Relayed LoRA: Extreme Parameter Efficiency via Quad-Matrix Low-Rank Adaptation

Anonymous ACL submission

Abstract

Fine-tuning Large Language Models (LLMs) via Parameter-Efficient Fine-Tuning (PEFT) has become the standard for adapting general-purpose models to specialized downstream tasks. Among these, Low-Rank Adaptation (LoRA) is widely adopted for its ability to reduce trainable parameters by utilizing a low-rank decomposition of weight updates. However, as the demand for deploying thousands of task-specific adapters on resource-constrained edge devices grows, the storage and memory overhead of standard LoRA remains a critical bottleneck. In this study, we propose Relayed LoRA, a novel algorithm designed to push the boundaries of parameter efficiency. Our approach introduces a secondary, "relayed" decomposition of the two standard LoRA matrices (A and B) into a quad-matrix structure (A_1, A_2, B_1, B_2). By leveraging a fixed structural mapping, we decouple the representational rank from the total number of trainable parameters, enabling the system to maintain a high-rank update while significantly reducing the parameter footprint—often by an order of magnitude. Empirical evaluations on the GLUE benchmark and a clinical dementia language dataset demonstrate that Relayed LoRA achieves performance comparable to standard LoRA while utilizing substantially fewer parameters. Our results suggest that Relayed LoRA provides a scalable and efficient framework for large-scale multi-task deployment, offering a new paradigm for extreme parameter-efficient fine-tuning in memory-sensitive environments.

1 Introduction

While LLMs are powerful for general purpose applications, they often need fine-tuning to suit to specific purpose tasks. In this process, PEFT offers a computationally efficient way of re-training an LLM (Hu et al., 2023; Han et al., 2024; Runwal et al., 2025). Among the PEFT techniques, LoRA is widely used for it offering large savings in the number of trainable parameters and good performance (Hu et al., 2022; Lin et al., 2024). Experiments show that in some cases, LoRA can even achieve better performance than tuning all the parameters of a model (Mao et al., 2025). The basic setup of LoRA is to approximate a large matrix by the multiplication of two small matrices with a rank lower than that of the large matrix. This results in fewer parameters to train and leads to a low computational cost. The phenomenon of a low rank update in transfer learning has been noticed by researchers for some time. It has been shown that when an LLM is trained long enough, changes to its parameters somehow become a low rank update (Aghajanyan et al., 2021). Later on, the effectiveness of LoRA in terms of its expressive power has been proved by Zeng and Lee (2023), providing mathematical evidence for its success.

Although LoRA dramatically reduces the number of parameters needed for fine-tuning an LLM, sometimes that number may still be too large, especially for edge devices. Researchers have proposed modifications to LoRA from various aspects (Gao et al., 2024; Liu et al., 2024; Tian et al., 2024; Babakniya et al., 2023; Qi et al., 2024). One cost or side effect of LoRA is that it introduces a longer computational path. To alleviate the problems associated with this long computational path, Shi et al. introduced a residual path in training LoRA (Shi et al., 2024). Their results indicate that the residual path leads to more efficient training. LoRA has also been integrated with quantization to further reduce the footprint of an LLM in implementation (Qin et al., 2024; Xu et al., 2023; Li et al., 2023).

In this work, we proposed a Relayed LoRA approach to further reduce the number of trainable parameters. In the standard LoRA, a large to-be-tuned parameter matrix ΔW is approximated by the multiplication of two smaller matrices, denoted as A and B, which are of lower rank than the large

matrix. With an approximate choice of the ranks of A and B, their multiplication can approximate ΔW. In the Relayed LoRA, we propose to further express A and B as the multiplication of four smaller matrices, denoted as A_1, A_2, B_1, and B_2, so the four small matrices together use even fewer parameters than A and B would use. In our framework, only the four small matrices are trained, and they are used to create matrices A and B. Then A and B are multiplied to create ΔW. In other words, A and B are no longer trainable in Relayed LoRA, instead, they just serve the purpose of passing the parameters in A_1, A_2, B_1, and B_2 to ΔW.

In this work, we proposed a new LoRA method that further reduces the number of parameters required in the standard LoRA. The main contribution of our work is that we proposed a relayed low rank adaptation method. The proposed Relayed LoRA method further reduces the number of trainable parameters required in the standard LoRA. In Relayed LoRA, the two low rank matrices used in the standard LoRA are decomposed into four smaller matrices to save on parameters. In training an LLM, only the four small matrices are updated. Their multiplication results are passed to the two large matrices in standard LoRA. The two large matrices, however, do not get trained. They only function as relay stations to pass the updates from the four small matrices to the trainable layers of an LLM. Hence, the new approach is termed Relayed LoRA. Because now only the four small matrices contain trainable parameters, the relay process substantially reduces the number of parameters. Experiments in this work show that Relayed LoRA can save up to 80% of the parameter compared to standard LoRA. Theoretically, more savings are possible when the number of parameters in the standard LoRA is even larger.

