# PARADE: Parameter-Efficient Fine-tuning with Prompt Aware Representation ADjustmEnt

**Anonymous ACL submission**


## Abstract

Despite the presence of many competitive PEFT methods like LoRA, we still need a PEFT method that is efficient under the single-backbone multi-tenant setting while performing competitively in the downstream tasks. In this work, we propose a novel PEFT method, Prompt Aware Representation ADjustmEnt (PARADE). First, we propose to install a lightweight vector generator at each Transformer layer to generate vectors that will modify the hidden states in the multi-head self-attention (MHSA) and position-wise feed-forward (FFN) modules and, as a result, modulate the behaviors of the pre-trained backbone. Second, the vector generators are modules with a bottleneck architecture consisting of a pooling operation, two linear projections, and an activation function. To enhance the downstream performance of vector generators, we propose an attention-based capsule network as the pooling operation, which can effectively summarize the semantic information in the input instructions. We have conducted experiments on various tasks, and the experimental results demonstrate that: (a) our PARADE method can outperform the recent baselines with comparable tunable parameters. (b) Our PARADE method is more efficient than LoRA under the single-backbone multi-tenant setting.[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 on many challenging evaluation tasks (Huang et al., 2023; Li et al., 2023) such as question answering in different domains, reasoning, mathematics, safety, and 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. Recently, there has also been a trend to host an LLM as a service (MaaS) (Gan et al., 2023), and different tenants can specialize the LLM with their privately owned PEFT modules (multi-tenant setting) (Chen et al., 2023).

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 widely applied in the era of LLMs. Although LoRA is effective and can bring stable performance with the original setting in Hu et al. (2021), it has a clear drawback: it has to add LoRA modules to multiple weights of the Transformer layer and introduce significant additional latency in every generation step under the multi-tenant setting. Thus, it is of central importance to develop a novel PEFT method that introduces minimum latency during generation and still can perform competitively in downstream tasks.

In this work, we propose a novel PEFT method called Prompt Aware Representation ADjustmEnt (PARADE) (depicted in Figure 1). Our method fine-tuned the LLMs by directly modifying the hidden representations in the model via multiplying

[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`.

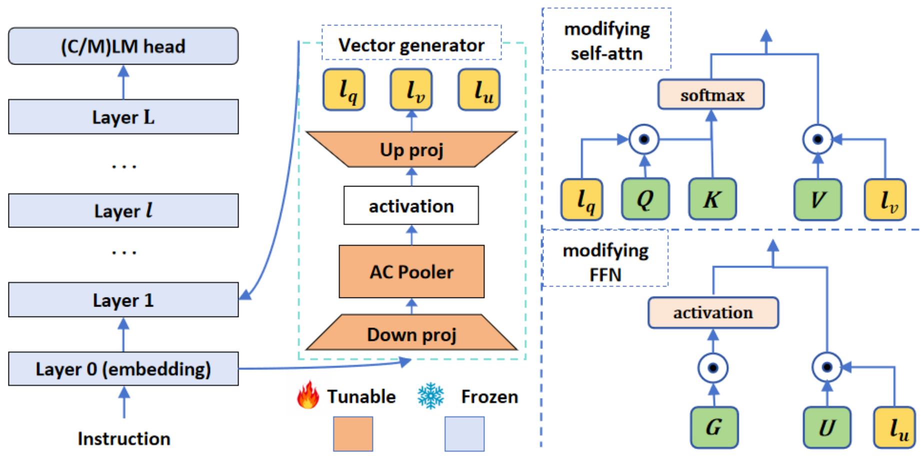



Figure 1: Schematic illustration of our PARADE method. **Left**: The vector generator consists of a down-projection, an attentional capsule-based pooler (AC pooler), an activation function, and an up-projection. The vector generator uses the instructions' hidden states as the input tensors and outputs the adjusting vectors. **Right**: The adjusting vectors multiply the Query (Q) and Value (V) hidden states in the MHSA module and the Up (U) hidden states in the feed-forward module.

them with certain vectors (called adjusting vectors) and thus regulating the attention patterns in the self-attention module and the activated knowledge in the feed-forward module. Unlike the previous literature like (Liu et al., 2022a) or BitFit (Ben-Zaken et al., 2021), the vectors are not randomly initialized and fixed across different input instructions. Instead, we install a vector generator before each Transformer layer. The vector generator takes the hidden states of the input instructions as input and generates the adjusting vectors with a down-projection layer, a novel attentional capsule-based pooling layer, and an up-projection layer.

