ALoRA: Allocating Low-Rank Adaptation for Fine-tuning Large Language Models

Anonymous ACL submission

Abstract

Parameter-efficient fine-tuning (PEFT) is widely studied for its effectiveness and efficiency in the era of large language models. Low-rank adaptation (LoRA) has demonstrated commendable performance as a popular and representative method. However, it is implemented with a fixed intrinsic rank that might not be the ideal setting for the downstream tasks. Recognizing the need for more flexible downstream task adaptation, we extend the methodology of LoRA to an innovative approach we call allocating low-rank adaptation (ALoRA) that enables dynamic adjustments to the intrinsic rank during the adaptation process. First, we propose a novel method, AB-LoRA, that can effectively estimate the importance score of each LoRA rank. Second, guided by AB-LoRA, we gradually prune abundant and negatively impacting LoRA ranks and allocate the pruned LoRA budgets to important Transformer modules needing higher ranks. We have conducted experiments on various tasks, and the experimental results demonstrate that our ALoRA method can outperform the recent baselines with comparable tunable parameters.[1]

1 Introduction

Large language models (LLMs) have been emerging and achieving state-of-the-art (SOTA) results not only on a variety of natural language processing tasks (Qin et al., 2023; Zhu et al., 2023), but also many challenging evaluation tasks (Huang et al., 2023; Li et al., 2023) like question answering in special domains, reasoning, math, safety, instruction following. Despite LLMs becoming general task solvers, fine-tuning still plays a vital role in efficient LLM inference and controlling the style of the LLMs' generated contents.[2] Fine-tuning such large models by full parameters is prohibitive since it requires a large amount of GPU memory and computations. Thus, parameter-efficient fine-tuning (PEFT) (Zhang et al., 2023c; Zhao et al., 2023) has raised much attention in the research field since in PEFT, the tunable parameters are often less than 1% of the LLMs and the computation costs will be significantly decreased.

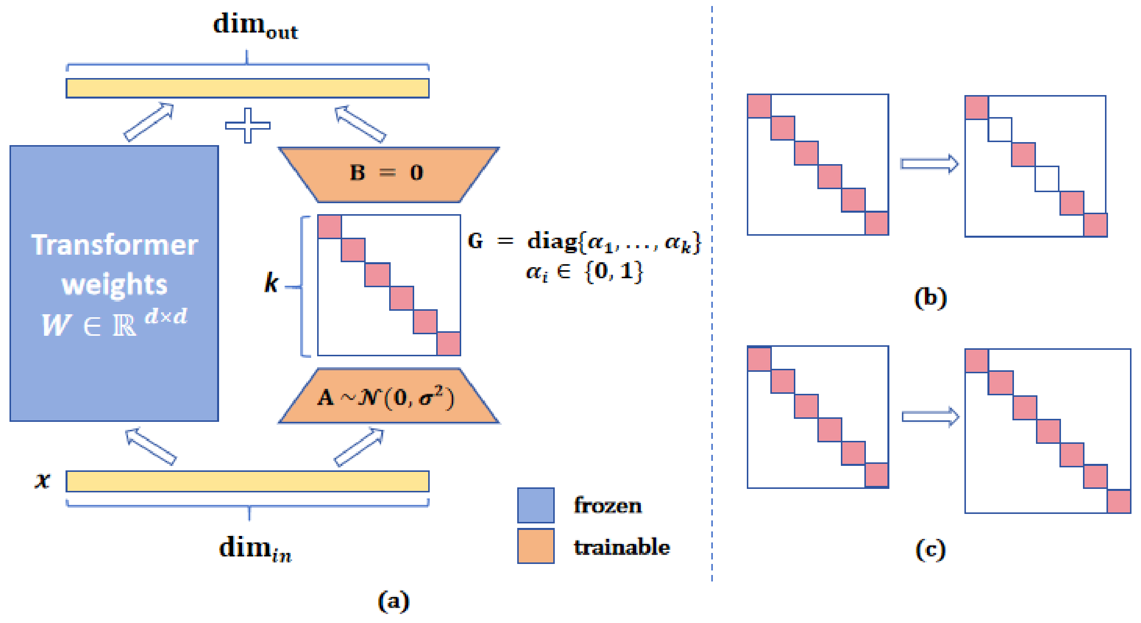
Many PEFT methods have been validated to be effective across various models and tasks, often yielding comparable results with full-parameter fine-tuning (He et al., 2021; Zhu and Tan, 2023; Zhang et al., 2023c). Among these PEFT methods, the reparameterization-based method low-rank adaptation (LoRA) (Hu et al., 2021) is considered one of the most efficient and effective methods at present. LoRA is especially popular after open-sourced LLMs become ubiquitous (Dettmers et al., 2023). LoRA assumes that the change of the model's parameters for adaptation is intrinsically low-dimensional and performs adaptation by optimizing the matrix obtained from low-rank decomposition. Since it is in the form of weight matrix reparameterization, LoRA parameters can be merged with the original LLMs and cause no forward propagation latency.

Although LoRA is effective and can bring stable performance with the original setting in Hu et al. (2021), how to fully exploit its potential for downstream tasks still needs to be determined. First, how to determine the intrinsic rank for each model weight in the Transformer block is still unclear. Moreover, is it reasonable to set the same LoRA rank number for adapting the query, key, and value matrix? Second, in practice, the optimal LoRA rank setting would vary according to multiple factors, such as the backbone model and the task.

In order to improve the performance of downstream task adaptation of LoRA, we now propose the Allocating LoRA (ALoRA) framework (depicted in Figure 1). First, LoRA modules with

[1]Codes and fine-tuned models will be open-sourced to facilitate future research.

[2]Recently, OpenAI also released the fine-tuning API for GPT-3.5-turbo. See blog post: `https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates`.



dim$_{out}$
B = 0
Transformer weights
$W \in \mathbb{R}^{d \times d}$
k
G = diag{$\alpha_1, \ldots, \alpha_k$}
$\alpha_i \in \{0, 1\}$
A ~$\mathcal{N}(0, \sigma^2)$
x
dim$_{in}$
frozen
trainable
(a)
(b)
(c)

Figure 1: Schematic illustration of our ALoRA. Left (a): ALoRA follows LoRA to update the weight matrix W by fine-tuning the low-rank matrices A and B with intermediate rank k. Matrix G is a diagonal matrix where each diagonal element is the gate unit α_i for each LoRA rank $i < k$. Each α_i is set to 1 at initialization. Right upper (b): Some abundant LoRA ranks are pruned by setting the corresponding gate α_i to zeros. Right lower (c): For weight matrix W whose LoRA ranks are not pruned, we will assign additional LoRA ranks to enhance reparameterization.

equal rank size are initialized at each Transformer weight, with all rank gates set to one. During fine-tuning, we re-allocate the LoRA ranks by (a) identifying which LoRA ranks are abundant or have negative contributions and prune those ranks by setting the rank gates to 0; (b) adding the pruned rank budgets to model weights that receive no pruning, that is, important model weights will be assigned more LoRA ranks. In order to calculate the contribution score of each LoRA rank efficiently and accurately, we propose a novel method, AB-LoRA. Our working procedure does not require re-training and does not require higher LoRA rank budgets at initialization or during training.

We conduct extensive experiments on a wide collection of tasks, including sentiment classification, natural language inference, question answering, natural language generation under constraint, and instruction tuning, to demonstrate the effectiveness of our method. Notably, our method can consistently outperform strong PEFT baselines with comparable tunable parameter budgets, especially the recent LoRA variants. We also conducted an analysis showing that (a) our AB-LoRA method indeed can reflect the contribution of each LoRA rank; (b) our method can work with different LoRA rank budgets and different backbone models.