The novelty of our work lies in the significant further reduction of trainable parameters achieved by Relayed LoRA. Built upon standard LoRA, which directly applies low-rank adaptation through two trainable matrices A and B, Relayed LoRA takes a step further by decomposing each of these matrices into two smaller components, effectively creating a cascaded low-rank adaptation pathway, allowing greater greater parameter efficiency.

2 Related Work

Since the introduction of LoRA, many works have been presented to further enhance the training process or apply it to different types of model configuration, such as federated learning (Cho et al., 2024; Wu et al., 2024b; Yang et al., 2024). In improving the training process, Valipour et al. (2022) proposed a DyLoRA method that was shown to speed up the training by four to seven times than the standard LoRA. Other modifications include ReLoRA by Lialin et al. (2023) in which the LoRA process was repeated several times in implementation. The essence of LoRA is to use low-rank matrices to approximate a large matrix. LoRA can be implemented for different layers of a large model and as such, the ranks of the small matrices can be set unequal, as proposed in AdaLoRA by Zhang et al. (2023). LoRA has also been combined with other techniques in implementation. For example, Dettmers et al. (2023) proposed QLoRA to integrate LoRA with quantization in facilitate the training of LLMs on devices with limited memory. Xia et al. (2024) designed multi-task LLM quantization and using multiple LoRA adaptors to achieve memory-efficient serving. Gao et al. (2024) presented a FashingGPT model which involves fine-tuning multiple independent LoRA-adapters to create a versatile LLM. In the standard LoRA, the two matrices A and B are updated with the same learning rate. Researchers have shown that this equal learning rate may lead to sub-optimal performance for LLMs when its embedding dimension is large and proposed a LoRA+ method, which updates A and B with different paces (Hayou et al., 2024). Though LoRA effectively reduces the number of trainable parameters, it may suffer from low serving efficiency. Wu et al. (2024a) proposed a dLoRA, which stands for dynamic LoRA, method that obtains high serving efficiency by dynamically orchestrating requests and LoRA adapters. Beyond saving trainable parameters, LoRA has been shown to exhibit a good capability to accommodate additional requirements by users. For example, there were reports that fine-tuning of an LLM may increase the risk that the model will generate malicious output (Huang et al., 2024a,b). Yet, it was shown that by patching some simple codes in LoRA, such risk can be reduced (Hsu et al., 2024).

3 Method

3.1 Brief Review of Standard LoRA

LoRA leverages low-rank matrix decomposition to update the pre-trained weights, with the primary goal of substantially reducing the number of train-

able parameters. In LoRA, the updated parameter matrix is regarded as the original pre-trained parameter matrix, which is frozen, plus an incremental matrix ΔW, that is

$$W = W_0 + \Delta W \tag{1}$$

where W represents the updated matrix, W_0 represents the original pre-trained parameter matrix, and ΔW represents the parameter incremental matrix. LoRA operates on the assumption that the update matrix possesses a low "intrinsic rank" r, where $r \ll d$, and d is the dimension of W. Here, for the purpose of discussion, we assume that W is a square matrix of size $d \times d$, but note that the proposed method and the discussion are equally applicable to W if it is not a square matrix. LoRA decomposes the parameter increment matrix ΔW into two low-rank matrices A and B to approximate it while significantly reducing the amount of trainable parameters. When LoRA is applied to a large language model based on the transformer architecture, the parameter matrix of the original pre-trained model is frozen so it does not participate in the parameter update. At the same time, matrices A and B of LoRA are trained to perform low-rank adaptation. In this way, LoRA divides the propagation path into two paths, one is the result of the pre-trained parameters, that is, the output result of the original model; the other is the result of the low-rank matrix adaptation. Then the two results are added together to obtain the final training result,

$$Wx = W_0x + (AB)x. \tag{2}$$

While in theory, the LoRA approach can be applied to any layers or modules of a large model, according to Hu et al. (2022), for an LLM using Transformer, applying the LoRA method to the query and value matrices of the self-attention module achieves the best results. According to the experiments in (Hu et al., 2022), the LoRA fine-tuning technology can improve the effect of the model by updating a small number of model parameters for different downstream tasks while maintaining the effect of the original model. Experimental results show that in the process of model fine-tuning, selecting a very low rank can well fit the original parameter matrix and update the parameters for a small number of features.

3.2 Our Method – Relayed LoRA

In Relayed LoRA, we propose that the A and B matrix in the standard LoRA method can be further decomposed. In the proposed Relayed LoRA method, matrices A and B are further decomposed as $A = A_1 \times A_2$ and $B = B_1 \times B_2$ of which A_1, A_2, B_1, and B_2 are of smaller sizes than A and B. For an original parameter matrix W of size (R_W, C_W), we assume that matrix A takes the size of (R_W, r) where r is generally a much smaller number than R_W and C_W. Similarly, we assume that matrix B has the size of (r, C_W). The number of elements in A is $R_W \times r$ and in B is $r \times C_W$. Take A for example, we compute the smallest integer d_A that is larger than the squared root of $R_W \times r$ such that

$$d_A = \lceil \sqrt{R_W \times r} \rceil. \tag{3}$$

The smaller matrices A_1 and A_2 are then defined with dimensions (d_A, k) and (k, d_A), respectively, where k is a user-specified hyperparameter. It can be seen that $A_1 \times A_2$ will give a square matrix A_{square} such that

$$A_{square} = A_1 \times A_2 \tag{4}$$

whose size is (d_A, d_A). We then populate A by extracting the first $R_W \times r$ elements of A_{square}. Since A_{square} typically contains more elements than A, the surplus elements are disregarded and do not participate in subsequent computations. In this step, it is important that we always assign the same elements from A_{square} to fill into A, while we are free to choose whether we assign the first $R_W \times r$ elements or last $R_W \times r$ elements of A_{square} or by some other manners to A, as long as we are consistent in selecting the elements to fill into A.