We conduct extensive experiments on a comprehensive collection of tasks, including sentiment classification, natural language inference, question answering, natural language generation under constraint, math reasoning, SQL generation, and instruction tuning, to demonstrate the effectiveness of our PARADE method. Notably, our method can consistently outperform strong PEFT baselines with comparable tunable parameter budgets, especially the recent LoRA variants and IA3 (Liu et al., 2022a) or BitFit (Ben-Zaken et al., 2021). We also use experiments or analysis to show that: (a) our method has significantly lower latency under the multi-tenant setting than the LoRA-based methods with comparable tunable parameters. (b) our proposed attentional capsule-based pooling module effectively enhances our PARADE method.

Our contributions are summarized as follows:

- we propose a novel method, PARADE, to a novel PEFT method that generates adjusting vectors to hidden representations conditioned on the input instructions.
- We propose a novel attentional capsule-based pooling module for effectively summarizing the information of the input instruction sequences to the generated vectors.
- We have conducted extensive experiments and analysis showing that our PARADE framework is (a) practical and outperforms the baselines under comparable parameter budgets. (b) efficient during inference for LLMs.

## 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). The addition-based methods insert 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), $(IA)^3$ (Liu et al., 2022a), and BitFit (Ben-Zaken et al., 2021). Another approach is called the specification-based approach, which 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 motivated by the observation that fine-tuning has a low intrinsic dimension (Aghajanyan et al., 2021). LoRA (Hu et al., 2021) 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).

In this work, we propose a novel framework, PARADE, for fine-tuning LLMs in a parameter-efficient fashion. Our method is efficient for LLM inference and performs well in downstream tasks.

### 2.2 Capsule network

Capsule networks encapsulate features into groups of neurons, that is, the so-called capsules (Sabour et al., 2017). Initially introduced for a handwritten digit image classification task, capsules have been shown to learn robust representations. The capsule network is widely studied in computer vision (Pawan and Rajan, 2022) and natural language processing (Kim et al., 2018). (Zhang et al., 2018) demonstrates that the capsule network performs well for n-ary relations. (Xiao et al., 2018) uses a capsule network to differentiate different tasks and improve multi-task learning performances. (Aly et al., 2019; Srivastava et al., 2018) applied capsules for sentence classification tasks. Since the beginning of the era of large-scale pre-trained models, capsule networks have been paid less attention. In this work, we apply the idea of capsule networks to the pooling operation in the vector generators of our PARADE framework.

## 3 Methods

### 3.1 Preliminaries

**Transformer model** Currently, the most widely used open-sourced language models and large language models adopt the stacked Transformer architecture (Vaswani et al., 2017). The 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. Denote the input sequence's length as $l$, the hidden states' dimension as $d_{model}$, and the dimension at the FFN module as $d_{ffn}$. The MHA is given as follows:[3]

$$\text{softmax}\left(\frac{QK}{\sqrt{d_{model}}}\right)V, \quad (1)$$

where $Q = xW^Q$, $K = xW^K$, $V = xW^V$, $x \in \mathbf{R}^{l \times d_{model}}$ is the input tensor. $W^Q, W^K, W^V \in \mathbf{R}^{d_{model} \times d_{model}}$ are the query, key, and value projection layers (denoted as the Query, Key, and Value modules). The FFN module consists of linear transformations and an activation function $g^{ffn}$ such as ReLU or GELU (Hendrycks and Gimpel, 2016). Take the FFN module in the LlaMA-2 models (Touvron et al., 2023) as example:

$$(g^{ffn}(G) * U)W^D, \quad (2)$$

where $G = xW^G$, $U = xW^U$, $W^G, W^U \in \mathbf{R}^{d_{model} \times d_{ffn}}$ (denoted as Gate and Up module).