Our contributions are summarized as follows:

- we propose a novel method, AB-LoRA, to estimate the importance of each LoRA rank.
- Built upon AB-LoRA, we propose our ALoRA framework, which can allocate LoRA ranks across different model weights and enhance the adaptation process.
- We have conducted extensive experiments and analysis showing that our ALoRA framework is practical and outperforms the baselines under comparable parameter budgets.

2 Related works

2.1 Parameter-efficient fine-tuning (PEFT) methods

Parameter-efficient fine-tuning (PEFT) is an approach of optimizing a small portion of parameters when fine-tuning a large pretrained backbone model and keeping the backbone model untouched for adaptation (Ding et al., 2022; Zhang et al., 2023c). A branch of PEFT methods inserts additional neural modules or parameters into the backbone model. Representative works in this direction are Adapter (Houlsby et al., 2019; Rücklé et al., 2020; Zhang et al., 2023c), Prefix tuning (Li and Liang, 2021), Prompt tuning (Lester et al., 2021), P-tuning V2 (Liu et al., 2022b). Another approach is to specify the particular parameters to be tunable or prunable (Ben-Zaken et al., 2021; Guo et al., 2021; Zhao et al., 2020). The reparameterization-based methods have attracted much attention (Hu et al., 2021). This branch of approaches transforms the adaptive parameters during optimization into low-rank and parameter-efficient forms. This type of PEFT method is closely related to intrinsic dimension (Aghajanyan et al., 2021; Li et al., 2018), that is, full parameter fine-tuning process of pre-trained models can be reparameterized into optimization within a low-dimensional subspace, i.e., fine-tuning has a low intrinsic dimension (Hu et al., 2021). Intuitively, a well pretrained model does not need to be altered significantly for downstream task adaptation. Qin et al. (2021) investigate whether we can find a common intrinsic subspace shared by various NLP tasks. LoRA (Hu et al., 2021) is inspired by (Aghajanyan et al., 2021; Li et al., 2018) and hypothesizes that the change of weights during model tuning has a low intrinsic rank and optimizes the low-rank decomposition for the change of original weight matrices. PEFT methods are widely applied, especially with the popularization of open-sourced large language models (Zhao et al., 2023) and instruction tuning with these models for different application scenarios (Taori et al., 2023; Dettmers et al., 2023).

2.2 The LoRA method and its variants

LoRA (Hu et al., 2021) is proven to be effective and yield stable results when applied to both relatively small pretrained backbones and large language models (Dettmers et al., 2023; Zhu et al., 2023). Despite its tractability and effectiveness, LoRA still has room for improvements in selecting optimal rank r_m for each Transformer model weight m. The rank r takes discrete values; thus, changing it will directly alter the model structures. The optimal choices of ranks can vary across backbone models, tasks, and even Transformer model weights. Setting a large rank value for r_m can waste training time and computation resources, while progressively setting a small r_m may degrade the model performance. These limitations highlight the importance of upgrading LoRA with an adaptive strategy.

There are already a few works investigating this direction. AdaLoRA (Zhang et al., 2023b) expresses the low-rank multiplication of LoRA in the form of singular value decomposition (SVD), and it identifies the most important ranks by a sensitivity-based importance score. SoRA (Ding et al., 2023) prunes abundant LoRA ranks by imposing a l_0 norm and optimizing with proximal gradient descent. SaLoRA (Hu et al., 2023) prunes the LoRA ranks via the Lagrange multiplier method. Despite these recent efforts, we believe issues still need to be investigated for LoRA rank allocation: (a) The current works initialize a larger value for each r_m and use certain heuristics to prune the number of ranks to meet a predefined budget. This training process inevitably requires additional GPU memory consumption. In addition, the maximum LoRA rank size for each model weight is limited, which restricts the solution space for LoRA rank allocations. (b) The current works depend on heuristic importance scores, which may not reliably reflect the contribution of each LoRA rank. Our work complements the existing literature by addressing the above issues.

3 Methods

3.1 Preliminaries

Transformer model Currently, most widely used open-sourced language models and large language models adopt the stacked Transformer architecture (Vaswani et al., 2017). Each Transformer block is primarily constructed using two key sub-modules: a multi-head self-attention (MHA) layer and a fully connected feed-forward (FFN) layer. The MHA is given as follows:

$$x^{'} = MHA(xW^{Q}, xW^{k}, xW^{V})W^{O}, \quad (1)$$

where $MHA()$ denotes the multi-head attention operation, $x \in \mathbf{R}^{l \times d}$ is the input tensor, $W^O \in \mathbf{R}^{d \times d}$ is the output projection layer (denoted as the Output module), and $W^Q, W^K, W^V \in \mathbf{R}^{d \times d}$ (denoted as the Query, Key, and Value modules). l is the sequence length, d is the hidden dimension. The FFN module consists of linear transformations and an activation function g such as ReLU or GELU (Hendrycks and Gimpel, 2016). Take the FFN module in the LlaMA-2 models (Touvron et al., 2023) as example:

$$x^{'} = (g(xW^{G}) * xW^{U})W^{D}, \quad (2)$$

where $W^G, W^U \in \mathbf{R}^{d \times d^{'}}$ (denoted as Gate and Up module) and $W^D \in \mathbf{R}^{d^{'} \times d}$ (denoted as the Down module), and $d^{'}$ is usually larger than d. For notation convenience, we will refer to the number of modules in a Transformer block as N_{mod}. Thus, in LlaMA-2, $N_{mod} = 7$.

Denote the task's training set as $\mathcal{D}_{\text{train}} = (x_m, y_m), m = 1, 2, ..., M$, where M represents the number of samples. In this work, we only consider the case where input x_m and target y_m are both text sequences. And we expect the language modeling head of LLMs to decode y_m during inference. That is, no additional linear prediction heads are considered for predicting categorical or numerical values.

3.2 Formulation

Our objective is to efficiently fine-tune the LLMs for a specific downstream task under a given LoRA parameter budget $R^{target} = \sum_{m=1}^{N_{mod}} r_m^{target}$. The previous literature (Ding et al., 2023; Hu et al., 2023; Zhang et al., 2023b) usually initialize the LoRA modules with a pre-defined large maximum rank r_{max}, consuming extra GPU memories. Different from the previous works, we now initialize each LoRA module with rank $r_m^{init} = R^{target}/N_{mod}$. That is, upon initialization, we have met the LoRA rank budget. Moreover, we will reallocate the LoRA ranks in order to enhance the fine-tuning performance.

In order to adjust the rank allocation of LoRA modules, we now inject gate units $\alpha_i \in \{0, 1\}$ $(i = 1, 2, ..., r_m)$ to each module m with LoRA rank r_m. Imitating the formulation of SVD, the forward propagation of ALoRA is given by:

$$\begin{aligned} z &= x W_m^A G_m W_m^B, \\ G_m &= \text{diag}(\alpha_{m,1}, ..., \alpha_{m,r_m}), \end{aligned} \tag{3}$$

where diag() denotes a diagonal matrix, $W_m^A \in \mathbf{R}^{d \times r_m}$, $W_m^B \in \mathbf{R}^{r_m \times d}$. At initialization, the gate units are all set to 1.

