.55% for Mistral-7B compared to LoRA), yet deliver significant performance gains. On these two models, SLoRA achieves average improvements of 38.68%, 15.37%, and 5.19% in mathematical reasoning, coding, and conversational tasks, respectively, relative to LoRA's performance, while NLoRA achieves improvements of 34.53%, 15.83%, and 5.78% over LoRA.

Although the addition of intermediate matrices results in additional matrix multiplication operations, the time overhead increases only slightly compared to LoRA. In the MetaMathQA dataset, the training time for SLoRA increases to 27,690.03 seconds, which is an increase of 10.13% compared to LoRA (25142.26 seconds). The training time for NLoRA increases to 25,323.34 seconds, which is almost identical to LoRA's training time. As for initialization time, SLoRA incurs only an 11.95% increase in initialization time compared to LoRA, while NLoRA adds just 12.66 seconds. Both are significantly lower than the time cost of PiSSA. Subsequently, we further discuss the effects under different ranks (Section 4.4), learning rates (Appendix C), and optimizers (Appendix D).

4.2 Experiments on Natural Language Understanding

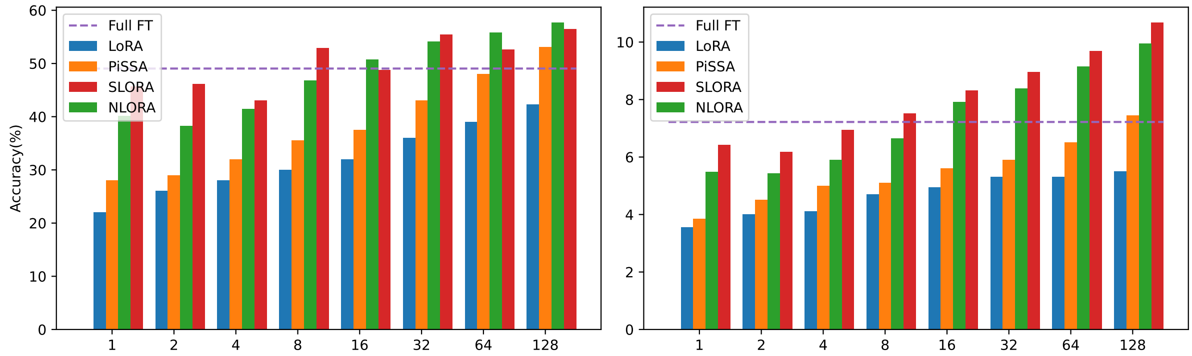
We also assess the NLU capabilities of RoBERTa-large and DeBERTa-v3-base on the GLUE benchmark. Table 2 summarizes the results of eight tasks performed using these two base models.

SLoRA demonstrates consistent improvements over the baseline LoRA across all tasks, as highlighted in blue. In addition, SLoRA surpasses PiSSA in several cases, showcasing the potential of incorporating an intermediate matrix in LoRA. NLoRA further enhances the performance of SLoRA in most tasks, achieving superior results in tasks such as QNLI, MRPC, and CoLA. For instances where NLoRA does not outperform PiSSA, NLoRA consistently achieves a lower training loss in these scenarios, suggesting its potential for further optimization and efficient fine-tuning. Details can be found in Appendix E.

4.3 NLoRA's Intermediate Matrix Fine-Tuning: A Minimalist Approach

To further improve the computational efficiency of NLoRA, we try to investigate reducing its trainable parameters without sacrificing much performance. Therefore, we propose **Int**ermediate**Tune** (IntTune), which exclusively fine-tune the intermediate matrix in SFT. To validate the effectiveness of IntTune, we conduct experiments using LLaMA-2-7B and DeBERTa-v3-base for NLG and NLU tasks, respectively. For NLG tasks, we set the learning rate to 2E-3 while keeping other settings unchanged. For NLU tasks, the specific parameter settings are detailed in Appendix E. The results are shown in Table 3 and Table 4.