Similarly, we set

$$d_B = \lceil \sqrt{r \times C_W} \rceil. \tag{5}$$

and compute two small matrices B_1 and B_2 with sizes of (d_B, k) and (k, d_B), respectively. And we can obtain a square matrix B_{square} such that

$$B_{square} = B_1 \times B_2 \tag{6}$$

whose size is (d_B, d_B). We then extract the first $r \times C_W$ elements from B_{square} to fill into B and leave extra elements in B_{square}, if any, unused. Figure 1 shows the process of Relayed LoRA. For the purpose of description, we assume that the big matrix ΔW is of size (R_W, C_W). We also assume that the rank of matrices A and B is r. So we have A of size (R_W, r) and B of size (r, C_W). Up to this point, the procedure is

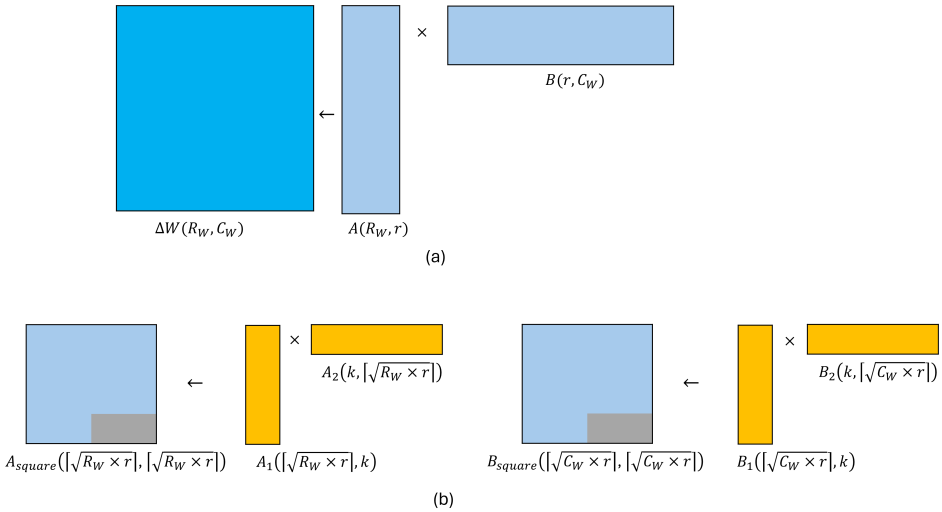


Figure 1: In the Relayed LoRA setup, (a) shows how a large parameter matrix ΔW is conceptually obtained by the multiplication of matrices A and B. (b) uses A for illustration to show that it is computed by the multiplication of two smaller matrices A_1 and A_2 to create matrix A_{square} and then the correct number of elements in A_{square} are selected to fill into A in (a). As the elements in A may not happen to be a square number, some elements are left unused (grayed area). The procedure for calculating B is the same.

the same as in the standard LoRA, Figure 1(a). Here note that in Relayed LoRA, A and B are not trained. Instead, we further decompose them by the multiplication of even smaller matrices. The procedure for decomposing A is shown in Figure 1(b). We first build a blank A_{square} which is the smallest square matrix that can contain the number of parameters of A, that is A_{sqaure} is of size $(\lceil\sqrt{R_W \times r}\rceil, \lceil\sqrt{R_W \times r}\rceil)$. We then set and train two smaller matrices A_1 of size $(\lceil\sqrt{R_W \times r}\rceil, k)$ and A_2 of size $(k, \lceil\sqrt{R_W \times r}\rceil)$ to obtain A_{square}. At this step, the A_{sqaure} we obtain will generally have more parameters than A has. So we select the first $R_W \times r$ parameters of A_{square} to fill into A and leave the remaining parameters unused, as indicated by the gray boxes in the figure. In this procedure, matrix A serves as a relay station and is not get trained. Hence we call A a relay matrix. The procedure for decomposing B into B_1 and B_2 is not shown as it is similar. The number of trainable parameters is now the sum of parameters in A_1, A_2, B_1, and B_2. This is a further reduction from the trainable parameters required in the standard LoRA.