Different from the previous literature (Ding et al., 2023; Hu et al., 2023; Zhang et al., 2023b), we take an alternative approach, that is, consider the problem of LoRA allocation as neural architecture search (White et al., 2023). We consider the gate units α_i as architecture parameters (denoted as the set Θ), the network with all the non-zero gate units as the super-network M, and denote the parameters in the down-projection and up-projection matrices as Ω, then the optimization objective is:

$$\begin{aligned} &\min_{\mathbf{\Theta}} \mathcal{L}(\mathcal{D}_2, \mathbf{\Omega}^*, \mathbf{\Theta}), \\ &s.t. \ \ \mathbf{\Omega}^* = \arg\min_{\mathbf{\Omega}} \mathcal{L}(\mathcal{D}_1, \mathbf{\Omega}, \mathbf{\Theta}), \end{aligned} \tag{4}$$

where $\mathcal{D}_1$ and $\mathcal{D}_2$ consists of a split of the training set D_{train}, $\mathcal{L}()$ is the loss function. This work uses the cross-entropy loss as the loss function. Note that with discrete values of α_i, solving the above optimization problem is challenging due to non-differentiability. Thus, following the work of differentiable neural architecture search (DNAS) (Liu et al., 2019a), α_i is relaxed to a continuous value in between (0, 1) and the equation 3 becomes:

$$\begin{aligned} z &= W_m^B G_m^{'} W_m^A x, \\ G_m^{'} &= \text{diag}(\alpha_{m,1}^{'}, ..., \alpha_{m,r_m}^{'}), \\ \alpha_{m,i}^{'} &= 2 * \text{Sigmoid}(a_{m,i}^{'}), \ a_{m,i}^{'} \in \mathbf{R}, \end{aligned} \tag{5}$$

where $a_{m,i}^{'}$ is initialized with zero value. With this setting, Equation 4 becomes differentable and can be optimized by bi-level optimization (Liu et al., 2019a).

3.3 Our novel AB-LoRA method

Under the DNAS setting, it is natural to consider the architecture weight $\alpha_i^{'}$ as the importance score for LoRA rank i, and one can use these scores to guide the pruning of abundant LoRA ranks. However, as pointed out by the literature (Zhang et al., 2023c; Chen et al., 2019), and as will be demonstrated in the experiment, the architecture weights are not reliable indicators for the final LoRA allocation's performance. This observation motivates us to propose a simple yet effective modification to the DNAS-style architecture search. Instead of relying on the architecture weights' values to keep the best LoRA ranks, we propose directly evaluating the LoRA rank's superiority by its contribution or influence on the super-network's performances. Since our method mimics conducting ablation studies of a certain LoRA rank from the hyper-network, we refer to our method as the ablation-based LoRA (AB-LoRA).

We now introduce the core of our AB-LoRA method: calculating each LoRA rank's importance score, defined as how much it contributes to the performance of the hyper-network. Denote the complete super-network as M. Super-network M is trained till convergence on the training set. We now consider a modified super-network obtained by zeroing out a single LoRA rank r while keeping all other LoRA ranks. This new hyper-network is denoted as $M_{\backslash r}$. We also consider another modified super-network M_r in which only LoRA rank r is kept while all other LoRA ranks are zeroed out. We

Algorithm 1: Workflow of ALoRA

```
Input: A super-network $M$, with $R^{target}$ LoRA ranks uniformly distributed in modules of $M$;
Output: A new allocation of $R^{target}$ LoRA ranks.
Data: Training set $D_{train}$, a batch of validation data $B_{val}$
1 Train hyper-network $M$ on the training set $D_{train}$ for $K_1$ epochs;
2 for $n = 0$; $n < N_A$ do
3     for a single LoRA rank $r_{m,i}$ on $M$ do
4         Calculate the importance score IS($r_{m,i}$) on $B_{val}$;
5     Prune $n_0$ LoRA ranks with lowest importance scores;
6     if there are modules not pruned then
7         Add $n_0$ LoRA ranks to the un-pruned modules;
8     Further train the Super-network $M$ on $D_{train}$ for $K_2$ epochs;
```

evaluate the three versions of hyper-networks on the same batch of validation data B_{val}. Denote the metric score as a function of a model M, $S(M)$, with the validation data fixed. Then, the importance score of LoRA rank r is given by

$$\text{IS}(r) = S(M) - S(M_{\setminus r}) + S(M_r). \quad (6)$$

In the above equation, $S(M)$ can be treated as a constant term. Thus the above equation can be simplified to $\text{CS}(o) = -S(M_{\setminus r}) + S(M_r)$. Intuitively, the LoRA rank that results in a significant performance drop upon zeroing out must play an important role in the super-network. Similarly, the one keeping most of the performance when acting alone contains important task-related knowledge and should be considered important. In the experiments, different from Chen and Hsieh (2020), we set $S()$ as the negative of the cross-entropy (CE) loss since the widely applied metrics like accuracy or F1: (a) may not vary if the super-network only masks out a single operation, and (b) is not suitable for generative language model fine-tuning.

3.4 The complete process of ALoRA

With the guidance of the importance score in Equation 6, we can now formally define the whole working process of our ALoRA framework (Figure 1). Our working flow of allocating the LoRA ranks builds upon the following intuitions: (a) the pruning and allocation of LoRA ranks is conducted gradually to avoid performance degradation. (b) if the LoRA ranks in a Transformer module receive relatively high importance scores and are not pruned, this module is deemed important. It may need more LoRA ranks for adaptation so that the LoRA parameters can better learn the task knowledge.

The framework of ALoRA is centered on our AB-LoRA method, which requires the super-network to be trained for K_1 epochs on the train set. We freeze the architectural parameters and train only the model parameters on the train set. No bi-level optimization is required, thus saving training time costs. Then, for each LoRA rank, we evaluate the importance score on a batch of samples B_{val} from the development set. Then, n_A LoRA ranks with the lowest scores are pruned by zeroing out their corresponding gate units. Moreover, if some Transformer modules do not have pruned LoRA ranks, we allocate the parameter budgets to them to enhance the adaptation further. [3][4] After the pruning and adding operations, we tune the altered super-network for $K_2 > 0$ epochs to recover the lost performance. The above steps are repeated for N_A times. Formally, we summarize the above process in Algorithm 1.

4 Experiments

In this section, we conduct a series of experiments to evaluate our ALoRA method.

4.1 Baselines

We compare our ALoRA framework with the current SOTA PEFT baseline methods.

Adapter-based tuning We consider the following adapter tuning baselines: (1) Houlsby-Adapter

[3]Note that increasing the rank size of a LoRA module from r_m to $r_m^{'}$ for Transformer module m involves concatenating newly initialized parameters for the matrices, so that $W_m^A \in \mathbf{R}^{d\times r_m}$ and $W_m^B \in \mathbf{R}^{r_m\times d}$ becomes $W_m^A \in \mathbf{R}^{d\times r_m^{'}}$ and $W_m^B \in \mathbf{R}^{r_m^{'}\times d}$. And the diagonal matrix G_m is changed from $\text{diag}(\alpha_{m,1}, ..., \alpha_{m,r_m})$ to $\text{diag}(\alpha_{m,1}, ..., \alpha_{m,r_m^{'}})$. The newly added gate units are initialized with ones.

[4]If n_A is not divided by the number of un-pruned modules, we allocate the n_A ranks as uniformly as possible, with priority given to modules with higher average importance scores. For example, if $n_A = 8$, and module m_1, m_2, m_3 are not pruned, and m_1 has the highest average importance score, m_2 ranks the second, m_3 receives the lowest average importance score. Then three ranks are given to m_1 and m_2, and two ranks are given to m_3.