**Task formulation** 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 Motivation

A good PEFT method should have the following properties: (a) Introducing as few additional tunable parameters as possible, limiting the additional storage and memory costs. (b) Strong performance in the adaptation of LLMs for downstream tasks. (c) efficient under the multi-tenant setting, where a single LLM backbone has to be paired with multiple sets of PEFT parameters from different tenants, and during batch-wise inference, a batch may contain samples from different tenants/tasks corresponding to different PEFT methods. The LoRA-based methods (Hu et al., 2021) are considered the most performant PEFT method for LLMs (Ding et al., 2022; Zhu et al., 2023; Xu et al., 2023). However, under the multi-tenant setting, significant latency is introduced since these methods require the LoRA modules to be added to almost

[3] We omit the multi-head setting for simplicity of illustrations.

all the Transformer modules. Prompt tuning (Liu et al., 2021, 2022b) inserts learnable soft prompts of length like 32 or 64 at the beginning of the input sequence, introducing additional complexity and latency. $(IA)^3$ (Liu et al., 2022a) and BitFit (Ben-Zaken et al., 2021) modifies the hidden representations/activations by adding or multiplying learnable vectors in an element-wise fashion, thus are efficient for inference. However, IA3 (Liu et al., 2022a) or BitFit (Ben-Zaken et al., 2021) have the following drawbacks: (a) in-flexible since the tunable parameters are fixed given a LLM backbone. (b) lag behind the LoRA method in terms of downstream performance. One possible reason may be that they use the same learnable vectors for different samples of different difficulty and thus may not be effective in directing the attention mechanism of LLMs.

### 3.3 PARADE

Now we present the framework of our novel Prompt Aware Representation ADjustmEnt (PARADE) method, which can be seen as a novel extension of (Liu et al., 2022a) and (Ben-Zaken et al., 2021).

**Formulation** In order to generate responses, the input instructions have to go through the LLM backbone to obtain the hidden representations. Denote the hidden state of the input instruction with length $T_{ins}$ at the current Transformer layer as $\mathbf{h}$. As shown in Figure 1, the vector generator VG() use $\mathbf{h}$ as input to generate three learned vectors, $l_q, l_v \in \mathbb{R}^{d_{model}}$ and $l_u \in \mathbb{R}^{d_{ffn}}$, with a vector generator:

$$l_q,\ l_v,\ l_u = \text{VG}(\mathbf{h}), \tag{3}$$

and these generated vectors are used to modify the hidden representations in the self-attention and FFN modules. Thus, under PARADE, the self-attention mechanism of Transformer (in Equation 1) is changed to

$$\text{softmax}\left(\frac{Q^{'}K}{\sqrt{d_{model}}}\right)V^{'}, \tag{4}$$

where $Q^{'} = l_q \odot Q$, $V^{'} = l_v \odot V$, $\odot$ denotes the element-wise product. And the FFN module (Equation 2) is modified to

$$(g^{ffn}(G) \odot U^{'})W^D, \tag{5}$$

where $U^{'} = l_u \odot U$.[4][5]

**Vector generator** Now, we introduce the core of our PARADE framework, the vector generator VG(). It takes $\mathbf{h}$ as input and consists of a pooling module and a pair of projection operations. It first projects $H$ from dimension $d_{model}$ to dimension $r < d_{model}$ via a projection layer $W_{down}^{vg} \in \mathbb{R}^{d_{model} \times r}$. Then, it obtains the vector representation through a pooling module Pooler(). Then the obtained vector representation will go through an activation function $g^{vg}$ and be projected to dimension $d_{out} = 2 * d_{model} + d_{ffn}$ via another projection layer with weight $W_{up}^{vg}$ and bias $b_{up}^{vg}$. Formally, the generator is given by the following equation:

$$\begin{aligned} l &= (g^{vg}(\text{Pooler}(\mathbf{h}W_{down}^{vg})))W_{up}^{vg} + b_{up}^{vg}, \\ l_q,\ l_v,\ l_u &= \text{Split}(l), \end{aligned} \tag{6}$$

where Split() is to split the vector into three vectors of dimension $d_{model}$, $d_{model}$, $d_{ffn}$, respectively.