For NLG tasks, IntTune achieves competitive performance, surpassing LoRA on the GSM8K, MATH, and HumanEval tasks, and attaining comparable results on MBPP and MT-Bench. Overall, the average performance of IntTune across all tasks exceeds that of LoRA, surpassing LoRA's average performance by 7.45%. In terms of computational efficiency, IntTune significantly reduces the num-



(a) GSM8K accuracy under various ranks (b) MATH accuracy under various ranks

Figure 4: Compare the performance of different ranks for NLoRA on NLG tasks

Strategy	rank=1	rank=2	rank=4	rank=8	rank=16	rank=32	rank=64	rank=128
LoRA	0.22	0.24	3.87	3.10	14.62	12.76	14.70	13.92
PiSSA	8124.14	7980.15	8078.53	7723.19	8141.76	8043.79	8044.24	8068.20
SLoRA	0.26	0.28	3.54	3.02	6.42	15.74	11.29	14.97
NLoRA	5.76	4.53	6.00	7.27	21.37	24.12	25.25	25.32

Table 5: Compare the initialization time(s) of different ranks for NLoRA and LoRA

ber of trainable parameters to 4M, accounting for only 0.05% of the total model parameters and just 1.13% of LoRA's trainable parameters. Despite this substantial reduction, on the MetaMathQA dataset, the training time is shortened to 85.2% of LoRA's. Specifically, LoRA's training time is 25,142.27s, IntTune's training time is reduced to 21,439.26s. Additionally, IntTune enables GPU memory allocation to decrease as well. The percentage of GPU memory allocated drops from 80.9% to 72.5%, with the average memory usage reduced from 36.42 GB to 32.78 GB, a reduction of 9.98%. These results highlight the method's potential for improving performance while optimizing computational resources, making it particularly suitable for SFT LLMs in resource-constrained scenarios.

For NLU tasks, the number of trainable parameters was reduced to 3.07K, representing 0.002% of the total model parameters. Despite this significant reduction, the approach achieved 92.61% of LoRA's average performance across all tasks. Specifically, it attained 96.2% of LoRA's performance on SST-2, 94.5% on QNLI, and 96.2% on STS-B, demonstrating comparable performance across various GLUE tasks, underscoring its robustness and effectiveness in diverse scenarios.

These results highlight the effectiveness of Nyström initialization, as IntTune achieves strong performance, especially in NLG tasks where feature transformation is key. The relatively lower performance in NLU suggests that deeper semantic understanding may require more trainable parameters, as indicated by the rank-performance trend in Section 4.4. This suggests room for further adaptation in understanding-based tasks.

4.4 Experiments on Various Ranks

In this section, we examine the impact of progressively increasing the rank of NLoRA and SLoRA from 1 to 128 to assess their ability to consistently outperform the baseline across different ranks. Training is performed on the MetaMathQA dataset for a single epoch, with validation conducted on the GSM8K and MATH datasets.

The experimental results are presented in Figure 4. On the GSM8K dataset, NLoRA performs relatively better at higher ranks, surpassing LoRA by 43.08% and 36.41% at ranks 64 and 128, respectively. SLoRA, on the other hand, exhibits relatively stronger performance at lower ranks, outperforming LoRA by 107.45%, 77.31%, 53.54%, and 76.13% at ranks 1, 2, 4, and 8, respectively. On the MATH dataset, SLoRA shows a slight overall advantage, while NLoRA continues to deliver strong performance, particularly at higher ranks. Notably, the improvement of SLoRA is not solely attributable to parameter count. For example, as shown in Figure 4, SLoRA with rank 64 (161M parameters) outperforms LoRA with rank 128 (320M parameters) on GSM8K. This highlights that the

Strategy	Parameters	GSM8K	MATH	HumanEval	MBPP	MT-Bench
LoRA	320M	42.30	5.50	18.29	35.34	**4.58**
IntTune(Rank=256)	15M	**49.51**	6.62	**21.30**	33.90	3.59
IntTune(Rank=128)	4M	44.28	**6.86**	20.70	34.40	4.46
IntTune(Rank=64)	0.9M	37.98	5.56	14.60	**34.70**	4.55