Method	Additional Params		SST-2	RTE	QNLI	BoolQ	COPA	ReCoRD	Squad
	Initial	Final	(acc)	(acc)	(acc)	(acc)	(acc)	(f1-em)	(f1-em)
					Baselines				
P-tuning v2	20.9M	20.9M	93.4	79.6	92.6	84.7	90.3	89.9	87.6
SPT	16.8M	16.8M	93.6	80.3	92.8	85.3	90.6	90.2	88.1
Housbly-Adapter	21.0M	21.0M	93.5	81.3	92.9	85.2	91.0	90.4	88.0
Parallel-Adapters	21.0M	21.0M	93.6	81.2	93.0	85.7	90.8	90.6	88.2
AdapterDrop	21.0M	21.0M	93.2	80.7	92.8	85.1	90.6	90.3	87.9
LST	21.1M	21.1M	93.4	81.6	93.0	86.2	91.0	90.4	87.9
Learned-Adapter	21.2M	21.2M	94.1	82.1	93.1	87.0	91.1	90.7	88.3
LoRA	20.0M	20.0M	94.1	83.3	93.1	87.3	91.3	90.8	88.4
AdaLoRA	40.0M	20.0M	94.1	83.5	93.2	87.1	91.6	91.1	88.3
SoRA	40.0M	20.0M	94.2	83.7	93.3	87.6	91.7	91.0	88.5
SaLoRA	40.0M	20.0M	93.9	83.4	93.2	87.2	91.5	90.9	88.4
SSP	40.0M	20.0M	94.1	83.1	93.1	87.1	91.6	90.6	88.2
					Our proposed methods				
ALoRA	20.0M	19.6M	**95.0**	**84.6**	**93.7**	**88.0**	**92.1**	**91.8**	**89.2**

Table 1: The Overall comparison of the three GLUE tasks and four question-answering tasks. The backbone model is LlaMA-2 7B. We report the median performance over five random seeds. Bold and Underline indicate the best and the second-best results. The metric for each task is explained in Appendix B.5.

(Houlsby et al., 2019); (2) Parallel-Adapter proposed by He et al. (2021); (3) AdapterDrop (Rücklé et al., 2020); (4) LST (Sung et al., 2022); (5) Learned-Adapter (Zhang et al., 2023c).

Prompt-based tuning For prompt-based tuning methods, we compare with (a) P-tuning v2 (Liu et al., 2021); (b) SPT (Zhu and Tan, 2023).

LoRA and its variants we consider the following LoRA variants as baselines: (a) LoRA (Hu et al., 2021); (b) AdaLoRA (Zhang et al., 2023b). (c) SoRA (Ding et al., 2023); (d) SaLoRA (Hu et al., 2023).

Other PEFT methods We also compare: (1) SSP (Hu et al., 2022), which combines different PEFT methods.

The baselines are implemented using their open-sourced codes. The hyper-parameter settings for the baselines are detailed in Appendix E.

4.2 Datasets and evaluation metrics

We compare our approach to the baselines on (a) four benchmark question-answering tasks: SQUAD (Rajpurkar et al., 2016) and three tasks from the SuperGLUE benchmark(Wang et al., 2019) (BoolQ, COPA and ReCoRD). (b) three sentence level tasks from GLUE benchmark (Wang et al., 2018), SST-2, RTE, QNLI. (d) Alpaca dataset (Taori et al., 2023) for instruction tuning, and MT-Bench (Zheng et al., 2023), to evaluate the instruction tuning quality of LLMs. The dataset introductions, statistics, and prompt-response templates for the above tasks are detailed in Appendix B. The above tasks' evaluation metrics or protocols are in Appendix B.5.

4.3 Experiment Settings

Computing infrastures We run all our experiments on NVIDIA A40 (48GB) GPUs.

Pretrained backbones The main experiments uses most recent open-sourced LLM, LlaMA-2 7B released by Meta (Touvron et al., 2023) as the pretrained backbone model. In the ablation studies, we will also use GPT2-large model (Radford et al., 2019), and RoBERTa-large (Liu et al., 2019b).

Prediction heads When fine-tuning LlaMA-2 7B, we only consider the supervised fine-tuning (SFT) setting (Ouyang et al., 2022), that is, all the predictions are generated using the language modeling head (LM head) upon receiving a prompt or an instruction. For decoding during inference, we use beam search with beam size 5.

Hyper-parameters for ALoRA In our experiments, unless otherwise specified, we set R^{target} to $8 * N_{mod}$, and initially all Transformer model weights are paired with LoRA modules with rank $r_m^{init} = 8$. In this setting, ALoRA satisfies the LoRA rank budget upon initialization, and thus during training and inference.[5] We set n_A to $1 * N_{mod}$. For training with the ALoRA's workflow, we set the batch size of B_{val} to 32, K_1 to 1 epoch, K_2 to 0.25 epoch, and the LoRA rank allocation procedure is conducted for at most $N_A = 8$ times.

Reproducibility We run each task under five

[5]Note that it is possible that the total LoRA ranks after training is smaller than that at initialization.

Method	BLEU	ROUGE-L	METEOR
Learned-Adapter	68.9	70.9	45.8
LoRA	68.9	<u>71.2</u>	46.1
SoRA	<u>70.0</u>	71.1	<u>46.3</u>
ALoRA	**70.6**	**71.8**	**47.1**

Table 2: Results for different PEFT methods on the E2E benchmark. The backbone LM is LlaMA-2 7B. The metrics are explained in Appendix B.5.

Method	Avg GPT-4 score (↑)	ROUGE-L (↑)
SoRA	7.16	53.2
ALoRA	7.47	54.3

Table 3: The performance of instruction tuning using the SoRA and ALoRA methods. The backbone model is LlaMA-2 7B. ↑ means the metric is higher the better.

different random seeds and report the median performance on the test set of each task.

Due to limited length, other experimental settings for the baseline methods and the training procedure are put in Appendix E.

4.4 Main results

The experimental results on the three classification tasks and 4 question answering tasks are presented in Table 1. In the second and third columns of Table 1, we present the initial number of tunable parameters and the final ones. Table 1 reveals that our ALoRA method outperforms the baseline methods across all seven tasks, with comparable or fewer tunable parameters throughout the training and inference processes. In particular, ALoRA successfully outperforms AdaLoRA, SoRA, and SaLoRA with comparable initial and final LoRA parameters. These results demonstrate that our method can better allocate LoRA parameters for better downstream task adaptation.

For the E2E benchmark (Novikova et al., 2017), the results are reported in Table 2. The results show that on the E2E task, our ALoRA method successfully outperforms LoRA and SoRA regarding BLEU, ROUGE-L, or METEOR scores.

After the LlaMA-2 7B is fine-tuned on the Alpaca dataset with our ALoRA and SoRA methods, we utilize the 80 instructions in the MT-Bench as the test set. We follow the current standard practice of utilizing GPT-4 as an unbiased reviewer (Zheng et al., 2023). The protocol of utilizing GPT-4 as the reviewer and scorer is specified in Appendix B.5. The average score provided by GPT-4 is presented in Table 3, along with the ROUGE-L scores

Method	Memory cost (GB)	Speed (it/s)	Time cost (h)
LoRA	17.6	5.01	2.68
SoRA	18.8	4.96	3.63
ALoRA	18.1	5.01	3.81

Table 4: The memory, speed and time cost for fine-tuning LlaMA-2 7B on the E2E task with different PEFT methods.

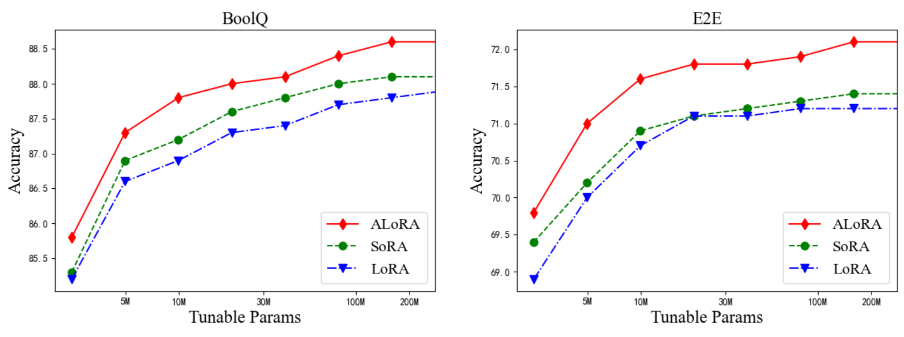
calculated by considering the GPT-4's answers as ground truth. Consistent with the previous experiments (Table 1 and 2), our ALoRA method outperforms the SoRA method in terms of the GPT-4 evaluation scores and ROUGE-L, demonstrating that ALoRA can enhance the instruction tuning quality of large language models. A case study of answers generated by different methods is presented in Table 9, showcasing that ALoRA leads to better instruction-tuned LLMs.