Note that the decoder-based causal language models (CLM) usually employ the KV cache mechanism[6] during generation to improve efficiency. Our vector generators work seamlessly with the KV cache mechanism since the generated vectors, $l_q$, $l_v$, $l_u$, are generated when the input instruction (or prompt) is passed through the LLM for the first time. These three vectors will be reused in the subsequent generation steps, and the vector generators will not be called again. In comparison, the LoRA method provides reparameterizations to the model parameters, and its low-rank weight matrices have to participate in the forward calculations during each generation step, causing higher latency.

**Pooler** From the above Equation 3, Pooler() is responsible for extracting and aggregating the semantic information from the input instructions into a vector. For the vector generators to generate high-quality vectors to regulate the behavior of the LLM backbone, it is of great importance for the Pooler to play its role. It is natural to use the most naive pooler to take the last token's hidden representations as the pooler's output. However, with our initial experiments, we find that applying a more complex architecture for Pooler is beneficial.

[4] We use the "broadcasting notation" in the Equations 4 and 5. Take so that the $(m, n)$-th entry of $U^{'}$ is $l_u[n] \odot U[m, n]$.

[5] From our preliminary experiments, we find that generating adjustment vectors for the other hidden states like K and G will not result in clear performance gains.

[6] See the blog post for an in-depth explanation of KV-cache: https://www.dipkumar.dev/becoming-the-unbeatable/posts/gpt-kvcache/.

Inspired by the previous works of capsule network (Sabour et al., 2017; Zhang et al., 2018; Aly et al., 2019) in the field of both computer vision and natural language processing, we now propose a lightweight but effective pooler, called attentional capsule pooler (AC pooler). Denote the input of the pooler as $\mathbf{h}^{'}$. Being consistent with the terminology of capsule networks (Sabour et al., 2017), we call each hidden vector $h_j^{'} = \mathbf{h}^{'}[j]$ as an input capsule ($j < T_{ins}$), and our target is to extract the semantic information of $\mathbf{h}^{'}$ into $K$ output capsules $[c_1, ..., c_K]$, i.e., $K$ fixed-length vectors, where $c_k \in \mathbb{R}^{d_{cap}}$, and $d_{cap} = d_{model}/K$, $k < K$. Denote the parameters of the $k$-th output capsule as $W_k^{cap} \in \mathbb{R}^{d_{model} \times d_{cap}}$. The information flowing from $h_j^{'}$ to $c_k$ is denoted as $m_{j \to k}$:

$$m_{j \to k} = h_j^{'} W_k^{cap}. \tag{7}$$

And then, $c_k$ is obtained by aggregating all the incoming information flow:

$$\begin{aligned} s_k &= \sum_{j=1}^{T_{ins}} n_{j,k} m_{j \to k}, \\ c_k &= \frac{\|s_k\|^2}{1 + \|s_k\|^2} \frac{s_k}{\|s_k\|}, \end{aligned} \tag{8}$$

where $n_{j,k}$ is a scalar indicating how much information is transferred from $h_j$ to $c_k$. The previous work (Zhang et al., 2018; Aly et al., 2019) usually initialized $n_{j,k}$ with zeros and used an iterative process to calculate its value, which inevitably introduces high latency. In this work, we propose initializing $n_{j,k}$ with attention scores and obtaining a more appropriate value via one iteration step. We initialize a learnable vector $v_k \in \mathbb{R}^{d_{model}}$ for each output capsule, and $n_{j,k}$ is initialized with:

$$\begin{aligned} m_{j,k} &= v_k^{\intercal} h_j^{'}, \\ n_{j,k} &= \text{Softmax}(m_{j,k}), \end{aligned} \tag{9}$$

where Softmax denotes the softmax normalization among the $j$ index. Now we have an initial value for $n_{j,k}$, and $c_k$ can be calculated via Equations 7 and 8. To adjust the values of $n_{j,k}$, we now calculates its value once again via

$$\begin{aligned} m_{j,k} &\leftarrow m_{j,k} + c_k^{\intercal} h_j^{'}, \\ n_{j,k} &= \text{Softmax}(m_{j,k}). \end{aligned} \tag{10}$$

With the new values of $n_{j,k}$, the output capsule $c_k$ is calculated again. Finally, the $K$ output capsules $[c_1, ..., c_K]$ are concatenated to a single vector and are the output of the capsule pooler.

# 4 Experiments

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

## 4.1 Baselines

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

**Representation modification** We consider the following methods for direct representation modifications: (a) BitFit (Ben-Zaken et al., 2021), which fine-tunes the model by adding learnable vectors to the hidden representations of LLMs as bias terms. (b) $(IA)^3$ (Liu et al., 2022a) multiplies learnable vectors to the hidden representations. For these two methods, the learnable vectors are fixed across different samples of the task at hand. To adjust the number of tunable parameters for these two methods, we first initialize the vectors in a smaller dimension $r^{'} < d_{model}$, then we use a learnable matrix to project the vectors to dimension $d_{model}$.

**Adapter-based tuning** We consider the following adapter tuning baselines: (1) Houlsby-Adapter (Houlsby et al., 2019); (2) Parallel-Adapter proposed by He et al. (2021); (3) AdapterDrop (Rücklé et al., 2020); (4) 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).

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

The baselines are implemented using Transformers (Wolf et al., 2020a) or 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 the following benchmark tasks: (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. (c) a constrained natural language generation task E2E (Novikova et al., 2017). (d) a mathematical solving dataset GSM8K (Cobbe et al.,

| Method | Tunable Params | SST-2 (acc) | RTE (acc) | QNLI (acc) | BoolQ (acc) | COPA (acc) | ReCoRD (f1-em) | SQuAD (f1-em) |
|---|---|---|---|---|---|---|---|---|
| ***Baselines*** | | | | | | | | |
| P-Tuning | 9.4M | 92.4 | 79.7 | 91.9 | 84.1 | 89.6 | 89.2 | 86.5 |
| P-tuning v2 | 9.4M | 92.8 | 80.6 | 92.1 | 85.2 | 90.1 | 89.4 | 86.9 |
| IDPG | 8.4M | 92.6 | 80.8 | 92.2 | 85.3 | 90.1 | 89.6 | 87.2 |
| LPT | 8.4M | 92.8 | 81.3 | 92.3 | 85.7 | 90.2 | 89.9 | 87.4 |
| Housbly-Adapter | 9.4M | 92.9 | 80.6 | 92.4 | 84.5 | 90.4 | 89.8 | 87.3 |
| Parallel-Adapters | 9.4M | 93.0 | 80.5 | 92.5 | 85.1 | 90.2 | 90.1 | 87.7 |
| AdapterDrop | 9.2M | 92.7 | 80.1 | 92.3 | 84.5 | 89.8 | 89.8 | 87.4 |
| Learned-Adapter | 9.5M | 93.6 | 81.5 | 92.4 | 86.2 | 90.4 | 90.1 | 87.6 |
| LoRA | 10.0M | 93.6 | 82.6 | 92.5 | 86.7 | 90.7 | 90.2 | 87.7 |
| AdaLoRA | 10.0M | 93.6 | 82.9 | 92.6 | 86.6 | 90.8 | 90.5 | 87.5 |
| SSP | 8.6M | 93.5 | 82.6 | 92.6 | 86.4 | 91.1 | 90.0 | 87.4 |
| BitFit | 10.9M | 92.9 | 81.9 | 92.2 | 85.6 | 90.5 | 89.8 | 87.2 |
| $(IA)^3$ | 9.8M | 93.0 | 82.7 | 92.5 | 86.4 | 90.7 | 90.1 | 87.6 |
| ***Our proposed methods*** | | | | | | | | |
| PARADE | 8.9M | **94.2** | **83.7** | **93.2** | **87.4** | **91.8** | **91.0** | **88.3** |

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.7.

