# LoRA-drop: Efficient LoRA Parameter Pruning based on Output Evaluation

**Anonymous submission**


## Abstract

Low-Rank Adaptation (LoRA) is currently the most commonly used Parameter-efficient fine-tuning (PEFT) method, it introduces auxiliary parameters for each layer to fine-tune the pre-trained model under limited computing resources. However, it still faces resource consumption challenges during training when scaling up to larger models. Most previous studies have tackled this issue by using pruning techniques, which involve removing LoRA parameters deemed unimportant. Nonetheless, these efforts only analyze LoRA parameter features to evaluate their importance, such as parameter count, size, and gradient. In fact, the output of LoRA (product of LoRA parameter and hidden state), directly impacts the final results. Preliminary experiments indicate that a fraction of LoRA elements possesses significantly high output values, substantially influencing the layer output. Motivated by the observation, we propose LoRA-drop. Concretely, LoRA-drop evaluates the importance of LoRA based on the LoRA output. Then we retain LoRA for important layers and the other layers share the same LoRA. We conduct abundant experiments with models of different scales on NLU and NLG tasks. Results demonstrate that LoRA-drop can achieve performance comparable to full fine-tuning and LoRA, while retaining 50% of the LoRA parameters on average.


## 1 Introduction

Parameter-efficient fine-tuning methods have attracted more and more attention with the development of large language models (LLM) (Li and Liang, 2021; Lester et al., 2021). Among various PEFT methods, LoRA (Hu et al., 2021) has been particularly prevalent in recent studies. LoRA freezes the pre-trained parameters and introduces auxiliary trainable parameters $\Delta\boldsymbol{W}$ for each layer as shown in Figure 1. LoRA significantly reduces the training cost while achieving impressive results.

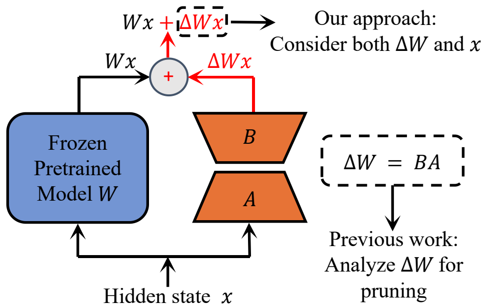


Figure 1: The diagram of LoRA. LoRA influences the pre-trained model through its output $\Delta\boldsymbol{W}\boldsymbol{x}$. This paper's method measures the importance of LoRA based on its output.

To further reduce the number of LoRA parameters being trained during efficient fine-tuning, previous studies employ pruning techniques that remove LoRA parameters deemed unimportant. The core of these methods lies in how to evaluate the importance of parameters. Sparse Adapter (He et al., 2022) evaluates the importance of LoRA based on different parameter features such as parameter count, parameter size, and parameter gradient. AdaLoRA (Zhang et al., 2022) designs importance criteria based on the singular value decomposition (SVD) of $\Delta\boldsymbol{W}$ to prune unimportant singular values. SoRA (Ding et al., 2023) prunes down-projection and up-projection matrices in LoRA by employing gate units and proximal gradient methods. All of these efforts only focus on analyzing LoRA parameter $\Delta\boldsymbol{W}$ features to evaluate their importance, thereby reducing the parameters required for LoRA training.

In fact, the output of LoRA, which is related to the parameters and data, directly impacts the final results. As shown in Figure 1, the LoRA adds a bias term $\Delta\boldsymbol{W}\boldsymbol{x}$ in each layer of the pre-trained model. Thus, the frozen model is fine-tuned by the bias term. Intuitively, if the norm of $\Delta\boldsymbol{W}\boldsymbol{x}$ is large, the LoRA of this layer has an important impact on the frozen model.

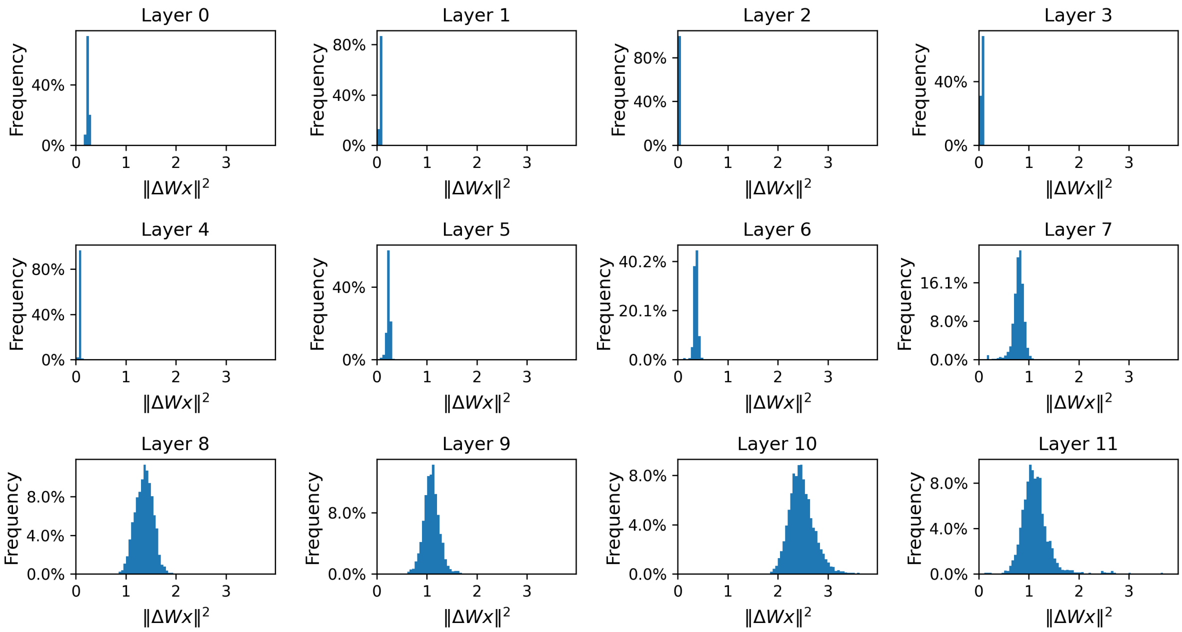


Figure 2: The frequency distribution of the squared norm of query LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ on the RTE task. Each subplot represents the distribution of $\|\Delta \boldsymbol{W}_i \boldsymbol{x}_i\|^2$ for query LoRA from layers 0 to 11, where the x-axis denotes the magnitude of $\|\Delta \boldsymbol{W}_i \boldsymbol{x}_i\|^2$ for different inputs $\boldsymbol{x}_i$, and the y-axis represents the frequency of $\|\Delta \boldsymbol{W}_i \boldsymbol{x}_i\|^2$.

We conducted an empirical study to analyze the distribution of LoRA output in LLMs. The findings derived from this study are presented in Section 2, revealing that the distribution of outputs from the LoRA of each layer is relatively concentrated. LoRA of some layers has little to no impact on specific tasks, while other layers exhibit more significant effects. Thus, we could prune non-salient LoRA parameters.

Motivated by the observation, we propose LoRA-drop, which evaluates the importance of parameters by analyzing the LoRA output for each layer. First, we sample specific task datasets and then utilize the sampled data to perform a limited number of updates to LoRA. The importance of LoRA for each layer is determined based on $\Delta \boldsymbol{W}\boldsymbol{x}$. Then, We retain the LoRA for layers with a large importance score, and the other layers share the same LoRA. Finally, we fine-tune the model with fewer trainable parameters under the new LoRA setting, while minimizing performance degradation.