4.5 Ablation studies and analysis

Analysis of Training Efficiency So far, we have demonstrated that our ALoRA can outperform LoRA and SoRA on a wide collection of tasks. One might suspect this advantage is achieved with significant time or memory costs. We compare the max training GPU memory, training speed, and training time costs of ALoRA, SoRA, and LoRA when fine-tuning LlaMA-2 7B with the E2E benchmark. From Table 4, one can see that ALoRA requires less memory costs during training than SoRA since it does not initialize with a larger LoRA rank. Moreover, under early stopping, the total training time cost of ALoRA remains comparable with SoRA and LoRA.

Ablation study of ALoRA framework We now consider the following variants of ALoRA: (a) instead of utilizing our novel AB-LoRA method, we follow the optimization procedure of Equation 4, and use the architectural weights $\alpha^{'}_{m,i}$ as the importance scores. This variant is denoted as ALoRA-DNAS. (b) Use the sensitivity-based metric in Zhang et al. (2023b) as the importance measurement. (denoted as ALoRA-Sensi). The experimental results on the BoolQ, ReCoRD, and SQUAD tasks are reported in Table 6 of Appendix F. The results show that ALoRA outperforms the two variants, demonstrating that our AB-LoRA method can provide better guidance in allocating LoRA ranks.

Visualization of the final rank allocations In this section, we visualize the final rank allocations of ALoRA after the training process on the E2E task in Figure 3. We also compare the LoRA rank



(a) BoolQ (b) E2E

Figure 2: Performances under different LoRA rank budgets. The x-axis represents the number of tunable parameters, and the y-axis represents the performance score.

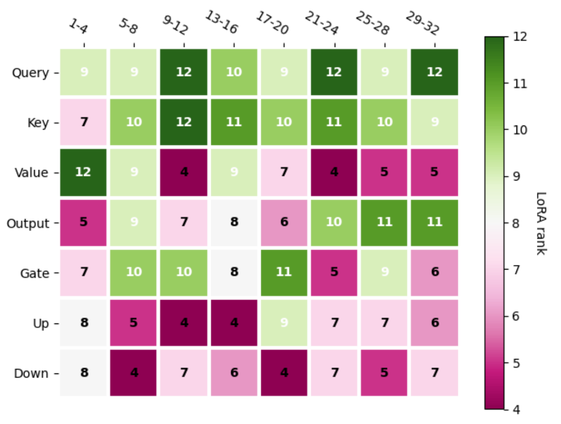


Figure 3: The final rank allocations of ALoRA after fine-tuning the LlaMA-2 7B model on the E2E task.

allocations by the SoRA method in Figure 4 of Appendix H. We can see from Figure 3 that (a) More LoRA rank budgets are put to adapt the query and key modules, while the value and output modules in the self-attention are less emphasized. (b) The feed-forward layer in the Transformer block requires fewer LoRA ranks, indicating that this layer stores general language knowledge, while the attention module will contain more task-specific knowledge after ALoRA fine-tuning. Compared with ALoRA's allocation, SoRA results in a more unbalanced allocation, putting more rank budgets to the Down module than our ALoRA method.

Comparisons under different LoRA rank budgets Note that in the main experiments, we set the targeted LoRA rank budget as $R^{target} = 8 * N_{mod}$. Now we vary this budget to any multiplier in 1, 2, 4, 8, 16, 32, 64, 128 times N_{mod}, and see how ALoRA, SoRA, and LoRA perform on the BoolQ and E2E tasks. The experimental results are presented in Figure 2(a) and 2(b). From the results, we can see that under different LoRA rank budgets, our ALoRA method can consistently outperform LoRA and SoRA by effectively allocating different LoRA ranks properly to different Transformer modules, thus enhancing the performance of fine-tuning.

Ablation on the pretrained backbones Our main experiments are conducted on the LlaMA-2 7B model. To demonstrate the wide applicability of our method, we now conduct experiments on RoBERTa-large and GPT2-large. The results are reported in Table 7 and 8. We can see that on these two backbones, our method can also outperform the baseline methods.

5 Conclusion

This work presents the Allocating Low-Rank Adaptation (ALoRA), an innovative method for parameter-efficient fine-tuning large language models. Upon the hypothesis that the adaptation for different Transformer modules could be of different tanks, we introduce a novel workflow for allocating LoRA ranks in the fine-tuning process. First, we propose a novel method, AB-DNAS, to accurately evaluate the importance scores of LoRA ranks. Second, guided by the AB-DNAS method, our workflow allows the pruning of ranks at specific modules and considers allocating more ranks to essential modules. Thus, our method does not require to set a more significant initial rank. Our method is convenient to implement and off-the-shelf. Experiments on various tasks demonstrate that our ALoRA method outperforms the baseline methods.

Limitations

We showed that our proposed method can greatly improve the performance of parameter-efficient tuning on diverse tasks and different pretrained models (i.e., LlaMA-2 7B, RoBERTa-large and GPT2-large). However, we acknowledge the following limitations: (a) the more super-sized open-sourced LLMs, such as LlaMA-2 13B and 70B, are not experimented due to limited computation resources. (b) Other tasks in natural language processing, like information extraction, were also not considered. But our framework can be easily transferred to other backbone architectures and different types of tasks. It would be of interest to investigate if the superiority of our method holds for other large-scaled backbone models and other types of tasks. And we will explore it in future work.

Ethics Statement

The finding and proposed method aims to improve the low-rank adaptation (LoRA) based tuning in terms of better rank allocations and performances. The used datasets are widely used in previous work and, to our knowledge, do not have any attached privacy or ethical issues. In this work, we have experimented with LlaMA-2 7B, a modern large language model. As with all LLMs, LlaMA-2's potential outputs cannot be predicted in advance, and the model may in some instances produce inaccurate, biased or other objectionable responses to user prompts. However, this work's intent is to conduct research on different fine-tuning methods for LLMs, not building applications to general users. In the future, we would like to conduct further testing to see how our method affects the safety aspects of LLMs.

References

Armen Aghajanyan, Sonal Gupta, and Luke 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, Online. Association for Computational Linguistics.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv*, abs/2106.10199.

Xiangning Chen and Cho-Jui Hsieh. 2020. Stabilizing differentiable architecture search via perturbation-based regularization. In *International Conference on Machine Learning*.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2019. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129:638 – 655.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Fine-tuning of Quantized LLMs. *arXiv e-prints*, page arXiv:2305.14314.

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 *Conference on Empirical Methods in Natural Language Processing*.

Ning Ding, Yujia Qin, Guang Yang, Fu Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Haitao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juan Li, and Maosong Sun. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *ArXiv*, abs/2203.06904.

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. 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 4884–4896, Online. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *ArXiv*, abs/2110.04366.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv: Learning*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Shengding Hu, Zhen Zhang, Ning Ding, Yadao Wang, Yasheng Wang, Zhiyuan Liu, and Maosong Sun. 2022. Sparse structure search for parameter-efficient tuning. *ArXiv*, abs/2206.07382.

Yahao Hu, Yifei Xie, Tianfeng Wang, Man Chen, and Zhisong Pan. 2023. Structure-aware low-rank adaptation for parameter-efficient fine-tuning. *Mathematics*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, et al. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*.