Table 6: Compare the performance of different ranks for IntTune on NLG tasks

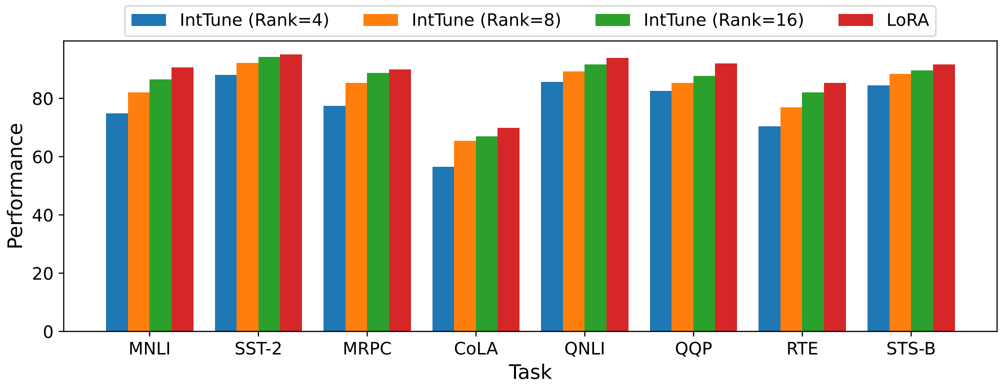


Figure 5: Compare the performance of different ranks for IntTune on NLU tasks

gain arises from SLoRA's structural enhancements, rather than merely relying on parameter increase.

The initialization time overhead for our methods and baselines, shown in Table 5. PiSSA initializes by directly decomposing the pre-trained weight matrix, resulting in an initialization time that is largely independent of the rank. The initialization time of other methods increases with higher ranks due to rank-dependent computations. Compared to PiSSA, our method achieves faster initialization, and although the overhead is slightly higher than that of LoRA, the gap remains within an acceptable range. This demonstrates that our method strikes a favorable balance between initialization efficiency and downstream performance.

For IntTune, we compared ranks of 64, 128, and 256 in the NLG tasks, following the same experimental setup as shown in Section 4.1. In the NLU experiments, we evaluated ranks of 4, 8, and 16. The results of these experiments are presented in Table 6 and Figure 5. On NLG tasks, IntTune does not exhibit a strictly increasing performance trend with higher ranks. Instead, different ranks excel in different tasks. Specifically, rank 128 and rank 256 achieve 7.45% and 5.62% higher performance than LoRA on average, both outperforming LoRA overall. Meanwhile, rank 64, though slightly lower, still reaches 93.66% of LoRA's performance, demonstrating the feasibility of fine-tuning with even fewer parameters while maintaining competitive results. On NLU tasks, the model performance gradually improves with increasing rank. For ranks 4, 8, and 16, the average performance reaches 86.20%, 92.61%, and 95.80% of LoRA's performance, respectively, while the number of parameters is only 1.35K, 3.07K, and 9.99K, respectively.

5 Conclusion

This work advances parameter-efficient fine-tuning strategies for large language models by introducing SLoRA and NLoRA, along with an exploration of an intermediate matrix fine-tuning method, IntTune. SLoRA incorporates a small intermediate matrix, enhancing expressiveness with minimal parameter overhead, while NLoRA leverages Nyström-based initialization to bypass the computational complexity of SVD, achieving competitive downstream performance. IntTune, by fine-tuning only the intermediate matrix in NLoRA, even boosts average NLG performance over LoRA while maintaining high parameter efficiency. Extensive experiments on NLG and NLU tasks demonstrate the robustness and adaptability of our methods, providing practical solutions for optimizing large models under resource constraints.