Our contributions are as follows:

- We conducted empirical research, and the analysis indicates that the distribution of outputs from the LoRA of each layer is relatively concentrated. LoRA of some layers has little to no impact on specific tasks, while other layers exhibit more significant effects.
- We propose LoRA-drop, which evaluates the importance of LoRA for different layers and significantly reduces the parameter required during LoRA training while maintaining performance comparable to standard LoRA.
- We conduct comprehensive experiments on multiple NLU and NLG tasks with various sizes of pre-trained models. Numerous analysis experiments demonstrate the effectiveness of LoRA-drop.

## 2 Preliminary Experiment

LoRA utilizes the product of two low-rank matrices to simulate incremental updates to a full-rank matrix. The pre-trained parameters are frozen during training and do not receive gradient updates, while the two low-rank matrices are trained. Let $\boldsymbol{W}_i$ denote the query/key/value matrix of $i$th Transformer layer and $\boldsymbol{x}_i$ denote the input of the $i$th Transformer. The two low-rank matrices are $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$. Thus, the query/key/value vector is as follows:

$$\boldsymbol{h}_i = \boldsymbol{W}_i \boldsymbol{x}_i + \Delta \boldsymbol{W}_i \boldsymbol{x}_i = \boldsymbol{W}_i \boldsymbol{x}_i + \boldsymbol{B}_i \boldsymbol{A}_i \boldsymbol{x}_i \quad (1)$$

where $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ is the bias introduced by the LoRA modules.

Obviously, the $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ is the factor that directly influences the frozen pre-trained model. The larger

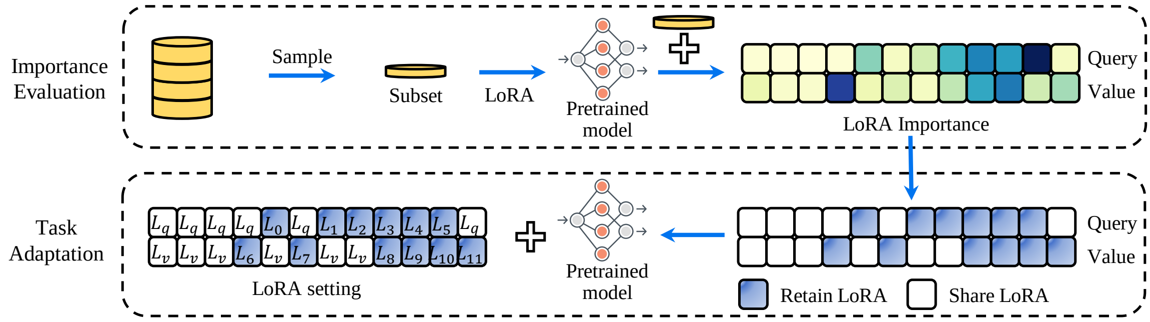


Figure 3: The overall workflow of LoRA-drop.

$\Delta \boldsymbol{W}_i \boldsymbol{x}_i$, the greater the impact of LoRA on the pre-trained model, and consequently, the more important LoRA is. In fact, the $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ is related to the LoRA parameter and the hidden state, where the hidden state is computed from downstream task data through the preceding layers of the model. However, previous work prunes LoRA by only analyzing its parameter features, ignoring the hidden state.

Preliminarily, we statistics the distribution of the LoRA output in each layer. Specifically, we fine-tune the RoBERTa-base model with LoRA separately on the RTE and MRPC dataset, and analyze the distribution of the squared norm of the LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ for each dataset. We evaluate the impact of LoRA by computing the squared norm of $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$. Following the setting of (Hu et al., 2021), the LoRA is added to the query and value matrix. The distribution of query and value LoRA for RTE is shown in Figure 2 and Figure 6. The distribution of query and value LoRA for MRPC is shown in Figure 7 and Figure 8.

As observed, the squared norm distribution of $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ for each layer is highly concentrated, showing a peak Gaussian frequency distribution, which suggests stability. Furthermore, Observations show that the squared norm of $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ for certain layers consistently remains close to zero, indicating that LoRA for these layers has almost no impact on the frozen model. Conversely, some layers show a more significant impact on the frozen model.

Moreover, RTE and MRPC exhibit different distribution patterns. It indicates that different layers play varying roles across different tasks.

This preliminary experiment demonstrates that we can prune the LoRA to reduce the number of trainable parameters. LoRA with small $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ is insignificant, and can be pruned.

## 3 Methodology

In this section, we introduce LoRA-drop, a novel parameter-efficient fine-tuning method that prunes based on LoRA output. We have designed a process to quantify the importance of LoRA for different layers based on its output. Then, we retain the more important LoRA and replace the less important ones with a shared LoRA parameter, thereby reducing the number of parameters required for LoRA training while maintaining performance comparable to that of the standard LoRA.

Specifically, LoRA-drop consists of two parts: **Importance Evaluation** and **Task Adaptation**. The overall process of LoRA-drop is illustrated in Figure 3.

### 3.1 Importance Evaluation

This step evaluates the importance of LoRA for different layers, providing a reference for its retention strategy in the Task Adaptation step.

Since the A and B matrices of LoRA are initialized with Kaiming and zero initialization, the initial output is all zeros. The output of LoRA becomes meaningful only after certain update steps.

So, we first perform stratified sampling on the downstream task dataset to obtain a subset $D_s$ of training data $D$. The sampling ratio is set to $\alpha$, where $0<\alpha<1$. After that, the LoRA parameters are updated with several steps using this subset.

Next, we compute the sum of the squared norm of the LoRA output for each layer, denoted as $\boldsymbol{g}$, the $\boldsymbol{g}$ of the $i$-th layer LoRA as expressed in Equation 2.

$$\boldsymbol{g}_i = \sum_{\boldsymbol{x} \in D_s} \|\Delta \boldsymbol{W}_i \boldsymbol{x}_i\|^2 \tag{2}$$

From section 2, the magnitude of $g$ reflects the importance of LoRA. To better represent the relative importance of LoRA for each layer, we normalized $g$, resulting in the importance $I$ for each layer of LoRA.

$$\boldsymbol{I}_i = \frac{\boldsymbol{g}_i}{\sum_i \boldsymbol{g}_i} \quad (3)$$

Thus, the importance of each layer of LoRA is bounded between 0 and 1, with a total sum of 1.

We find that sampling a small subset from the training data is able to obtain a LoRA importance distribution similar to that of the full dataset. This was verified by experiments in Section 4.3. Our experiments' default value of $\alpha$ is set to 10%.

### 3.2 Task Adaptation

This step sets the LoRA-drop fine-tuning strategy suitable for the downstream task based on the LoRA importance distribution.

With the importance of LoRA for each layer, we sort the layers according to $\boldsymbol{I}_i$. We select the layers from most to least important until the sum importance of the selected layer reaches a threshold $T$. In this paper, $T$ is set to 0.9 by default, and the value of $T$ is discussed in section 4.3.

The LoRA of these selected layers will be retained during training, while a shared LoRA parameter will replace the LoRA of the other layers. The hyper-parameter $T$ controls the number of the selected layers. Finally, we fine-tune the model using the training dataset under the new LoRA setting.

## 4 Experiments

### 4.1 Setup

**Datasets.** We evaluate our model on both Natural Language Understanding (NLU) and Natural Language Generation (NLG) tasks.