Shibo Jie and Zhifang Deng. 2022. Convolutional bypasses are better vision transformer adapters. *ArXiv*, abs/2207.07039.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2023. Vera: Vector-based random matrix adaptation. *ArXiv*, abs/2310.11454.

Tuan Le, Marco Bertolini, Frank No'e, and Djork-Arné Clevert. 2021. Parameterized hypercomplex graph neural networks for graph classification. In *International Conference on Artificial Neural Networks*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv e-prints*, page arXiv:1804.08838.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. 2023. Cmmlu: Measuring massive multitask language understanding in chinese. *arXiv preprint arXiv:2306.09212*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055.

Xiangyang Liu, Tianxiang Sun, Xuanjing Huang, and Xipeng Qiu. 2022a. Late prompt tuning: A late prompt could be better than many prompts. *ArXiv*, abs/2210.11292.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *ArXiv*, abs/2110.07602.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Annual Meeting of the Association for Computational Linguistics*.

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

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2021. Cross-task generalization via natural language crowdsourcing instructions. In *Annual Meeting of the Association for Computational Linguistics*.

Nafise Sadat Moosavi, Quentin Delfosse, Kristian Kersting, and Iryna Gurevych. 2022. Adaptable adapters. In *North American Chapter of the Association for Computational Linguistics*.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 Technical Report. *arXiv e-prints*, page arXiv:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is chatgpt a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476*.

Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Zhiyuan Liu, Juan-Zi Li, Lei Hou, Peng Li, Maosong Sun, and Jie Zhou. 2021. Exploring low-dimensional intrinsic task subspace via prompt tuning. *ArXiv*, abs/2110.07867.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. In *Conference on Empirical Methods in Natural Language Processing*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan D. Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng-Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. 2021. Multitask prompted training enables zero-shot task generalization. *ArXiv*, abs/2110.08207.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *ArXiv*, abs/2206.06522.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537.

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 *BlackboxNLP@EMNLP*.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2021. Finetuned language models are zero-shot learners. *ArXiv*, abs/2109.01652.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. 2023. Neural Architecture Search: Insights from 1000 Papers. *arXiv e-prints*, page arXiv:2301.08727.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020a. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020b. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, V. G. Vinod Vydiswaran, and Hao Ma. 2022. Idpg: An instance-dependent prompt generation method. In *North American Chapter of the Association for Computational Linguistics*.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023a. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *ArXiv*, abs/2308.03303.

Qingru Zhang, Minshuo Chen, Alexander W. Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and

Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. *ArXiv*, abs/2303.10512.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *ArXiv*, abs/2006.05987.

Yuming Zhang, Peng Wang, Ming Tan, and Wei-Guo Zhu. 2023c. Learned adapters are better than manually designed adapters. In *Annual Meeting of the Association for Computational Linguistics*.

Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2226–2241, Online. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. *arXiv e-prints*, page arXiv:2303.18223.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *arXiv e-prints*, page arXiv:2306.05685.

Wei Zhu and Ming Tan. 2023. SPT: Learning to selectively insert prompts for better prompt tuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11862–11878, Singapore. Association for Computational Linguistics.

Wei Zhu, Xiaoling Wang, Huanran Zheng, Mosha Chen, and Buzhou Tang. 2023. PromptCBLUE: A Chinese Prompt Tuning Benchmark for the Medical Domain. *arXiv e-prints*, page arXiv:2310.14151.

A Additional related works

Adapter-based tuning. One of the most important research lines of PEFT is adapter-based tuning. Adapter (Houlsby et al., 2019) inserts adapter modules with bottleneck architecture between every consecutive Transformer (Vaswani et al., 2017) sublayers. AdapterFusion (Pfeiffer et al., 2021) only inserts sequential adapters after the feed-forward module. Adapter-based tuning methods have comparable results with model tuning when only tuning a fraction of the backbone model's parameter number. Due to their strong performance, a branch of literature has investigated the architecture of adapters in search of further improvements. He et al. (2021) analyze a wide range of PETuning methods and show that they are essentially equivalent. They also propose the general architecture of PEFT, and derive the Parallel Adapter which connects the adapter modules in parallel to the self-attention and MLP modules in the Transformer block. AdapterDrop (Rücklé et al., 2020) investigates the efficiency of removing adapters from lower layers. Adaptive adapters (Moosavi et al., 2022) investigate the activation functions of adapters and propose to learn the activation functions of adapters via optimizing the parameters of rational functions as a part of the model parameters. Compacter (Mahabadi et al., 2021) uses low-rank parameterized hypercomplex multiplication (Le et al., 2021) to compress adapters' tunable parameters. LST (Sung et al., 2022) improves the memory efficiency by forming the adapters as a ladder along stacked Transformer blocks, and it enhances the adapter module by adding a self-attention module to its bottleneck architecture. (Sung et al., 2022; Jie and Deng, 2022) try to add different encoding operations, like self-attention operations and convolutions between the bottleneck structure of adapters, and achieve better performances. Learned-Adapter (Zhang et al., 2023c) builds upon the above adapter-based methods and enhance the performance of adapter tuning by automatically learning better architectures for adapters.

Prompt tuning methods Prompt tuning (Lester et al., 2021) and P-tuning (Liu et al., 2022b) insert a soft prompt to word embeddings only, and can achieve competitive results when applied to supersized PTMs. Prefix-tuning (Li and Liang, 2021) and P-tuning v2 (Liu et al., 2021) insert prompts to every hidden layer of PTM. IDPG (Wu et al., 2022) uses the prompt generator with parameterized hypercomplex multiplication (Le et al., 2021) to generate a soft prompt for every instance. LPT (Liu et al., 2022a) improves upon IDPG by selecting an intermediate layer to start inserting prompts. SPT (Zhu and Tan, 2023) designs a mechanism to automatically decide which layers to insert new instance-aware soft prompts.

Literature LoRA methods Since LoRA is the most popular PEFT method in the era of large language models, there are many works that are orthogonal to AdaLoRA, SoRA and our work that are devoted to improve LoRA on many different

Datasets	#train	#dev	#test	$\|\mathcal{Y}\|$	Type	Labels	Metrics
SuperGLUE tasks							
BoolQ	9.4k	1.6k	1.6k	2	Question Answering	True, False	acc
COPA	0.4k	0.05k	0.05k	2	Question Answering	choice1, choice2	acc
ReCoRD	101k	1k	7.4k	-	Question Answering	-	f1-em
GLUE tasks							
SST-2	66k	1k	0.8k	2	sentiment classification	positive, negative	acc
RTE	2.5k	0.1k	0.1k	2	NLI	entailment, not entailment	acc
QNLI	104k	1k	5.4k	2	NLI	entailment, not entailment	acc
Other tasks							
Squad	87k	1k	5.9k	-	Question Answering	-	f1-em
E2E	42k	4.6k	4.6k	-	NLG	-	BLEU/ROUGE-L/METEOR
Alpaca	52k	-	-	-	Instruction tuning	-	-
MT-Bench	-	-	80	-	Instruction tuning	-	GPT-4 scores

Table 5: The dataset statistics of the GLUE and SuperGLUE benchmark tasks evaluated in this work. $|\mathcal{Y}|$ is the number of classes for a classification task.

aspects. QLoRA (Dettmers et al., 2023) proposes a novel quantization method that can significantly reduce the memory consumptions of LLMs during LoRA fine-tuning. LoRA-FA (Zhang et al., 2023a) freezes parts of the randomly initialized LoRA matrices. (d) VERA (Kopiczko et al., 2023) investigate whether one could froze the randomly initialized LoRA matrices and only learns a set of scaling vectors. Tying LoRA matrices across layers are also investigated by VERA.