Limitations

While our method demonstrates strong performance in both NLG and NLU tasks, its applicability to ultra-low parameter fine-tuning approaches, such as IntTune, warrants further exploration. Additionally, extending our approach to visual tasks could provide valuable insights into its generalization and versatility across modalities. Furthermore, integrating SLoRA with advanced LoRA variants presents a compelling direction for future research to further enhance fine-tuning efficacy.

References

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*.

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

Christopher TH Baker and RL Taylor. 1979. The numerical treatment of integral equations. *Journal of Applied Mechanics*, 46(4):969.

Yupeng Chang, Yi Chang, and Yuan Wu. 2024. Balora: Bias-alleviating low-rank adaptation to mitigate catastrophic inheritance in large language models. *arXiv preprint arXiv:2408.04556*.

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*.

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. 2023. Qlora: efficient finetuning of quantized llms (2023). *arXiv preprint arXiv:2305.14314*, 52:3982–3992.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, and 1 others. 2023. Parameter-efficient fine-tuning of large-scale pretrained language models. *Nature Machine Intelligence*, 5(3):220–235.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*.

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

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*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. 2009. Ensemble nystrom method. *Advances in Neural Information Processing Systems*, 22.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and 1 others. 2024. Mixlora: Enhancing large language models finetuning with lora-based mixture of experts. *arXiv preprint arXiv:2404.15159*.

Mu Li, James Tin-Yau Kwok, and Baoliang Lü. 2010. Making large-scale nyström approximation possible. In *Proceedings of the 27th International Conference on Machine Learning, ICML 2010*, page 631.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2023. Relora: Highrank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

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

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364.

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*.

Arik Nemtsov, Amir Averbuch, and Alon Schclar. 2016. Matrix compression using the nyström method. *Intelligent Data Analysis*, 20(5):997–1019.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

David Persson, Nicolas Boullé, and Daniel Kressner. 2024. Randomized nystr\" om approximation of non-negative self-adjoint operators. *arXiv preprint arXiv:2404.00960*.

Si Si, Cho-Jui Hsieh, and Inderjit Dhillon. 2016. Computationally efficient nyström approximation using fast transforms. In *International conference on machine learning*, pages 2655–2663. PMLR.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Max Vladymyrov and Miguel Carreira-Perpinan. 2016. The variational nystrom method for large-scale spectral problems. In *International Conference on Machine Learning*, pages 211–220. PMLR.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Shaowen Wang, Linxi Yu, and Jian Li. 2024. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*.

Shusen Wang, Alex Gittens, and Michael W Mahoney. 2019. Scalable kernel k-means clustering with nystrom approximation: Relative-error bounds. *Journal of Machine Learning Research*, 20(12):1–49.

Shusen Wang and Zhihua Zhang. 2013. Improving cur matrix decomposition and the nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1):2729–2769.

Christopher Williams and Matthias Seeger. 2000. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13.

Tingyu Xia, Bowen Yu, Kai Dang, An Yang, Yuan Wu, Yuan Tian, Yi Chang, and Junyang Lin. 2024. Rethinking data selection at scale: Random selection is almost all you need. *arXiv preprint arXiv:2410.09335*.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148.

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*.

Zhiming Yang, Haining Gao, Dehong Gao, Luwei Yang, Libin Yang, Xiaoyan Cai, Wei Ning, and Guannan Zhang. 2024. Mlora: Multi-domain low-rank adaptive network for ctr prediction. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 287–297.

Zhengmao Ye, Dengchun Li, Zetao Hu, Tingfeng Lan, Jian Sha, Sicong Zhang, Lei Duan, Jie Zuo, Hui Lu, Yuanchun Zhou, and 1 others. 2023. mlora: Fine-tuning lora adapters via highly-efficient pipeline parallelism in multiple gpus. *arXiv preprint arXiv:2312.02515*.

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*.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.

A Detailed Derivation for Nyström Approximation

This section provides a detailed derivation of the Nyström approximation presented in Section 3, following the approach proposed in (Nemtsov et al., 2016). Specifically, the quadrature technique is applied to the sample matrix of W, followed by an out-of-sample extension to approximate W.