For NLU, we evaluate our method on the GLUE benchmark (Wang et al., 2018), which consists of eight datasets: CoLA, SST-2, MRPC, QQP, STS-B, MNLI, QNLI, and RTE. We use Matthew's correlation coefficient, Spearman's correlation coefficient, and overall accuracy (for both matched and mismatched sentences) to evaluate the CoLA, STS-B, and MNLI datasets. For the remaining datasets, we apply accuracy as the evaluation metric.

The NLG tasks in our experiments include the table-to-text datasets E2E (Dušek et al., 2020) and DART (Nan et al., 2021), the summarization dataset DialogSum (Chen et al., 2021), as well as the Mathematical Reasoning dataset GSM8K (Cobbe et al., 2021). We use BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), and accuracy to evaluate the E2E(&DART), DialogSum, and GSM8K datasets.

**Baselines.** The following methods are chosen as baselines: **FULL-FT** updates all model parameters which need a lot of computing resources. **LoRA** (Hu et al., 2021) represents the original LoRA method. **Sparse Adapter** (He et al., 2022) applies typical pruning methods to LoRA and reduces the trainable parameters. **VeRA** (Kopiczko et al., 2024) shares and freezes randomly initialized LoRA and introduces trainable vectors for each layer to reduce the parameters of LoRA. **Tied-LoRA** (Renduchintala et al., 2023) makes the frozen LoRA in VeRA trainable. **SoRA** (Ding et al., 2023)uses a gate unit with proximal gradient methods to control LoRA's sparsity.

**Models & Implementation.** To evaluate the effectiveness of our method on various models, we conduct experiments on RoBERTa-base, RoBERTa-large(Liu et al., 2019), and Llama2-7b(Touvron et al., 2023). We conduct NLU experiments on the GLUE benchmark using all three models. We performed 3 runs with different random seeds for each dataset and recorded the best results for each run. The average results and the standard deviation are calculated.

To evaluate the effectiveness of our method on generation tasks, we conduct NLG experiments using the Llama2-7b on the table2text datasets: E2E and DART, the summarization dataset DialogSum, as well as the Mathematical Reasoning dataset GSM8K.

The hyperparameter settings for each baseline and LoRA-drop can be found in Section A.1.

### 4.2 Main Results

The main results of RoBERTa-base with different training methods on the GLUE benchmark are shown in Table 1. It is noted that our motivation is to reduce the number of trainable parameters while ensuring that the performance does not degrade, or even improve. As shown in Table 1, with an approximately 50% reduction in standard LoRA parameters, our proposed LoRA-drop achieves an average score of 86.2, on par with Full-FT (86.4) and LoRA (86.1). This indicates the effectiveness of our proposed LoRA-drop, which outperforms LoRA by 0.1 scores while reducing training parameters.

| Model RoBERTa-base | #Tr. Params | RTE (Acc) | MRPC (Acc) | STS-B (Spea.) | CoLA (Matt.) | SST-2 (Acc) | QNLI (Acc) | MNLI (Acc) | QQP (Acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Full-FT* | 125M | 78.7 | **90.2** | **91.2** | $\underline{63.6}$ | **94.8** | 92.8 | **87.6** | **91.9** | **86.4** |
| LoRA | 0.29M | $\underline{80.8}_{\pm 1.5}$ | $89.1_{\pm 0.6}$ | $\mathbf{91.2}_{\pm 0.2}$ | $62.4_{\pm 0.7}$ | $94.3_{\pm 0.3}$ | $\underline{93.0}_{\pm 0.2}$ | $\underline{87.5}_{\pm 0.2}$ | $\underline{90.3}_{\pm 0.1}$ | 86.1 |
| SoRA | 0.21M | $79.7_{\pm 0.7}$ | $\underline{89.7}_{\pm 1.0}$ | $89.8_{\pm 0.1}$ | $\mathbf{63.8}_{\pm 1.0}$ | $\mathbf{94.8}_{\pm 0.4}$ | $92.4_{\pm 0.3}$ | $86.1_{\pm 0.1}$ | $88.9_{\pm 0.3}$ | 85.6 |
| Sparse Adapter | 0.15M | $78.7_{\pm 1.1}$ | $88.0_{\pm 0.5}$ | $89.5_{\pm 0.4}$ | $60.1_{\pm 0.7}$ | $94.1_{\pm 0.1}$ | $92.8_{\pm 0.1}$ | $87.1_{\pm 0.2}$ | $89.6_{\pm 0.1}$ | 85.0 |
| VeRA | 0.03M | $78.0_{\pm 1.1}$ | $88.4_{\pm 0.1}$ | $89.8_{\pm 0.2}$ | $60.9_{\pm 0.5}$ | $93.7_{\pm 0.1}$ | $89.6_{\pm 0.1}$ | $83.7_{\pm 0.1}$ | $86.8_{\pm 0.1}$ | 83.9 |
| Tied-LoRA | 0.15M | $80.0_{\pm 0.9}$ | $89.1_{\pm 0.6}$ | $90.3_{\pm 0.1}$ | $62.0_{\pm 0.8}$ | $94.1_{\pm 0.3}$ | $91.6_{\pm 0.4}$ | $86.9_{\pm 0.1}$ | $89.7_{\pm 0.1}$ | 85.5 |
| LoRA-drop (ours) | 0.15M | $\mathbf{81.4}_{\pm 0.5}$ | $89.5_{\pm 0.5}$ | $\underline{91.0}_{\pm 0.1}$ | $62.9_{\pm 0.2}$ | $\underline{94.5}_{\pm 0.2}$ | $\mathbf{93.1}_{\pm 0.1}$ | $87.3_{\pm 0.2}$ | $90.1_{\pm 0.1}$ | $\underline{86.2}$ |

Table 1: Results of the RoBERTa-base with different training strategies on the GLUE benchmark. The results are averaged from three seeds to produce solid results. The subscript is the standard deviation. Bold and underlined indicate the first and second best results in the corresponding regime. #Tr. refers to trainable. * refers to the results directly from their original paper, in which Full-FT is derived from (Liu et al., 2019).

| Model Llama2 7b | #Tr. Params | E2E (BLEU) | DART (BLEU) | Dialogsum (ROUGE) | GSM8K (Acc) | Avg. |
|---|---|---|---|---|---|---|
| Full-FT | 6.6B | 55.65 | **59.68** | 40.77 | 31.16 | 46.82 |
| LoRA | 0.13B | 56.38 | 58.51 | **41.03** | 34.04 | 47.49 |
| LoRA-drop (ours) | 0.09B | **57.06** | 58.82 | 40.68 | **34.50** | **47.77** |

Table 2: Results of Llama2-7b with different training strategies on two table2text datasets including E2E and DART, the summarization dataset Dialogsum, and the mathematical reasoning dataset GSM8K. For all the scores, BLEU, ROUGE, and Acc, higher is better.

Moreover, LoRA-drop achieves 0.6, 1.2, 2.3, and 0.7 improvements in average scores compared to the four baselines: SoRA, Sparse Adapter, VeRA, and Tied-LoRA respectively. Although all four methods effectively reduce LoRA parameters, their performance drops significantly. The results demonstrate that LoRA-drop is a superior strategy for evaluating the importance of trainable parameters and reducing less important ones, thereby enhancing parameter efficiency.

The results of RoBERTa-large and Llama2-7b with different training strategies on the GLUE benchmark are presented in Table 6 and Table 7. It is noted that we use Llama2-7b to obtain the token representation rather than generate the answer. On both models, our method utilizes about 52% of the standard LoRA parameters and achieves average scores of 89.1 and 89.3 for RoBERTa-large and Llama2-7b respectively, outperforming LoRA and Full-FT. This demonstrates the effectiveness of our method across models of different scales.