B Appendix for the datsets and evaluation metrics

B.1 Datasets from GLUE and SuperGLUE

We experiment on three tasks from the GLUE (Wang et al., 2018) benchmark: (a) (a) a sentiment classification task, SST-2. (b) two benchmark natural language inference tasks, RTE and QNLI. We also experiment with three question-answering tasks: (a) two question answering tasks in the format of binary choices, COPA and BoolQ. (b) A Squad (Rajpurkar et al., 2016) style question answering task, ReCoRD.

Since the original test sets are not publicly available for these tasks, we follow Zhang et al. (2020); Mahabadi et al. (2021) to construct the train/dev/test splits as follows to ensure a fiar comparison: (a) for datasets with fewer than 10k samples (RTE, COPA, BoolQ), we divide the original validation set in half, using one half for validation and the other for testing. (b) for larger datasets, we split 1k samples from the training set as the development set, and use the original development set as the test set. The detailed statistics of the GLUE and SuperGLUE benchmark tasks is presented in Table 5.

B.2 The Squad task

Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. This task is one of the most widely studied question answering task in the field.

In this work, we use the v1.1 version of SQUAD. Since the original test sets are not publicly available for these tasks, we follow Zhang et al. (2020); Mahabadi et al. (2021) and split 1k samples from the training set as the development set, and use the original development set as the test set. The detailed statistics of this task is presented in Table 5.

B.3 Datasets: E2E benchmark

The E2E benchmark dataset for training end-to-end, data-driven natural language generation systems in the restaurant domain. It asks a model to generate natural utterances based on a set of given key contents. This dataset has a 42061/4672/4693 train/dev/test split.

B.4 Dataset: Instruction tuning

Instruction tuning is an important method to improve the general capabilities of large language models (Ouyang et al., 2022). With the rise of large language models in the scale of 10B parameters or more, like GPT-3, T5, PaLM, researchers have actively explored the few-shot or zero-shot capabilities of these models. (Mishra et al., 2021) find that fine-tuning these LLMs on a large scale datasets containing hundreds of NLP tasks significantly improves the zero-shot performances on unseen tasks, establishing the scaling law of task numbers. The previous works like (Wei et al., 2021) and T0 (Sanh et al., 2021) establishes the instruction tuning datasets by transforming the traditional NLP tasks into a unified prompt format. InstructGPT (Ouyang et al., 2022) conducts instruction tuning using the dataset constructed based the user queries from the OpenAI API users. Note that this work is also a seminal work for human feedback learning with reinforcement learning. However, the complete instruction tuning dataset from (Ouyang et al., 2022) remains closed. With the launch of ChatGPT, (Taori et al., 2023) (Alpaca) constructs an instruction tuning dataset with diverse topics using the self-instruct techniques.

For our experiment, we employ the Alpaca dataset (Taori et al., 2023) for instruction tuning. Specifically, we employs its cleaned version[6]. This dataset comprises 51K instructions and demonstrations, and is suitable for instruction tuning. The cleaned version corrects multiple issues such as hallucinations, merged instructions, and empty outputs.

B.5 Evaluation metrics/protocols

For the three GLUE tasks we experiment on, we report accuracy (denoted as acc). For ReCoRD, we report the average of the F1 score and the exact match score (denoted as f1-em). For the BoolQ and COPA tasks, we report accuracy. The above choices of evaluation metrics strictly follow (Wang et al., 2018) and (Wang et al., 2019).

For the SQUAD dataset, we also report the average of the F1 score and the exact match score (denoted as f1-em).

Following (Novikova et al., 2017), we report three different metrics on the E2E task: (a) BLEU; (b) ROUGE-L; (c) METEOR. We rely on the HuggingFace Evaluate package[7] for computing these metrics.

For evaluating the quality of instruction tuned LlaMA-2 7B, we follow the current common practice of utilizing GPT-4 as a unbiased reviewer (Zheng et al., 2023). 80 instructions from the MT-Bench is set as a test set. We generate model responses from a fine-tuned model with beam size 5 with the generation function in Huggingface Transformers (Wolf et al., 2020a). Then we compare SoRA and ALoRA's answers with GPT-4. For each instruction in MT-Bench, GPT-4 (OpenAI, 2023) is asked to write a review for both answers from the two methods, and assigns a quantitative score on a scale of 10 to each response. The prompts of instructing GPT-4 for evaluation is presented in Appendix D. ROUGE-L scores computed by considering the answers generated by GPT-4 as the ground truth.

[6]https://huggingface.co/datasets/yahma/alpaca-cleaned.

[7]https://huggingface.co/docs/evaluate/index

C Prompt templates for fine-tuning LlaMA-2 7B

Since we fine-tune LlaMA-2 7B without introducing task-specific prediction heads, we need to transform all the tasks into a prompt-response format. Now we present the prompt-response template for each task.

Templates for RTE and QNLI Since these two tasks are NLI tasks, the samples in them consists of two input text, [sentence1] and [sentence1], and a label [label_name] (entailment or not entailment). Thus, we use the following templates:

Template for prompt:

```
sentence 1: [sentence1]
sentence 2: [sentence1]
Are sentence 1 and sentence 2 have
entailment relation or not?
```

Template for response:

```
[label_name]
```

Templates for SST-2 The samples in this task consists of one input text, [sentence], and a label [label_name] (positive or negative).

Template for prompt:

```
[sentence]
The sentiment of the given sentence is:
```

Template for response:

```
[label_name]
```

Templates for BoolQ The samples in this task consists of a reference document, [doc], a query, [query], and a label [label_name] (yes or no).

Template for prompt:

```
Reference document:
[doc]
Question:
[query]
```

Template for response:

```
[label_name]
```

Templates for COPA The samples in this task consists of a premise, [premise], two choices, [choice1] and [choice2], a query, [query], and a label [label_name] (1 or 2, indicating which choice is consistent with the premise).

Template for prompt:

```
Premise:
[premise]
Choice 1: [choice1]
Choice 2: [choice2]
Question:
[query]
```

Template for response:

```
[label_name]
```

Templates for ReCoRD and SQUAD The samples in these two tasks consist of a context document, [context], a question, [query], and a answering span, [answer].

Template for prompt:

```
Context:
[context]
Question:
[query]
```

Template for response:

```
[answer]
```

Templates for E2E The samples in this task consists of a reference [ref], consisting required information, and a targeted response, [target], which is a customer review written according to the reference's contents.

Template for prompt:

```
Reference:
[ref]
Generate a customer review following the
given reference.
```

Template for response:

```
[target]
```

D Prompt templates for GPT-4 evaluations

In this work, we utilize the powerful LLM GPT-4 (OpenAI, 2023) as the evaluator for comparing the instruction tuning quality. As a reviewer, GPT-4 will receive a query [query], two responses, [response1] and [response2], from two assistants. We will ask GPT-4 to write a review for each response, assessing the quality of the response, and then ask GPT-4 to assign a score on a scale of 10 to each response.

Template for prompt:

```
Task Introduction
you will be given a query, and two responses
from two assistants,
could you compare the two responses,
and do the following:
(1) write a concise review for each
assistant's response, on how well the
response answers the query, and whether
it will be helpful to humans users, and any
issues in the response;
(2) assigns a quantitative score on a scale
of 10 to each response, reflecting
your assessment of the two responses
Query:
[query]
Response 1 from assistant 1:
[response1]
Response 2 from assistant 2:
[response2]
```

E Appendix for Experimental settings

Here, we provide more details for experimental settings.

Hyper-parameters for the baseline PEFT methods For P-tuning V2, the number of prompt tokens at each layer is set to 160. For SPT, the bottleneck dimension is set to 256, and the number of prompt layers is set to 8. For adapter-based methods, the bottleneck dimension is set to 40, and the adapter modules are added on the self-attention and feed-forward module. For LoRA and ALoRA, the initial rank at each module is set to 8. For AdaLoRA, SoRA, and SaLoRA, the initial rank at each module is set to 16, and half of the rank budget is pruned during fine-tuning. We adjust the sparsity for SSP so that the number of tunable parameters is comparable with ALoRA and the other baselines.