Now, given A and B obtained from the above steps, we then follow the steps of the standard LoRA to build matrix ΔW. In Relayed LoRA, only the four small matrices A_1, A_2, B_1, and B_2 are trained, while A_{square}, B_{square} and A, B are just placeholders and do not get trained. In other words, A_{square}, B_{square}, A, and B serve as relay stations by passing the trainable parameters of A_1, A_2, B_1, and B_2. In the procedure of relay, matrix mapping is carried out from A_{square} to A and from B_{square} to B. The mapping is from a square matrix to a rectangle matrix, as illustrated in Figure 2, in which we assume that A_{sqaure} is of size 4×4 and A of 7×2. The mapping from A_{sqaure} to A is represented by the matched color codes in the figure. Since A_{square} has more elements than A, the last several elements are simply left unused. The mapping from B_{square} to B is similar. Here we note that the mapping can take other patterns, as long as the pattern is consistent during the training process of Relayed LoRA, it will not affect the performance of the method. After obtaining A and B, the rest steps is the same as in standard LoRA.

In Relayed LoRA, each matrix is calculated as follows.

$$W_0 + \Delta W = W_0 + AB \tag{7}$$

in which

$$A = A_1 \times A_2 \tag{8}$$
$$B = B_1 \times B_2. \tag{9}$$

The number of trainable parameters depends on R_W, C_W, r, and k. The total number of trainable parameters in Relayed LoRA is

$$\begin{aligned} N_{new} &= 2d_A \times k + 2d_B \times k \\ &= 2\lceil\sqrt{R_W \times r}\rceil k + 2\lceil\sqrt{r \times C_W}\rceil k \end{aligned} \tag{10}$$

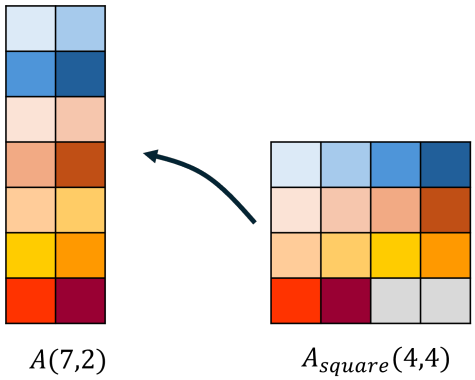


Figure 2: Illustration of the mapping from matrix A_{square} to A. We use matched color codes to illustrate the mapping from locations in A_{square} to A. The two extra elements in A_{square} are left unused (gray).

while in the standard LoRA, the total number $N_{standard}$ of trainable parameter is

$$N_{standard} = R_W \times r + C_W \times r. \quad (11)$$

It can be seen that N_{new} is smaller than $N_{standard}$ as long as r and k are considerably smaller than R_W and C_W, which is often the case in reality. The dimensions of each matrix are summarized in Table 1. In the initialization of the standard LoRA, one of matrices A and B, e.g., B is set to zero. In the initialization of Relayed LoRA, we similarly set B_1 to zero so that B_{square} and hence B will be zero at first iteration.

Matrix	Dimensions
Original ΔW	(R_W, C_W)
Relay matrix A	(R_W, r)
Relay matrix B	(r, C_W)
A_{square}	$(\lceil\sqrt{R_W \times r}\rceil, \lceil\sqrt{R_W \times r}\rceil)$
B_{square}	$(\lceil\sqrt{C_W \times r}\rceil, \lceil\sqrt{C_W \times r}\rceil)$
Small A_1	$(\lceil\sqrt{R_W \times r}\rceil, k)$
Small A_2	$(k, \lceil\sqrt{R_W \times r}\rceil)$
Small B_1	$(\lceil\sqrt{C_W \times r}\rceil, k)$
Small B_2	$(k, \lceil\sqrt{C_W \times r}\rceil)$

Table 1: Dimensions of matrices in Relayed LoRA.

3.3 Theoretical insight into parameter savings in Relayed LoRA

To ensure the resulting product aligns with the required dimensions of the base model ($d \times r$), we employ a Fixed Structural Mapping (Φ) scheme. Rather than using random projections, Φ acts as a deterministic indexing function that reshapes the quad-matrix product. By keeping Φ static during training, we preserve the differentiability of the system, allowing gradients to flow seamlessly from the objective function through the intermediate product to the trainable quad-matrices.

In this work, we choose to decompose each matrix A and B into two small square matrices, A_1 and A_2, and B_1 and B_2. The reason of decomposing A and B into the multiplication of two square matrices is to train as few trainable parameters as possible. It can be proved that when a large matrix is decomposed into two square matrices, the total number of elements in the two square matrices is smaller than any other kinds of decomposition. In other words, such a decomposition is the most efficient one. In this work, we decompose A and B separately, thus we have four small trainable matrices. We can also consider A and B together as one big matrix such that $C = A \oplus B^T$ where $\oplus$ stands for concatenation and superscript T is transpose.

Relayed LoRA is designed to yield further savings in parameters for LoRA-kind approach. Comparing Eq. (10) to (11), it can be seen that the saving is proportionally larger for bigger R_W and C_W. Mathematically, this can be proved by comparing the corresponding terms of Eq. (10) with those of Eq. (11). For example, for the terms involving R_W, it is $2\lceil\sqrt{R_W \times r}\rceil k$ in Eq. (10) and $R_W \times r$ in Eq. (11). The savings in percentage of the number of parameters is

$$\epsilon = 1 - \frac{2\lceil\sqrt{R_W \times r}\rceil k}{R_W \times r} \approx 1 - \frac{2k}{\sqrt{R_W \times r}} \,(\%). \quad (12)$$

As k is usually a much smaller number than R_W, it can be seen that the saving is more significant for large R_W and r. The same proof can be derived for the terms involving C_W. Here we note that in theory, k can be set to one to give the highest savings in parameters. Nevertheless, it can be set to greater than one to give the small matrices more expression power.