The basic quadrature technique of the Nyström method is used to approximate the Singular Value Decomposition (SVD) of a matrix. In this context, no eigen-decomposition is required. Specifically, denote the matrix $W \in \mathbb{R}^{m\times n}$ can be decomposed as:

$$W = \begin{bmatrix} A_W & B_W \\ F_W & C_W \end{bmatrix}. \tag{8}$$

where, $A_W \in \mathbb{R}^{r\times r}$ is designated to be the sample matrix, $B_W \in \mathbb{R}^{r\times(n-r)}$ and $F_W \in \mathbb{R}^{(m-r)\times r}$ represent the remaining sampled column and row components, respectively, and $C_W \in \mathbb{R}^{(m-r)\times(n-r)}$ corresponds to the remainder of the matrix W.

The derivation begins with the SVD of A_W, expressed as:

$$A_W = U\Lambda H^T, \tag{9}$$

where $U, H \in \mathbb{R}^{r\times r}$ are unitary matrices, and $\Lambda \in \mathbb{R}^{r\times r}$ is a diagonal matrix. Assuming that zero is not a singular value of A_W, the decomposition can be further approximated. Accordingly, the matrix U is formulated as:

$$U = A_W H\Lambda^{-1}. \tag{10}$$

Let $u^i, h^i \in \mathbb{R}^r$ represent the i-th columns of U and H, respectively. Denote $u^i = \{u^i_l\}_{l=1}^r$ as the individual elements of the i-th column of U. Using Eq. (10), each element u^i_l is expressed as the sum:

$$u^i_l = \frac{1}{\lambda_i}\sum_{j=1}^{n} W_{lj}\cdot h^i_j. \tag{11}$$

The elements of F_W can be used as interpolation weights to extend the singular vector u^i to the k^{th} row of W, where $s+1 \le k \le n$. Let $\tilde{u}^i = \{\tilde{u}^i_{k-s}\}_{k=s+1}^n \in \mathbb{R}^{n-s\times 1}$ denote a column vector comprising all the approximated entries. Each element $\tilde{u}^i_k$ is computed as:

$$\tilde{u}^i_k = \frac{1}{\lambda_i}\sum_{j=1}^{n} W_{kj}\cdot h^i_j. \tag{12}$$

Thus, the matrix form of $\tilde{u}^i$ is given by $\tilde{u}^i = \frac{1}{\lambda_i}F_W\cdot h^i$. By arranging all the $\tilde{u}^i$'s into a matrix $\tilde{U} = \begin{bmatrix}\tilde{u}^1 & \tilde{u}^2 & \ldots & \tilde{u}^r\end{bmatrix} \in \mathbb{R}^{n-s\times r}$, the following expression is obtained:

$$\tilde{U} = F_W H\Lambda^{-1}. \tag{13}$$

The Eq. (9) can also be written as $H = A_W^T U\Lambda^{-1}$. To approximate the right singular vectors of the out-of-sample columns, a symmetric argument is applied, yielding:

$$\tilde{H} = B_W^T U\Lambda^{-1}. \tag{14}$$

In that case, the full approximations of the left and right singular vectors of $\widehat{W}$, represented by $\tilde{U}$ and $\tilde{H}$, respectively, are then obtained as follows:

$$\widehat{U} = \begin{bmatrix} U \\ F_W H\Lambda^{-1} \end{bmatrix}, \quad \widehat{H} = \begin{bmatrix} H \\ B_W^T U\Lambda^{-1} \end{bmatrix}. \tag{15}$$