The results of NLG tasks, including table2text, summarization, and mathematical reasoning, are shown in Table 2. On Llama2-7b, our method achieves results on par with the Full-FT and LoRA while using approximately 68% of the original LoRA parameters for all three tasks. Additionally, the average score of our method (47.77) exceeds that of Full-FT (46.82) and LoRA (47.49). This confirms the effectiveness of our method across both NLU and NLG.

### 4.3 Analysis

**The value of LoRA output indicated the importance.** As described in Section 3.1, the importance evaluation step quantifies the importance of LoRA based on its output. In this section, we verify the effectiveness of the output-based evaluation method. Specifically, we first perform standard LoRA fine-tuning and obtain the importance score. Based on this score, we retain either the largest or the smallest of the LoRA layers for inference, the number of retained LoRA is consistent with the number retained by LoRA-drop in Section 4.2. We then evaluate these two settings, and the final results are presented in Table 3.

It is evident that when only approximately half of the LoRA modules are retained, the model's performance decreases significantly. When we retain the LoRA modules with larger $I$, the performance is substantially better than those with smaller $I$. This indicates that the LoRA-drop method's layer-specific LoRA Importance Evaluation is effective. LoRA with a larger squared norm output indeed has a greater contribution to the model's fine-tuning performance.

| Model (RoBERTa-base) | RTE (Acc) | MRPC (Acc) | STS-B (Spea.) | CoLA (Matt.) | SST-2 (Acc) | QNLI (Acc) | MNLI (Acc) | QQP (Acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| LoRA | 79.4 | 89.2 | 91.0 | 63.1 | 94.6 | 92.7 | 87.6 | 90.3 | 86.0 |
| LoRA(large $I$) | 72.2 | 77.5 | 85.9 | 58.9 | 92.9 | 73.6 | 71.2 | 82.6 | 76.9 |
| LoRA(small $I$) | 47.7 | 69.9 | 49.6 | 23.5 | 88.2 | 55.4 | 32.2 | 63.9 | 53.8 |

Table 3: Verification of Importance Evaluation Method. The data in the table represents the results from a single run with the same random seed. LoRA (large $I$) retains the few LoRAs with the highest $I$ values, while LoRA (small $I$) retains the few with the lowest $I$ values. The number retained is consistent with the LoRA-drop setting in Table 1.

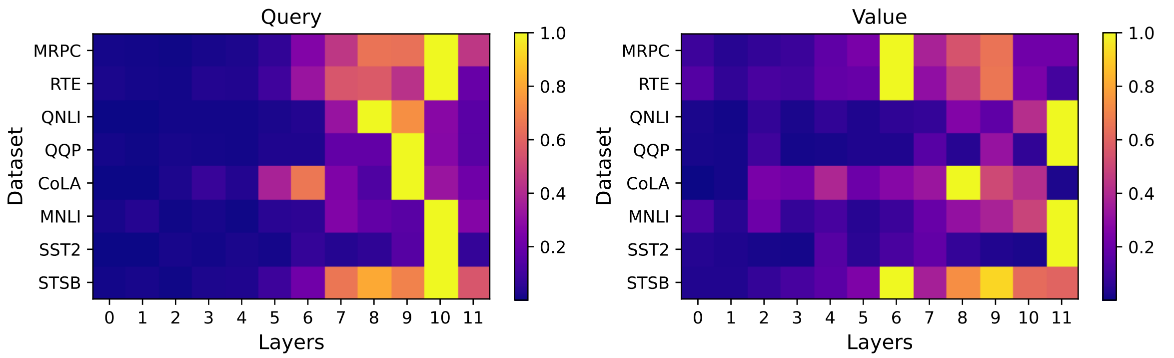


Figure 4: LoRA Importance Distribution in Different Downstream Task Data. To unify the importance scales across different datasets, we divide the importance of each dataset by its maximum value so that the importance of the most important layer of LoRA in that dataset is 1.

**Distribution of LoRA importance varies across different tasks.** The insight of our approach is to derive LoRA importance adapted to the distribution of different downstream task data, enabling the simplification of LoRA parameters. To further validate the rationality of this insight, we plot heatmaps illustrating the distribution of LoRA importance $\boldsymbol{I}$ for eight different datasets in GLUE on RoBERTa-base and Llama2.

The results are presented in Figure 4 and Figure 11. We observe that the importance distributions differ across datasets, indicating that the importance assigned by LoRA is data-dependent.

**The influence of LoRA share.** In our method, the layers with low importance are shared with the same LoRA parameters. We investigate the influence after the LoRA parameters are shared. Following the LoRA share operation on the RoBERTa-base model trained on 20% of the RTE training set data for 4 epochs, we plot the importance distribution for each layer of the model.

The results of query and value distribution are shown in Figure 9 and Figure 10. It shows that the importance distribution of LoRA for each layer remains almost consistent with the original LoRA after the LoRA parameters are shared. This suggests that the sharing LoRA for layers with low importance does not affect the importance distribution of other layers, thereby maintaining good performance.

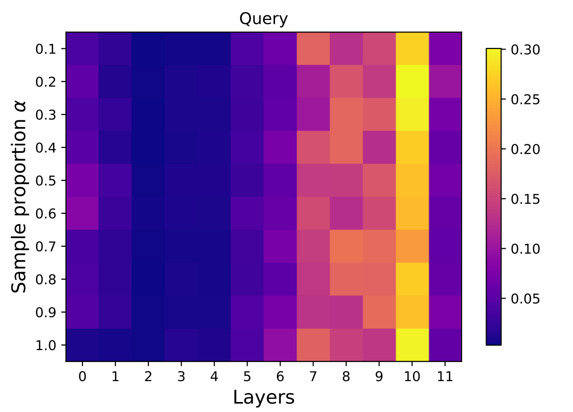


Figure 5: Importance distribution of LoRA for query in RTE under different sample proportions. Each point on the heatmap represents the importance $I_i$ of the query LoRA in layer $i$ under $\alpha$ sample proportion.

**The influence of sample proportion.** We investigate the influence of the sample proportion when calculating the importance of LoRA. We sample ten different-sized datasets from the RTE dataset with sampling ratios from 10% to 100%. We train the RoBERTa-base model using LoRA for the same

| Threshold | Avg. layer num W_query | Avg. layer num W_value | RTE (ACC) | CoLA (Matt.) | QNLI (ACC) | QQP (ACC) | Avg. |
|---|---|---|---|---|---|---|---|
| 1(LoRA) | 12 | 12 | 82.3 | 61.9 | 93.1 | 90.4 | 82.0 |
| 0.95 | 6 | 9 | 83.0 | 62.6 | 93.1 | 90.2 | 82.2 |
| 0.9 | 5 | 7 | 81.9 | 63.1 | 93.2 | 90.2 | 82.1 |
| 0.8 | 5 | 5 | 80.9 | 63.1 | 93.2 | 89.6 | 81.7 |
| 0.7 | 4 | 4 | 78.3 | 62.1 | 92.5 | 89.3 | 80.6 |

Table 4: The influence of the threshold $T$ and its equivalent average number of layers.

| Model (RoBERTa-base) | RTE (Acc) | MRPC (Acc) | STS-B (Spea.) | CoLA (Matt.) | SST-2 (Acc) | QNLI (Acc) | MNLI (Acc) | QQP (Acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| LoRA-drop* | 81.9 | 90.0 | 91.1 | 63.1 | 94.7 | 93.2 | 87.5 | 90.2 | 86.5 |
| LoRA-drop(w/o share) | 80.4 | 88.9 | 90.7 | 62.8 | 94.1 | 92.9 | 86.9 | 89.7 | 85.8 |
| LoRA-drop($\Delta Wx$ inv) | 79.1 | 89.7 | 90.4 | 60.5 | 94.3 | 92.9 | 87.3 | 89.9 | 85.5 |
| LoRA-drop(random) | 79.1 | 89.2 | 90.2 | 62.0 | 94.6 | 92.7 | 86.9 | 89.8 | 85.6 |
| LoRA-drop(top $k$) | 81.9 | 89.2 | 90.7 | 62.3 | 94.5 | 93.0 | 86.8 | 89.8 | 86.0 |

Table 5: Ablation experiments.

number of steps and obtain the LoRA importance for each sample proportion.