4 Experiments

4.1 Implementation

All the experiments were conducted on a workstation equipped with an NVIDIA RTX4090D GPU with 24 GB of memory and CUDA 2.1. The codes were implemented in Python 3.0. We used AdamW for optimizer. We did not use warm-up ratio and learning rate schedule.

In this work, we used RoBERTa base, DeBERTa base, and RoBERTa large models as the backbone

LLMs (Liu et al., 2019). Experiments were set up in the same configuration of Hu et al. (2022). When applying LoRA fine-tuning, we only focused on the query and value layers of the self-attention module of the Transformer and froze the parameters of all layers except the classification head to ensure that only the LoRA part is involved in training.

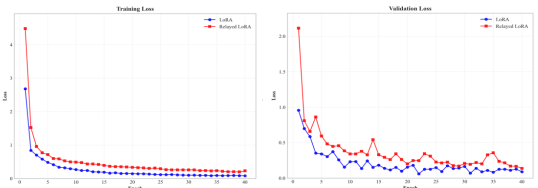
4.2 GLUE data and results

We first evaluate Relayed LoRA on the GLUE benchmark, a collection of datasets developed by multiple institutions to assess natural language understanding capabilities (Wang et al., 2018). GLUE contains several NLU tasks performed in English, including sentiment analysis, natural language inference, and semantic similarity. Due to computational constraints, we selected a subset of GLUE tasks with smaller dataset sizes for our experiments. Information about the data is given in Table 2. All the tasks are binary classification, except for STS-B, whose task is regression over seven degrees of semantic similarity, i.e., a seven-category classification task. In training the LLMs, learning rate was set to 4e-4 for CoLA, MRPC, and STS-B data and 5e-4 for RTE and SST-2 data. For the first experiment, we set both r and k to eight and evaluated Relayed LoRA on the GLUE datasets. The test results are shown in Table 3. As shown in our results, setting $r = 8$ and $k = 8$ leads to a substantial reduction in the number of trainable parameters for each LLM. For the RoBERTa base model, the number of parameters was reduced from 294,912 for standard LoRA to 60,672 for Relayed LoRA, representing a 79.5% reduction. The performance of Relayed LoRA was very close to that of the standard LoRA across all the datasets. For most datasets such as QNLI, SST-2, QQP and STS-B, the performance of Relayed LoRA was within one percentage point to that of the standard LoRA. As shown in Table 3, a divergence is observed in the RTE dataset (69.2% vs. 83.1% in standard LoRA). We hypothesize that this is not a limitation of the quad-matrix architecture, but rather a sensitivity to the gradient sparsity inherent in extreme compression. In low-resource settings like RTE, the restricted optimization landscape of Relayed LoRA requires more sophisticated learning rate scheduling. Future work will explore whether dynamic rank allocation can mitigate this bottleneck.

For the DeBERTa base model, the number of parameters for the standard LoRA was 442,368 and 91,008 for Relayed LoRA. The savings in parameters were 79.4%, similar to that of RoBERTa base model. The performance of Relayed LoRA was also very close to that of the standard LoRA. For example, the accuracy of the standard LoRA on QNLI was 50.5%, while Relayed LoRA had almost the same accuracy of 50.1%.

For the RoBERTa large model, the savings in parameters was 82.2%, 139,776 trainable parameters for Relayed LoRA versus 786,432 trainable parameters for standard LoRA. For the RoBERTa large model, there are cases that the performance of Relayed LoRA was a few percentage lower than that of standard LoRA. This was more obvious for the RTE dataset, for which the standard LoRA had an accuracy of 83.1% while Relayed LoRA's accuracy as 69.2%. Similar observation can be made about the STS-B dataset. This performance gap may be attributed to the relatively small size of the RTE and STS-B datasets, which might have provided insufficient training data for Relayed LoRA. So from the above experiments, we observed that Relayed LoRA offered a large saving in trainable parameters while attaining approximately the same performance as the standard LoRA for most datasets and LLMs we tested.

To assess how Relayed LoRA behaves during training, we plotted the training and validation losses in Figure 3. It is seen that in this case, Relayed LoRA had similar training and validation losses as the standard LoRA.



(a) (b)

Figure 3: (a), training loss and (b), validation loss of Relayed LoRA and standard LoRA, with RoBERTa base as the backbone model.