The explicit Nyström form of $\tilde{M}$ is given by:

$$\begin{aligned} \widehat{W} &= \widehat{U}\Lambda\widehat{H}^T \\ &= \begin{bmatrix} U \\ F_W H\Lambda^{-1} \end{bmatrix} \Lambda \begin{bmatrix} H^T & \Lambda^{-1}U^T B_W \end{bmatrix} \\ &= \begin{bmatrix} A_W & B_W \\ F_W & F_W A_W^+ B_W \end{bmatrix} \\ &= \begin{bmatrix} A_W \\ F_W \end{bmatrix} A_W^+ \begin{bmatrix} A_W & B_W \end{bmatrix}, \end{aligned} \tag{16}$$

where A_W^+ denotes the pseudo-inverse of W. In this approximation, $\widehat{W}$ does not modify A_W, B_W and F_W but approximates C_W by $F_W A_W^+ B_W$. This approach achieves a matrix approximation using only the selected rows and columns, effectively capturing the essential structure with reduced computational complexity.

B Experiments on Various Initializations

For SLoRA, we kept the initialisation of the A and B matrices the same as for LoRA, and in turn explored the effect of different methods of initialisation of the intermediate matrices on the results. Specifically, we experimented with Kaiming initialization and Gaussian initialization on all the NLG tasks of LLaMA 2-7B, with the same experimental setup as in Section 4. The performance of the models under these settings is shown in Table 7. The results indicate that Kaiming initialization consistently achieves better performance across all tasks. Gaussian initialization also achieves competitive results, which demonstrates the robustness of our method. In our experiments, we use kaiming to initialize SLoRA.

Tasks	Kaiming	Gaussian
GSM8K	**56.48**	56.10
MATH	**10.68**	9.56
HumanEval	**23.78**	23.2
MBPP	**42.32**	40.5
MT-Bench	**4.85**	3.93

Table 7: Different Initialization on SLoRA

Strategy	LR	GSM8K	MATH
SLoRA	2E-04	56.48	10.68
	5E-04	**59.51**	**11.04**
	2E-05	51.02	6.94
	5E-05	52.84	8.36
NLoRA	2E-04	**57.70**	9.94
	5E-04	54.81	**10.60**
	2E-05	45.11	6.42
	5E-05	52.39	7.58

Table 8: Comparasion of different learning rate on SLoRA and NLoRA

LR	GSM8K	MATH
2E-04	43.29	5.74
5E-04	44.20	5.70
2E-03	**44.28**	**6.86**
5E-03	40.86	6.08

Table 9: Comparasion of Different Learning Rates on IntTune

C Experiments on Various Learning Rates

We evaluated the impact of four learning rates: 2E-4, 2E-5, 5E-4 and 5E-5 on the model's performance. The experimental setup remains the same as described earlier. The results of these experiments are presented in Table 8. Among the evaluated learning rates, 5E-4 achieved the best overall performance. However, we opted for 2E-4 in our experiments, as its performance, while slightly lower than that of 5E-4, remained comparable and still exceeded the original baseline. Moreover, at the learning rate of 2E-4, NLoRA exhibited lower loss and better convergence behavior, making it a more appropriate choice for our experimental setup.

For the case of fine-tuning only the intermediate matrix, we tested the performance under different learning rates. The results indicate that a learning rate of 2E-3 achieved the best performance. The result is shown in Figure 9.

D Experiments on Various Optimizers

We experimented with different optimizers on both NLG and NLU tasks. In addition to the default AdamW optimizer, we also evaluated the RMSProp optimizer. Other experimental setups are the same as Section 4. The experimental results are shown in Table 10 and Table 11.

On NLG tasks, we observed that the RMSProp optimizer further improved the model's performance. However, its performance on NLU tasks was relatively mediocre. This discrepancy might stem from the underlying differences in the nature of NLG and NLU tasks. NLG tasks typically involve generating coherent sequences of text, which require more stable gradient updates over longer contexts. RMSProp's adaptive learning rate mechanism, which emphasizes recent gradients, may help maintain stability and enhance performance in such scenarios. In contrast, NLU tasks often involve classification or regression over shorter input sequences, where AdamW's weight decay and bias correction might be more effective in avoiding overfitting and ensuring generalization, thus outperforming RMSProp in these tasks.