2021). (e) a SQL generation task WikiSQL (Zhong et al., 2017). (f) Alpaca dataset (Taori et al., 2023) for general-purpose 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.7.

### 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 Pythia-1.4B (Biderman et al., 2023).

**Prediction heads** When fine-tuning LlaMA-2 7B, we only consider the supervised fine-tuning (SFT) setting (Ouyang et al., 2022). After receiving a prompt or instruction, all the predictions are generated using the language modeling head (LM head). No additional prediction heads are installed for making categorical or numerical predictions. For decoding during inference, we use beam search with beam size 3.

**Hyper-parameters for the PARADE framework** In our experiments, unless otherwise specified, we set: (a) the bottleneck dimension $r$ of the PARADE vector generator to 12, (b) the number of capsules $K$ to 4, (c) the activation function $g^{vg}$ to the GeLU activation function (Hendrycks and Gimpel, 2016). (d) The $W^{G}_{down}$ and $W^{cap}_{k}$ parameters are initialized with uniformly with mean 0 and std 0.02. $W^{G}_{up}$ is zero initialized, and $b^{vg}_{up}$ is initialized with ones. Under the above settings, our PARADE method will introduce 8.9M tunable parameters to the LlaMA-2 7B backbone. The hyper-parameters for training is specified in Appendix E.

**Reproducibility** We run each task under five 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 in Appendix E.

### 4.4 Main results

**Results on the GLUE and SuperGLUE tasks** The experimental results on the three classification tasks and 4 question answering tasks are presented in Table 1. We present the number of tunable parameters in the second column of Table 1. Table 1 reveals that our PARADE method outperforms the baseline methods across all seven tasks, with comparable or fewer tunable parameters. In particular, PARADE outperforms previous SOTA representation modification methods like $(IA)^3$ and strong LoRA style baselines like LoRA and AdaLoRA

| Method | E2E (rouge-l) | GSM8K (acc) | WikiSQL (acc) |
|---|---|---|---|
| $(IA)^3$ | 70.2 | 34.3 | 84.2 |
| LoRA | 70.7 | 35.1 | 85.4 |
| AdaLoRA | 70.8 | 35.2 | 85.2 |
| PARADE | 71.4 | 36.1 | 85.8 |

Table 2: Results for different PEFT methods on the E2E, GSM8K, and WikiSQL benchmark. The backbone LLM is LlaMA-2 7B. The metrics are explained in Appendix B.7.

| Method | Avg GPT-4 score (↑) | ROUGE-L (↑) |
|---|---|---|
| AdaLoRA | 7.01 | 51.1 |
| PARADE | 7.23 | 52.6 |

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

with comparable parameters. These results demonstrate that our method is good at downstream task adaptation of large language models.

**Results on the three specialized generation task** For the E2E, GSM8K, and WikiSQL benchmarks, the results are reported in Table 2. The results show that our PARADE method successfully outperforms LoRA, AdaLoRA, and $(IA)^3$ on the three tasks.