The results of LoRA for Query and Value are shown in Figure 5 and Figure 12. As the training data increases, the importance order of each layer remains consistent. For LoRA applied to the query matrices, the 10th layer has always been the most important, while the importance of layers 7, 8, and 9 maintains a consistently high level of importance. Indicating that this operation is insensitive to the size of the sampled data and exhibits robustness.

**Selection of threshold $T$.** LoRA-drop introduces a hyper-parameter $T$ to control the number of selected layers. We select four datasets from GLUE to analyze the impact of threshold $T$.

The results are shown in Table 4. When T is set to 1, all layers are preserved, representing the standard LoRA method. When $T$ is less than 0.9, the model performance increases with $T$, at this time, LoRA modules with higher importance are selected. When $T$ equals 0.9, approximately half of the layers' LoRA are selected on average. If $T$ continues to increase, the newly added LoRA modules have lower importance, and the model performance no longer significantly improves. Hence in our experiments, we default to setting $T$ as 0.9.

### 4.4 Ablation Study

In this subsection, we conduct ablation experiments to verify the following two questions:

- Q1: Is replacing LoRA for layers with small $\boldsymbol{I}$ with shared parameters better than directly removing them in the task adaptation step?
- Q2: Is retaining LoRA with large $\boldsymbol{I}$ in the task adaptation step reasonable?

To answer these two questions, we compare LoRA-drop with the following variants on the RoBERTa-base model, where $k$ refers to the number of LoRA retained by LoRA-drop.

**LoRA-drop (w/o share)** directly removes the low-importance layers of LoRA without using additional shared parameters in the Task Adaptation step. As opposed to LoRA-drop, **LoRA-drop ($\Delta \boldsymbol{W}\boldsymbol{x}$ inv)** replace high-importance layers of LoRA with shared LoRA and retain the other LoRA. **LoRA-drop (random)** randomly selects $k$ layers that retain LoRA parameters. Houlsby et al. (2019) found that lower layers often have a small impact on performances, so **LoRA-drop (top $k$)** selects the top $k$ layers of the 12-layer model. We experiment with these four settings on the validation set of the GLUE benchmark.

The results are shown in Table 5.

**Regarding Q1**, directly removing less important LoRA parameters, i.e., the LoRA-drop (w/o share) setting, performs worse across all tasks than LoRA-drop, with an average reduction of 0.7 scores.

This indicates that sharing a LoRA among the layers with low importance is necessary to achieve better fine-tuning results compared to directly removing them.

**Regarding Q2**, the $\Delta \boldsymbol{W}\boldsymbol{x}$ inv setting achieved the worst average performance, slightly worse than the random setting. This indicates that LoRA with smaller $I$ contributes less to model performance improvement. The top $k$ setting, which empirically retains the top $k$ layers, performed well but had an average performance gap of 0.5 scores compared to the LoRA-drop.

LoRA-drop yields better performance compared to all the other three variants. It confirms the reasonableness of retaining the LoRA of layers with significant importance and further validates the effectiveness of the method proposed in this paper for evaluating the importance of LoRA.

# 5 Related Work

Parameter Efficient Fine-Tuning (PEFT) is the mainstream method for the current fine-tuning of pre-trained models, which can be broadly categorized into additive methods, selective methods, and reparameterized (Han et al., 2024).

## 5.1 Additive Methods

Additive methods inject new trainable modules or parameters into pre-trained models. During fine-tuning for a specific downstream task, only the weights of these newly added modules are updated.

Adapter (Houlsby et al., 2019) involves inserting small adapter layers within Transformer blocks. There are two ways to inject adapters into pre-trained models: Serial Adapter (Houlsby et al., 2019) adds two adapter modules in each Transformer block. Parallel Adapter (He et al., 2021) transforms the serial adapter layers into parallel side networks. Adapter Drop (Rücklé et al., 2021) empirically removes lower-layer Adapters considered to have a small impact on task performance.

Soft Prompt uses continuous embedding of soft prompts instead of optimizing discrete token representations through in-context learning. Prefix-tuning (Li and Liang, 2021) inserts trainable vectors prepended to keys and values at all Transformer layers. P-tuning (Liu et al., 2021) and Prompt-tuning (Lester et al., 2021) only insert trainable vectors at the initial word embedding layer.

## 5.2 Selective Methods

Selective methods make a small subset of parameters in the pre-trained model trainable while keeping the rest frozen. Diff pruning (Guo et al., 2021) employs a learnable binary mask on model weights. BitFit (Zaken et al., 2022) only fine-tunes the bias parameters of each FFN, achieving competitive results for small models. Lee et al. (2019) fine-tunes only the last quarter of BERT and RoBERTa's final layer, achieving 90% of the performance of full fine-tuning. HiFi (Gui and Xiao, 2023) fine-tunes attention heads that are highly informative and strongly correlated for a specific task.

## 5.3 Reparameterized Methods

In the context of PEFT, reparameterization often involves constructing a low-rank parameterization to enhance parameter efficiency during training.

LoRA (Hu et al., 2021) introduces low-rank matrices during fine-tuning and can merge with pre-trained weights before inference. There are many derivative works based on LoRA. QLoRA (Dettmers et al., 2023) quantifies the parameters of large models doubly, significantly reducing memory usage. AdaLoRA (Zhang et al., 2022) transforms the low-rank matrices in LoRA into SVD matrices $P\Lambda Q$. During training, the singular values are iteratively pruned. SoRA (Ding et al., 2023) eliminates the matrix orthogonality premise of $P$ and $Q$ in AdaLoRA and instead applies a gating unit between them. Sparse Adapter (He et al., 2022) enhances the parameter efficiency of LoRA and other Adapters using network pruning methods. S2-LoRA (Liu et al., 2023) shares the LoRA parameters, and introduces trainable scaling vectors with inter-layer variations. VeRA (Kopiczko et al., 2024) and Tied-LoRA (Renduchintala et al., 2023), further reduce the parameter count by sharing parameters for all layers and modules of LoRA. DoRA (Liu et al., 2024) uses LoRA for directional updates, enhancing learning capacity and training stability.

# 6 Conclusion

In this paper, we propose a new parameter-efficient fine-tuning method LoRA-drop based on LoRA. our motivation is to reduce the number of trainable parameters during fine-tuning while ensuring that the performance does not degrade, or even improve. Concretely, we calculate the importance of LoRA for each layer based on the output. The LoRA parameters of layers with large importance are retained and the other layers share the same parameter, resulting in a significant reduction in the number of parameters that need to be trained compared to the original LoRA. Abundant experiments on multiple NLU and NLG datasets show that LoRA-drop can achieve comparable results with origin LoRA with 50% of LoRA parameters.