E Experimental Settings on NLU

We evaluate the performance on the GLUE benchmark, which includes two single-sentence tasks (CoLA and SST-2), three natural language inference tasks (MNLI, QNLI, and RTE), and three similarity and paraphrase tasks (MRPC, QQP, and STS-B). For evaluation metrics, we report overall accuracy (matched and mismatched) for MNLI, Matthew's correlation for CoLA, Pearson's correlation for STS-B, and accuracy for the remaining datasets.

In DeBERTa-v3-base, SLoRA and NLoRA were applied to the W_Q, W_K, and W_V matrices, while in RoBERTa-large, they were applied to the W_Q and W_V matrices. The experiments for natural language understanding (NLU) were conducted using the publicly available LoRA codebase. For MRPC, RTE, and STS-B tasks, we initialized RoBERTa-large with a pretrained MNLI checkpoint. The rank of SLoRA and NLoRA in these experiments was set to 8. Optimization was performed using AdamW with a cosine learning rate schedule. Table 12 and Table 13 outline the hyperparameters used for the GLUE benchmark experiments.

For IntTune, we set both the LoRA rank and LoRA alpha to 8. The remaining parameter configurations are provided in Table 14.

Strategy	Parameters	GSM8K	MATH	HumanEval	MBPP	MT-Bench
LoRA	320M	42.30	5.50	18.29	35.34	4.58
NLoRA	323M	57.70	9.94	25.00	43.12	4.82
NLoRA+RMSProp	323M	**58.10**	**10.82**	**25.60**	**43.40**	**4.99**

Table 10: Comparision of Adamw and RMSProp on NLG

Strategy	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
LoRA	90.65	94.95	89.95	69.82	93.87	91.99	85.20	91.60
NLoRA	**90.74**	**96.22**	**91.91**	**73.41**	**94.45**	**92.03**	**88.09**	**92.14**
NLoRA+RMSProp	90.41	**96.22**	**91.91**	68.61	94.18	**92.03**	**88.09**	91.86

Table 11: Comparision of Adamw and RMSProp on NLU

Dataset	DeBERTa-v3-base				RoBERTa-large			
	LR	BS	Epoch	LoRA alpha	LR	BS	Epoch	LoRA alpha
CoLA	3E-04	16	40	16	4E-04	8	20	8
SST-2	5E-04	16	10	8	5E-04	16	10	8
MRPC	5E-04	32	100	16	2E-04	32	50	16
MNLI	3E-04	32	10	16	3E-04	32	10	16
QNLI	2E-04	32	20	16	6E-04	16	10	8
QQP	6E-04	32	20	8	6E-04	16	10	16
RTE	3E-04	32	40	16	5E-04	32	30	16
STS-B	5E-04	16	10	16	3E-04	16	30	16

Table 12: Hyperparameters of NLoRA on GLUE

Dataset	DeBERTa-v3-base				RoBERTa-large			
	LR	BS	Epoch	LoRA alpha	LR	BS	Epoch	LoRA alpha
CoLA	3E-04	16	40	16	4E-04	8	20	8
SST-2	5E-04	16	10	8	5E-04	16	10	8
MRPC	5E-04	32	100	16	2E-04	32	50	16
MNLI	3E-04	32	10	16	3E-04	32	20	16
QNLI	2E-04	32	20	16	6E-04	16	10	8
QQP	6E-04	32	20	8	6E-04	16	10	16
RTE	3E-04	32	40	16	5E-04	32	30	16
STS-B	5E-04	16	10	16	3E-04	16	30	16

Table 13: Hyperparameters of SLoRA on GLUE

Dataset	LR	BS	Epoch
CoLA	7E-03	16	40
SST-2	6E-03	32	30
MRPC	4E-03	16	50
MNLI	6E-03	64	20
QNLI	8E-03	64	20
QQP	6E-03	32	20
RTE	6E-03	16	25
STS-B	6E-03	16	60

Table 14: Hyperparameters for IntTune