Next, to assess how Relayed LoRA performs for different k, we used MRPC as a test bed. We chose RoBERTa base as the backbone model and set r to 64 and let k change from 4 to 32. For comparison, the standard LoRA had 6,291,456 trainable parameters and obtained an F1 score of 90.5 and accuracy of 87.1%. The results of Relayed LoRA are plotted in Figure 4. In the plot, the savings in parameters were calculated as the number of parameters in Relayed LoRA over that in the standard LoRA. From

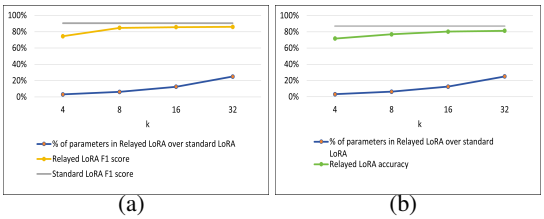
dataset	Task	Evaluation metrics	Data size
QNLI	Question natural language inference	Accuracy	100,000
SST-2	Sentiment analysis	Accuracy	67,000
QQP	Quora question pairs	F1 score, accuracy	400,000
MRPC	Synonym detection	F1 score, accuracy	3,700
MNLI	Multi-genre natural language inference	Accuracy	433,000
RTE	Text entailment	Accuracy	2,000
CoLA	Grammatical correctness	Matthew's correlation coefficient	900
STS-B	Semantic similarity	Pearson correlation coefficient	5,700

Table 2: GLUE datasets used in this work.

Model	PEFT	Trainable parameters	QNLI	SST-2	QQP	MRPC	MNLI-m	MNLI-mm	RTE	CoLA	STS-B*
RoBERTa base	LoRA	294,912	51.2%	49.8%	18.2%/ 70.1%	70.3%/ 58.6%	85.7%	85.5%	48.9%	54.2%	88.1%/ 87.7%
	Relayed LoRA	60,672	50.1%	49.3%	17.7%/ 70.8%	67.3%/ 55.7%	85.9%	85.4%	50.8%	50.8%	87.1%/ 86.8%
DeBERTa base	LoRA	442,368	50.5%	49.6%	19.3%/ 68.9%	67.4%/ 55.7%	89.5%	89.5%	54.1%	67.1%	90.4%/ 89.9%
	Relayed LoRA	91,008	50.1%	49.8%	19.6%/ 68.1%	70.1%/ 58.4%	88.6%	88.7%	48.7%	65.1%	86.9%/ 85.4%
RoBERTa large	LoRA	786,432	50.4%	50.4%	21.0%/ 65.8%	90.3%/ 86.8%	87.1%	86.5%	83.1%	60.3%	89.9%/ 89.6%
	Relayed LoRA	139,776	50.7%	48.2%	14.3%/ 74.7%	86.1%/ 80.3%	78.8%	79.9%	69.2%	59.8%	70.3%/ 70.6%

Table 3: Test results on using $r = 8$ and $k = 8$. Performance measured in single metrics was accuracy. Performance measured in two metrics were F1 score and accuracy. *STS-B's performance was measured in Pearson correlation coefficient and Spearman correlation coefficient.

the plots it is seen that as k became large, the performance of Relayed LoRA increased, though it appeared to plateau when k reached 32 for $r = 64$. As expected, for smaller k, the savings in trainable parameters is more prominent. However, even for large k such as 32, the savings in parameters are still significant as Relayed LoRA used only about 25% of the parameters as the standard LoRA.



(a) (b)

Figure 4: (a), F1 score and (b), accuracy in analyzing MRPC dataset by Relayed LoRA. Gray lines are the results of standard LoRA.

4.3 Dementia data and results

For the second experiments, we used three dementia transcript datasets, all of which are from the dialogue task of picture description in English and Chinese. They are ADReSS, Pitt, and iFLY datasets. The ADReSS dataset contains 78 dementia patients and 78 normal controls from the 2020 ADReSS Challenge. The Pitt dataset is the Pittsburgh corpus taken from the Dementiabank dataset, which comes from an annual longitudinal study conducted by the University of Pittsburgh School of Medicine. There are 498 patient data, including 242 controls and 256 patients with possible or very likely Alzheimer's disease (AD), and the two groups are basically balanced. The iFLY Chinese dataset contains 111 controls and 68 Alzheimer's patients, including 60 women and 51 men in the control group, and 38 women and 30 men in the AD group. The data is available at http://challenge.xfyun.cn/2019/gamedetail?blockId=978.

All datasets contain two columns, with one column being the transcripted text of patient conversation and the other being the label. The conversation text length is uneven, with the longest text length being 2588 and the shortest text length being 79. In the dataset, the ADReSS and Pitt datasets are English texts with two labels: 0 means that the patient does not have Alzheimer's disease; 1 means that the patient may have mild cognitive impairment (or Alzheimer's disease). The iFLY dataset is Chinese text with three labels: 0 means that the patient does not have Alzheimer's disease; 1 means that the patient may have mild cognitive impairment

(or Alzheimer's disease); 2 means that the patient has Alzheimer's disease. In addition, we merged the ADReSS and Pitt datasets as a whole English dataset to expand the data volume. We call this dataset the combined set. Information about the data is shown in Table 4. We first used RoBERTa

Label	ADReSS	Pitt	iFLY	Combined
0	78	242	68	320
1	78	256	111	334
2	-	-	144	-
Total	156	498	323	654

Table 4: Distribution of different stages of dementia in each dataset. The combined dataset is the combination of ADReSS and Pitt data.

base as the backbone model and trained standard and Relayed LoRA to analyze the transcripts of AD patients and health controls. Results are shown in Table 5. Relayed LoRA used about 20% of the trainable parameters as standard LoRA, yet the two methods had almost the same accuracy in classifying patients' transcripts.