## Limitations

Currently, our method operates on the LoRA structure as a whole, with a relatively coarse granularity. Future work will refine this method to a finer granularity. While this technique reduces the number of training parameters during LoRA training, it does not decrease the inference cost. Pruning increases the model's complexity, making it more difficult to identify the sources of issues when performance falls short of expectations. This, in turn, complicates the processes of debugging and error analysis.

## References


Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021. DialogSum: A real-life scenario dialogue summarization dataset. In *Findings of the Association for Computational Linguistics*, pages 5062–5074.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Proceedings of the Conference on Neural Information Processing Systems*.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the State-of-the-Art of End-to-End Natural Language Generation: The E2E NLG Challenge. *Computer Speech & Language*, pages 123–156.

Anchun Gui and Han Xiao. 2023. HiFi: High-information attention heads hold for parameter-efficient model adaptation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 8521–8537.

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the annual Meeting of the Association for Computational Linguistics*.

Zeyu Han, Chao Gao, Jinyang Liu, Sai Qian Zhang, et al. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. In *Proceedings of the International Conference on Learning Representations*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. 2022. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics*, pages 2184–2190.

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 *Proceedings of the International Conference on Machine Learning*, pages 2790–2799.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. 2024. VeRA: Vector-based random matrix adaptation. In *Proceedings of the Annual Meeting of the International Conference on Learning Representations*.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 4582–4597.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

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*.

Wei Liu, Ying Qin, Zhiyuan Peng, and Tan Lee. 2023. Sparsely shared lora on whisper for child speech recognition. *arXiv preprint arXiv:2309.11756*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-domain structured data record to text generation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 432–447.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the annual meeting of the Association for Computational Linguistics*, pages 311–318.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2023. Tied-lora: Enhacing parameter efficiency of lora with weight tying. *arXiv preprint arXiv:2311.09578*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. Adapterdrop: On the efficiency of adapters in transformers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the International Conference on Learning Representations*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1–9.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2022. Adaptive budget allocation for parameter-efficient fine-tuning. In *Proceedings of the International Conference on Learning Representations*.

# A Appendix

## A.1 Implementation Details

Our LoRA configuration aligns with the experimental setup of (Hu et al., 2021), where LoRA is applied to the query and value matrices in each self-attention module. We each use a shared LoRA in place of the low-importance query LoRA and value LoRA.

The low-rank matrix $A$ of the LoRA architecture is initialized using Kaiming initialization (He et al., 2015), while matrix $B$ is initialized with zeros. Unless specified otherwise, the default rank for LoRA is set to 8.

We conducted NLU experiments on the GLUE benchmark using RoBERTa-base (Liu et al., 2019). The data sampling ratio $\alpha$ is set to 0.1, the number of training epochs $n$ is set to 3, and the threshold $T$ for LoRA-drop is set to 0.9. To ensure consistency in the trainable parameter count between the baseline and our method, we set the sparsity rate of the Sparse Adapter to 0.5. We set the pruning method of the Sparse Adapter to the performance-optimal SNIP (Lee et al., 2018). The rank of Tied-LoRA is set to 56. The design characteristics of the VeRA method determine that its trainable parameter count cannot reach the same order of magnitude as LoRA; otherwise, VeRA would no longer be a low-rank matrix. Therefore, we set the rank of VeRA to 512 based on the best hyperparameters provided in the original paper.

To evaluate the effectiveness of our method on generation tasks, we conducted NLG experiments using the Llama2 7b on the table2text datasets: E2E and DART, the summarization dataset DialogSum, as well as the mathematical reasoning dataset GSM8K. For all three tasks, we set the rank of LoRA to 64. It is worth noting that, in the NLG experiment we applied LoRA to the query, key, value, and output matrices in Attention, and up and down matrices in MLP, as we found that only fine-tuning the query and value matrices with LoRA would cause significant performance degradation.

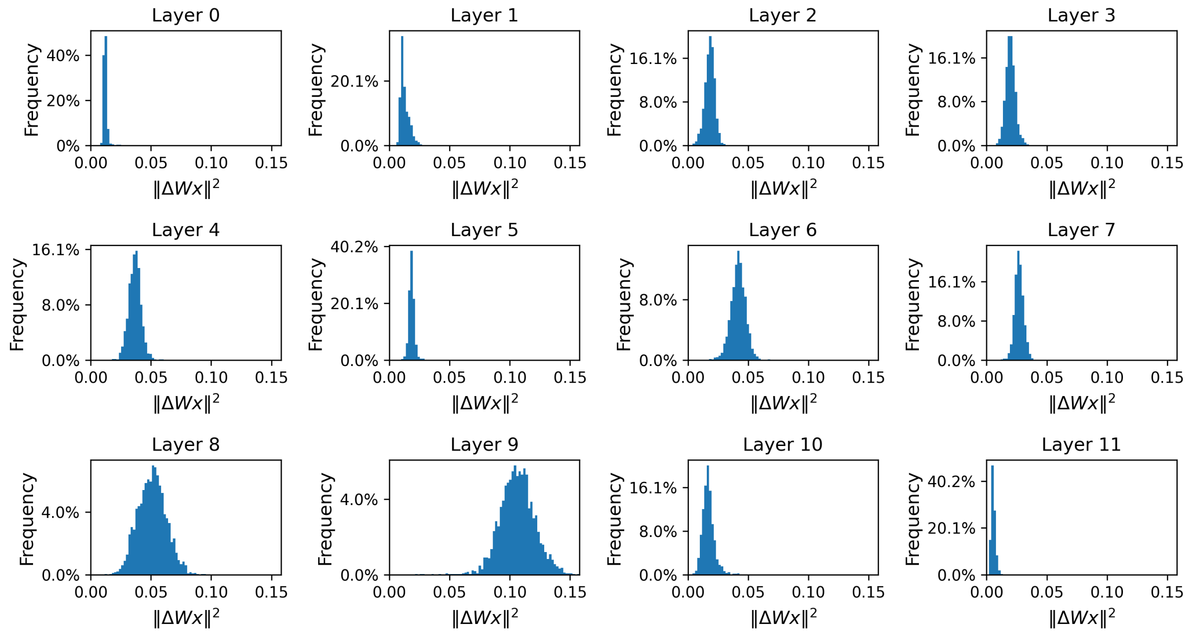


Figure 6: The frequency distribution of the squared norm of value LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ after fine-tuning on the RTE task.