Training settings for PEFT methods We use the HugginFace Transformers (Wolf et al., 2020b) and PEFT (Mangrulkar et al., 2022) for implementing all the methods, and for training and making predictions. For fine-tuning LlaMA-2 7B model, the maximum sequence length is set to 2048. The maximum training epoch is set to 10. The batch size is set between 16 for task with less than 10k training set, and 128 otherwise. We use AdamW as the optimizer with a linear learning rate decay

Method	BoolQ (acc)	ReCoRD (f1-em)	Squad (f1-em)
ALoRA-DNAS	87.6	91.2	88.7
ALoRA-Sensi	87.5	91.3	88.6
ALoRA	88.0	91.8	89.2

Table 6: The comparison of ALoRA's variants on the BoolQ, ReCoRD, and Squad tasks. The backbone model is LlaMA-2 7B.

schedule and 6% of the training steps for warm-up. The learning rate is set to 1e-4. The other hyper-parameters are kept the same with (Wolf et al., 2020b). In every 200 steps, the model is evaluated on the dev set. Patience is set to 10, that is, if the model does not achieve a lower development set loss for 10 evaluation runs, the training stops. The best checkpoint on the dev set is used to run predictions on the test set.

F Ablation on the ALoRA framework

We consider two variants of ALoRA: (a) use the architectural weights $\alpha'_{m,i}$ as the importance scores during bi-level optimization (Liu et al., 2019a). This variant is denoted as ALoRA-DNAS. (b) Use the sensitivity-based metric in Zhang et al. (2023b) as the importance measurement. (denoted as ALoRA-Sensi). The experiments on the BoolQ and E2E methods are provided in 6

G Ablation on the pretrained backbones

Our main experiments are conducted on the LlaMA-2 7B model. To demonstrate that our method works well regardless of the backbone models, we now conduct experiments on the RoBERTa-large. In this experiment, since the language modeling capabilities of these RoBERTa-large can not match LlaMA-2 7B model, we change the following setting for the prediction head: (a) we use a linear layer as the prediction head for classification tasks. (b) for the ReCoRD task, we use two linear layers to predict the starting and ending positions of a entity span. The other experimental settings are kept the same with the main experiments (Table 1).

We conduct experiments on the BoolQ, ReCoRD and Squad tasks. The results are reported in Table 7. We can see that on the RoBERTa-large backbone, our method can also outperform the baseline methods.

We also run GPT2-large on the E2E task, and the results are reported in Table 8. The results demonstrate that when the GPT2-large is the backbone model, our ALoRA method also outperforms the baselines.

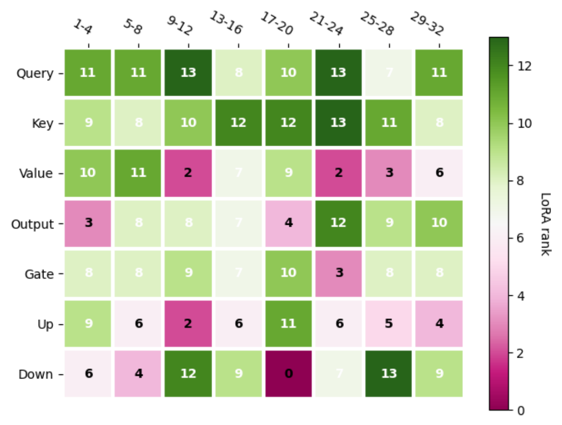


Figure 4: The final rank allocations of SoRA after fine-tuning the LlaMA-2 7B model on the E2E task.

H Visualization of the final rank allocations of SoRA

In the main contents, we visualize the final rank allocations of ALoRA after the training process on the E2E task in Figure 3. As comparison, we now present the LoRA rank allocations by the SoRA method in Figure 4.

I Case studies of Instruction tuning

In the Section 4.4 of the main content, we present the overall performance of ALoRA and SoRA on the MT-Bench, after fine-tuning LlaMA-2 7B on the Alpaca dataset. Now we present concrete examples in Table 9 to showcase the Superiority of ALoRA.

Method	**Additional Params**		**BoolQ**	**ReCoRD**	**Squad**
	Initial	**Final**	**(acc)**	**(f1-em)**	**(f1-em)**
Results for RoBERTa-large					
Learned-Adapter	366M	354M	86.8	90.2	88.7
LoRA	3.54M	3.54M	86.9	90.0	88.6
SoRA	708M	3.53M	87.2	90.1	88.7
ALoRA	3.54M	3.42M	87.6	90.7	89.4

Table 7: The comparison on the BoolQ, ReCoRD, and Squad tasks, when the backbone model is RoBERTa-large. We report the median performance over 5 random seeds. Bold and Underline indicate the best and the second best results. The metric for each task is explained in Appendix B.5.

Method	**BLEU**	**ROUGE-L**	**METEOR**
Learned-Adapter	68.6	69.6	45.2
LoRA	68.7	69.8	45.3
SoRA	68.9	69.9	45.4
ALoRA	**69.3**	**70.4**	**46.0**

Table 8: Results for different PEFT methods on the E2E benchmark. The backbone LM is GPT2-large. The metrics are explained in Appendix B.5.

Query	Edit the following paragraph to correct any grammatical errors: She didn't remembre where is her purse, so I thinks its in the car but he's say it's on kitchen table but he are not sure, and then they asked me to looking for it, she's say, "Can you?", and I responds with, "Maybe, but ain't no sure," and he not heard me, and, "What?", he asks, "Did you found it?".
	ALoRA's response
Responses	She didn't remember where her purse was, so I thought it was in the car, but he said it was on the kitchen table, but he wasn't sure, and then they asked me to look for it, she said, "Can you?" and I responded with, "Maybe, but I'm not sure," and he didn't hear me, and then he asked, "Did you find it?"
GPT-4's comments	Assistant 1's response effectively corrects all the grammatical errors in the original paragraph. The assistant has correctly changed "remembre" to "remember," rearranged the clause order from "where is her purse" to "where her purse was," and adjusted verb forms for proper agreement (e.g., changing "thinks" to "thought"). Quotation marks and commas are properly used, making direct speech clear and punctuated correctly. The conjunctions are better placed, contributing to a more natural flow of sentences. Additionally, all instances of colloquial or incorrect language such as "ain't no sure" have been revised into standard English ("I'm not sure"). Overall, this response would be very helpful to human users as it provides a grammatically sound version of the original text.
GPT-4 score	9
	SoRA's response
Responses	She didn't remember where her purse was, so I thought it was in the car, but he said it was on the kitchen table, but he wasn't sure. They asked me to look for it, and she said, "Can you?" I responded with, "Maybe, but I'm not sure." He didn't hear me, and he asked, "What?" I said, "Did you find it?"
GPT-4's comments	Assistant 2 also does an excellent job correcting grammatical mistakes from the initial query. This assistant similarly fixes issues with verb tense consistency and subject-verb agreement (e.g., "he's say" corrected to "he said"). However, there is a slight deviation at the end where Assistant 2 writes: "I said, 'Did you find it?'". This seems like an attempt by Assistant 2 to make sense of a confusing part of dialogue in which speaker roles were unclear but results in altering the meaning slightly because it suggests that 'I' am asking if 'he' found it rather than him asking 'me'. This could be misleading or confusing without additional context indicating who is speaking at that moment. Nevertheless, punctuation around quotations is accurate which aids readability.
GPT-4 score	8

Table 9: An example showcasing that the fine-tuning quality of ALoRA is higher than SoRA.