PEFT	Trainable parameters	ADRess	Pitt	iFLY	Combined
LoRA	294,912	75%	82%	49%	90.8%
Relayed LoRA	60,672	78%	81%	51%	91.6%

Table 5: Number of trainable parameters and accuracy on test set by using RoBERTa base as the backbone model to classify subjects' transcripts into healthy, mild cognitive impairment, and AD.

4.4 Depression data and results

As another test, we applied Relayed LoRA to a depression transcript dataset, whose purpose is to identify whether a speaker is potentially a depression patient. The data was downloaded from http://www.kaggle.com/datasets/kyharndeok/dpression. Results are shown in Table 6, from which we note that Relayed LoRA achieved similar accuracy as conventional LoRA but used far fewer parameters, pointing to the parameter-efficient advantage of Relayed LoRa.

Model	PEFT	Trainable parameters	Accuracy
RoBERTa base	LoRA	294,912	80.46%
	Relayed LoRA	60,672	79.06%
DeBERTa base	LoRA	442,368	79.06%
	Relayed LoRA	91,008	77.24%
RoBERTa large	LoRA	786,432	79.24%
	Relayed LoRA	139,776	77.47%

Table 6: Test results on the depression dataset.

5 Discussion and Conclusions

The primary innovation of this research is the introduction of a hierarchical decomposition strategy for adapter-based tuning. To the best of our knowledge, this is the first work to successfully implement a quad-matrix relay to achieve extreme parameter efficiency without sacrificing the underlying rank of the weight update. By decoupling the rank (r) from the number of trainable parameters, Relayed LoRA breaks the linear dependency that has historically limited the efficiency of PEFT methods. This work provides a scalable path forward for the 'one model, millions of adapters' vision, enabling a more democratized and efficient landscape for LLM customization. In our experiments, the savings can be as high as 80% or even more. This reduction in parameters makes it more feasible to train LLMs on edge devices (Zhao et al., 2023) and facilitate remote fine-tuning (Yang et al., 2025). In the current work, we multiplied the four small matrices A_1 and A_2, and B_1 and B_2 to construct the large matrices A_{square} and B_{square}, respectively. Other operations could be used to build A_{square} and B_{square} from the four small matrices. It would also be interesting to establish a relationship between r and k in determining the optimal range for k so Relayed LoRA can obtain good performance while requiring smallest possible number of parameters.

Experiments demonstrate that Relayed LoRA largely preserves the fine-tuning performance of standard LoRA while significantly reducing the number of trainable parameters, and even outperforms it on certain datasets. However, the performance on some data sets is relatively low. The precise cause of this phenomenon remains unclear, but several factors may contribute to it. The first cause may be the sizes of datasets. Limited training data sets may cause the model to overfit by learning specific characteristics of the training data. The second cause could be the quality of training data and the difference between the original task that an LLM was trained on and the new task for which it was fine-tuned. For example, in the GLUE benchmark, different datasets originate from various sources, leading to variations in data quality, which could affect model training.

Limitations

A limitation of this study, constrained by computational resources, is that we did not initialize our models using LLMs that had been further pre-trained on large, task-specific corpora such as MNLI. Using the parameters from an LLM already trained on a large data set may lead to higher performance for both standard LoRA and Relayed LoRA.

Ethical considerations

The push for extreme parameter efficiency in Relayed LoRA could facilitate the rapid, low-cost proliferation of specialized malicious models, such as those used for generating sophisticated misinformation or deepfake text, by lowering the computational barrier for bad actors. Additionally, the algorithmic opacity introduced by the quad-matrix decomposition may make it harder to audit and mitigate embedded biases, potentially leading to the deployment of "black-box" models that reinforce systemic social prejudices under the guise of technical efficiency.

References

A Aghajanyan, S Gupta, and L Zettlemoyer. 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328. Association for Computational Linguistics.

S Babakniya, AR Elkordy, YH Ezzeldin, Q Liu, K-B Song, M El-Khamy, and S Avestimehr. 2023. Slora: Federated parameter efficient fine-tuning of language models. *arXiv preprint arXiv:2308.06522*.

YJ Cho, L Liu, Z Xu, A Fahrezi, and G Joshi. 2024. Heterogeneous LoRA for Federated Fine-tuning of On-Device Foundation Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12903–12913.

T Dettmers, A Pagnoni, A Holtzman, and L Zettlemoyer. 2023. QLORA: efficient finetuning of quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 10088–10115.

D Gao, Y Ma, S Liu, M Song, L Jin, W Jiang, X Wang, W Ning, S Yu, Q Xuan, and X Cai. 2024. FashionGPT: LLM instruction fine-tuning with multiple LoRA-adapter fusion. *Knowledge-Based Systems*, 299:112043.

Z Han, C Gao, J Liu, J Zhang, and SQ Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

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

C-Y Hsu, Y-L Tsai, C-H Lin, C-M Chen, P-Yand Yu, and C-Y Huang. 2024. Safe lora: The silver lining of reducing safety risks when finetuning large language models. *Advances in Neural Information Processing Systems*, 37:65072–65094.