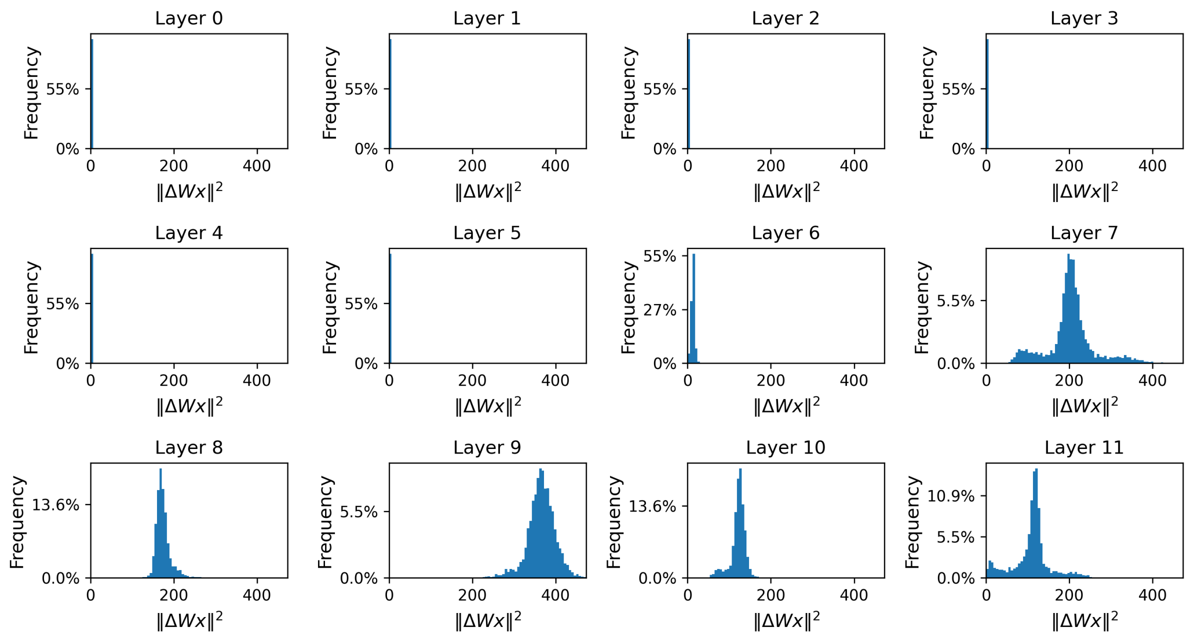


Figure 7: The frequency distribution of the squared norm of query LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ after fine-tuning on the MRPC task.

| Model RoB-large | #Tr. Params | RTE (Acc) | MRPC (Acc) | STS-B (Spea.) | CoLA (Matt.) | SST-2 (Acc) | QNLI (Acc) | MNLI (Acc) | QQP (Acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Full-FT* | 355M | 86.6 | **90.9** | **92.4** | 68.0 | **96.4** | 94.7 | 90.2 | **92.2** | 88.9 |
| LoRA | 0.79M | 88.5$_{\pm0.7}$ | 90.1$_{\pm0.8}$ | **92.4**$_{\pm0.1}$ | 67.8$_{\pm1.3}$ | 96.0$_{\pm0.1}$ | 94.8$_{\pm0.1}$ | 90.6$_{\pm0.0}$ | 91.4$_{\pm0.1}$ | 88.9 |
| LoRA-drop (ours) | 0.41M | **88.8**$_{\pm0.7}$ | 89.9$_{\pm0.3}$ | 92.2$_{\pm0.1}$ | **68.5**$_{\pm1.7}$ | 96.2$_{\pm0.1}$ | **94.9**$_{\pm0.1}$ | **90.7**$_{\pm0.1}$ | 91.3$_{\pm0.5}$ | **89.1** |

Table 6: The performance of the RoBERTa-large on GLUE benchmark. * refers to the results directly from their original paper, in which Full-FT is derived from (Liu et al., 2019).

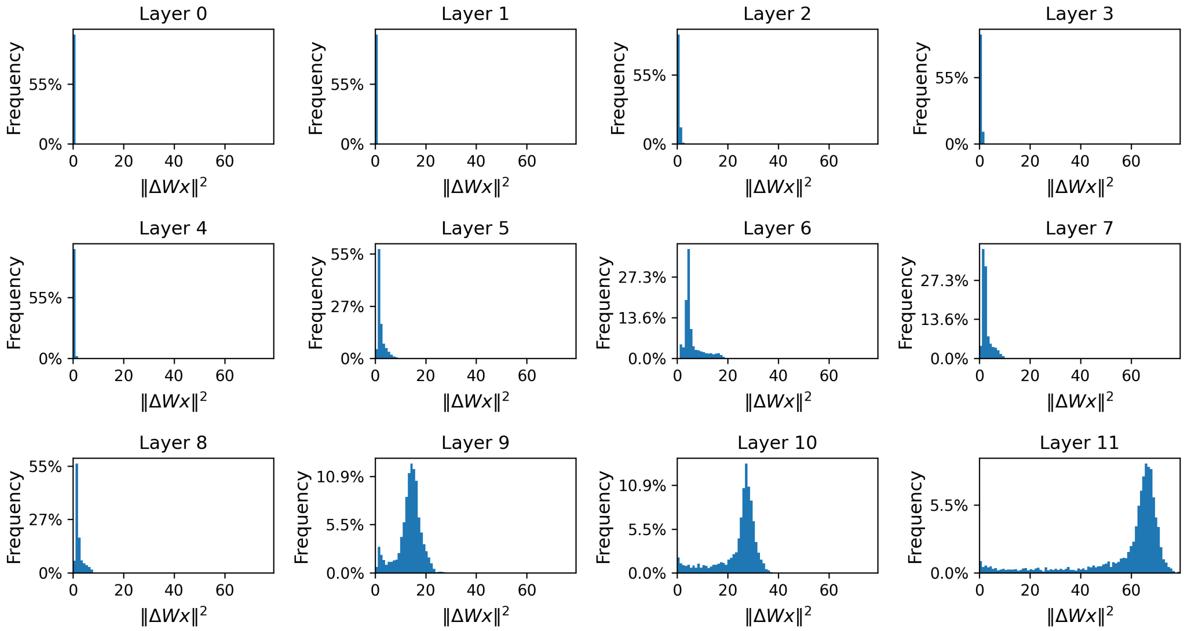


Figure 8: The frequency distribution of the squared norm of value LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ after fine-tuning on the MRPC task.

| Model<br>Llama2 7b | #Tr.<br>Params | RTE<br>(Acc) | MRPC<br>(Acc) | STS-B<br>(Spea.) | CoLA<br>(Matt.) | SST-2<br>(Acc) | QNLI<br>(Acc) | MNLI<br>(Acc) | QQP<br>(Acc) | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Full-FT | 6.6B | 88.4 | 88.7 | 89.8 | 67.9 | $\underline{92.3}$ | 93.6 | 86.3 | **91.7** | 87.3 |
| LoRA | 4.2M | $\underline{89.2}_{\pm 0.5}$ | $\underline{89.7}_{\pm 0.5}$ | $\underline{89.9}_{\pm 0.1}$ | $\mathbf{70.6}_{\pm 0.7}$ | $\mathbf{96.8}_{\pm 0.2}$ | $\underline{94.7}_{\pm 0.2}$ | $\mathbf{90.9}_{\pm 0.2}$ | $\underline{91.6}_{\pm 0.1}$ | $\underline{89.2}$ |
| LoRA-drop (ours) | 2.2M | $\mathbf{91.0}_{\pm 0.5}$ | $\mathbf{90.2}_{\pm 0.3}$ | $\mathbf{90.1}_{\pm 0.1}$ | $\underline{69.0}_{\pm 1.2}$ | $\mathbf{96.8}_{\pm 0.2}$ | $\mathbf{94.8}_{\pm 0.2}$ | $\underline{90.6}_{\pm 0.1}$ | $\underline{91.6}_{\pm 0.3}$ | **89.3** |

Table 7: The performance of the Llama2-7b on GLUE benchmark.

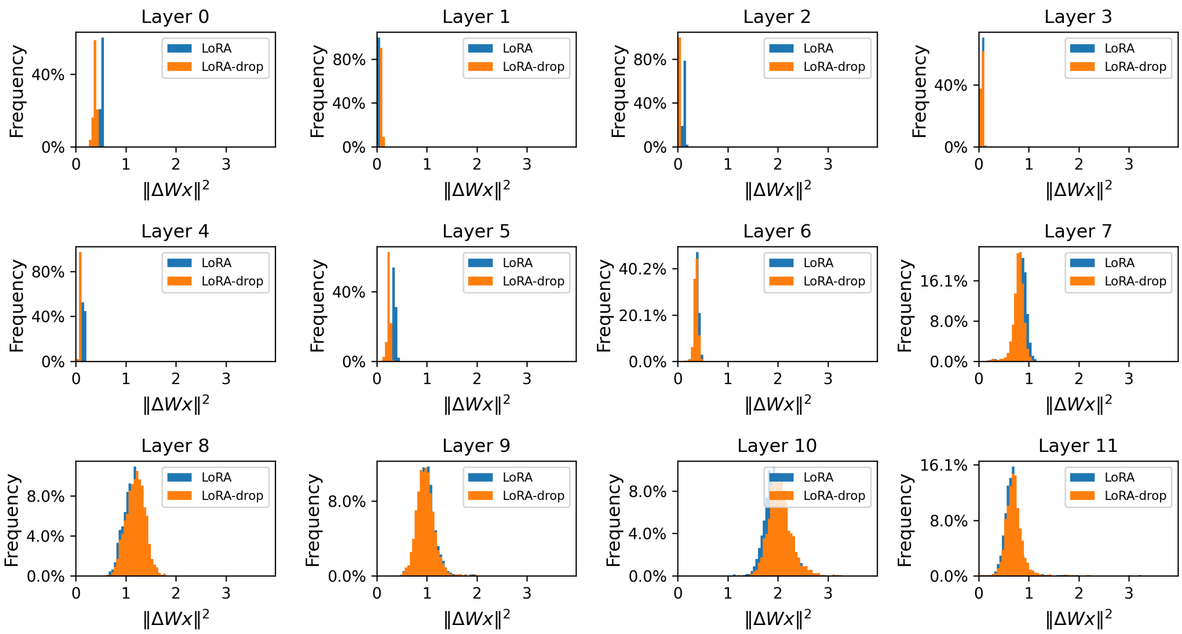


Figure 9: The query LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ squared norm frequency distribution of LoRA and LoRA-drop.

| Model<br>Llama2 7b | #Tr.<br>Params | Dialogsum<br>ROUGE-1 | <br>ROUGE-2 | <br>ROUGE-L | <br>Avg. |
|---|---|---|---|---|---|
| Full-FT | 6.6B | 49.86 | **29.37** | 43.07 | 40.77 |
| LoRA | 0.13B | **50.15** | 29.28 | **43.65** | **41.03** |
| LoRA-drop (ours) | 0.09B | 49.84 | 28.99 | 43.22 | 40.68 |

Table 8: Results of Llama2-7b with different training strategies on the summarization dataset Dialogsum.

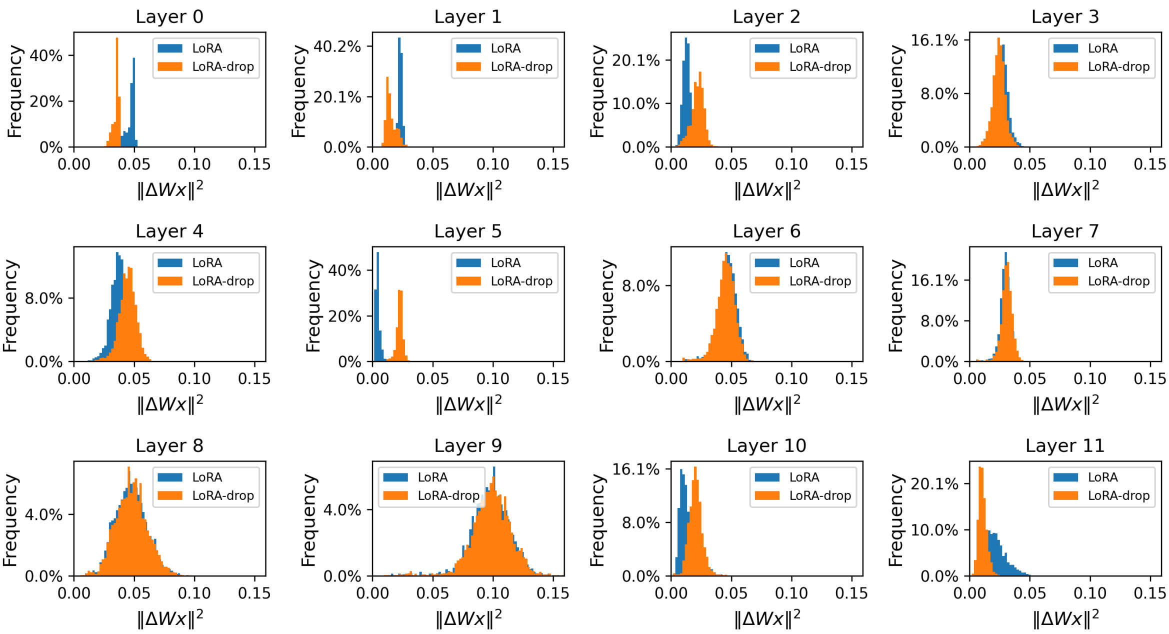

Figure 10: The value LoRA output $\Delta \boldsymbol{W}_i \boldsymbol{x}_i$ squared norm frequency distribution of LoRA and LoRA-drop.

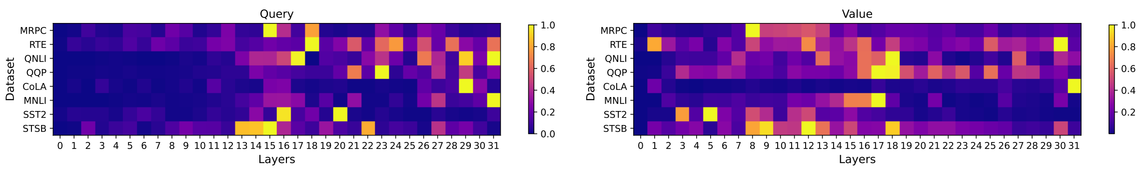

Figure 11: The relative magnitudes of LoRA outputs across different layers of Llama2-7b on various datasets. The left subplot shows the LoRA outputs corresponding to each layer's query matrix, and the right subplot shows the LoRA outputs corresponding to each layer's value matrix. For display, the value of the largest layer's LoRA output is normalized to 1 for each dataset.

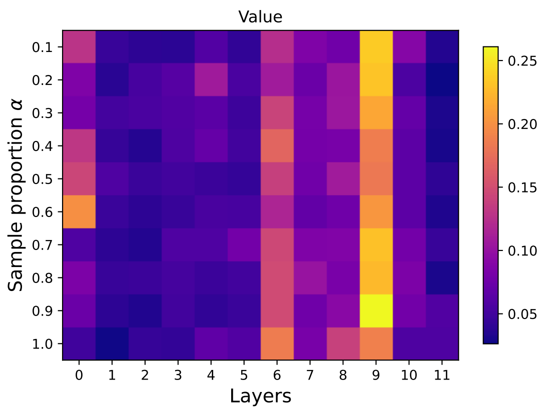


Figure 12: Importance distribution of LoRA for value in RTE under different sample proportions. Each point on the heatmap represents the importance $I_i$ of the query value in layer $i$ under $\alpha$ sample proportion.