EJ Hu, Y Shen, P Wallis, Z Allen-Zhu, Y Li, S Wang, L Wang, and W Chen. 2022. Lora: Low-rank adaptation of large language models. In *ICLR*, volume 1(2), page 3.

Z Hu, L Wang, Y Lan, W Xu, E-P Lim, L Bing, X Xu, S Poria, and R K-W Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*.

T Huang, S Hu, F Ilhan, S Tekin, and L Liu. 2024a. Lisa: Lazy safety alignment for large language models against harmful fine-tuning attack. *Advances in Neural Information Processing Systems*, 37:104521–104555.

T Huang, S Hu, F Ilhan, SF Tekin, and L Liu. 2024b. Harmful fine-tuning attacks and defenses for large language models: A survey. *arXiv preprint arXiv:2409.18169*.

Y Li, Y Yu, C Liang, P He, N Karampatziakis, W Chen, and T Zhao. 2023. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*.

V Lialin, N Shivagunde, S Muckatira, and A Rumshisky. 2023. ReLoRA: High-Rank Training Through Low-Rank Updates. In *arXiv preprint*, page 2307.05695.

L Lin, H Fan, Z Zhang, Y Wang, Y Xu, and H Ling. 2024. Tracking meets lora: Faster training, larger model, stronger performance. In *European Conference on Computer Vision*, pages 300–318. Springer.

S-Y Liu, C-Y Wang, H Yin, P Molchanov, Y-C F Wang, K-T Cheng, and M-H Chen. 2024. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*.

Y Liu, M Ott, N Goyal, J Du, M Joshi, D Chen, O Levy, M Lewis, L Zettlemoyer, and V Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. In *arXiv preprint*, page arXiv:1907.11692.

Y Mao, Y Ge, Y Fan, W Xu, Y Mi, Z Hu, and Y Gao. 2025. A survey on LoRA of large language models. *Frontiers of Computer Science*, 19(7):p.197605.

J Qi, Z Luan, S Huang, C Fung, H Yang, and D Qian. 2024. Fdlora: Personalized federated learning of large language model via dual lora tuning. *arXiv preprint arXiv:2406.07925*.

H Qin, X Ma, X Zheng, X Li, Y Zhang, S Liu, J Luo, X Liu, and M Magno. 2024. Accurate lora-finetuning quantization of llms via information retention. *arXiv preprint arXiv:2402.05445*.

B Runwal, T Pedapati, and P-Y Chen. 2025. From peft to deft: Parameter efficient finetuning for reducing activation density in transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20218–20227.

S Shi, S Huang, M Song, Z Li, Z Zhang, H Huang, F Wei, W Deng, F Sun, and Q Zhang. 2024. Reslora: Identity residual mapping in low-rank adaption. In *arXiv preprint*, page arXiv:2402.18039.

C Tian, Z Shi, Z Guo, L Li, and C-Z Xu. 2024. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:9565–9584.

M Valipour, M Rezagholizadeh, I Kobyzev, and A Ghodsi. 2022. DyLoRA: Parameter-Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation. *arXiv*, abs/2210.07558.

A Wang, A Singh, J Michael, F Hill, O Levy, and SR Bowman. 2018. GLUE:AMulti-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

B Wu, R Zhu, Z Zhang, P Sun, X Liu, and X Jin. 2024a. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 911–927.

F Wu, Z Li, Y Li, B Ding, and J Gao. 2024b. Fedbiot: Llm local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3345–3355.

Y Xia, F Fu, W Zhang, J Jiang, and B Cui. 2024. Efficient multi-task llm quantization and serving for multiple lora adapters. In *Advances in Neural Information Processing Systems*, volume 37, pages 63686–63714.

Y Xu, L Xie, X Gu, X Chen, H Chang, H Zhang, Z Chen, X Zhang, and Q Tian. 2023. Qa-lora: Quantization-aware low-rank adaptation of large language models. *arXiv preprint arXiv:2309.14717*.

S Yang, X Yu, R Li, J Zhu, Z Zhao, and H Zhang. 2025. AirLLM: Diffusion Policy-based Adaptive LoRA for Remote Fine-Tuning of LLM over the Air. *arXiv preprint arXiv:2507.11515*.

X Yang, J Leng, G Guo, J Zhao, R Nakada, L Zhang, H Yao, and B Chen. 2024. S^2ft: Efficient, scalable and generalizable llm fine-tuning by structured sparsity. *Advances in Neural Information Processing Systems*, 37:59912–59947.

Y Zeng and K Lee. 2023. The expressive power of low-rank adaptation. In *arXiv preprint*, page arXiv:2310.17513.

Q Zhang, M Chen, A Bukharin, P He, Y Cheng, W Chen, and T Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *arXiv preprint*, page arXiv:2303.10512.

W Zhao, Y Huang, X Han, Z Liu, Z Zhang, K Li, C Chen, T Yang, and M Sun. 2023. Ca-lora: Adapting existing lora for compressed llms to enable efficient multi-tasking on personal devices. *arXiv preprint arXiv:2307.07705*.