**Results for general-purpose instruction tuning** After the LlaMA-2 7B is fine-tuned on the Alpaca dataset with our PARADE and AdaLoRA 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.7. The average score provided by GPT-4 is presented in Table 3, along with the ROUGE-L scores calculated by considering the GPT-4's answers as ground truth. Consistent with the previous experiments (Table 1 and 2), our PARADE method outperforms the AdaLoRA method in terms of the GPT-4 scores and ROUGE-L, demonstrating that PARADE can enhance the instruction tuning quality of large language models. A case study of answers generated by different methods is presented in Table 8, showcasing that PARADE leads to better instruction-tuned LLMs.

| Method | Beam size | Speed (tps) | Memory cost (MiB) |
|---|---|---|---|
| LoRA | 1 | 25.1 | 14616 |
| | 3 | 21.9 | 16104 |
| $(IA)^3$ | 1 | 33.1 | 14572 |
| | 3 | 27.6 | 16036 |
| PARADE | 1 | 32.5 | 14508 |
| | 3 | 27.3 | 15982 |

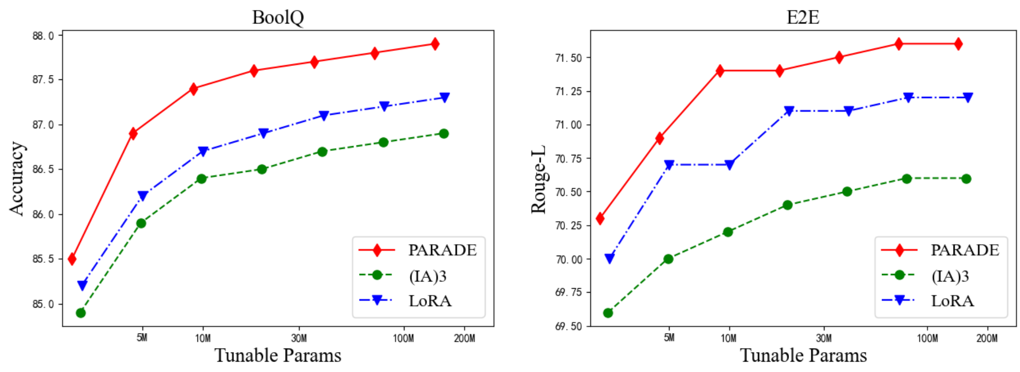
Table 4: The memory and speed of LlaMA-2 7B for generating responses given the input instruction (Appendix F), with different PEFT methods.

### 4.5 Ablation studies and analysis

**Analysis of the inference efficiency** To demonstrate the inference efficiency of our PARADE method, we now compare the GPU memory and generation speed of PARADE, LoRA, and $(IA)^3$. In this experiment, LoRA parameters are not merged to the backbone to mimic the single-LLM multi-tenant setting (Chen et al., 2023). The detailed settings for efficiency analysis are presented in Appendix F. We present two metrics for measuring efficiency: (a) peak memory cost during generation. (b) tokens generated per second (tps). The results are presented in Table 4.

From Table 4, one can see that under the hyper-parameter settings specified in the main experiments (Table 1), our PARADE method can outperform $(IA)^3$ with comparable tunable parameters, memory costs, and generation speed during generation. We can also see that the PARADE is much faster than LoRA. The speed advantages of PARADE over LoRA come from the following factors: (a) our vector generators are lightweight and efficient during inference. (c) The vectors, $l_q$, $l_v$, $l_u$, are only generated once the input instructions are passed to the LLM and before generating the first new token. The vectors will be reused in the following generation steps with KV-cache, and the vector generators are not called repetitively. In contrast, the LoRA method requires the model to call the LoRA modules at each generation step, resulting in higher latency.

**Ablation study of our PARADE framework** We now consider the following variants of PARADE: (a) PARADE-1 uses the most direct pooling operation of using the instructions' last tokens as the pooled tensor. (b) PARADE-2: substituting our attentional capsule pooler to the one in (Zhang et al., 2018) with one iteration step. (c) PARADE-3 asks the vector generators to generate $l_q$, $l_k$, $l_v$, $l_g$, and $l_u$, the adjusting vectors for Query, Key, Value, Gate, and Up modules. We set $r = 8$ for




(a) BoolQ (b) E2E

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

PARADE-3 so that the number of added tunable parameters is 9.8M. (d) PARADE-4 asks the vector generators to generate $l_q$, $l_v$, the adjusting vectors for the Query and Value modules. We set $r = 24$ for PARADE-4 so that the number of added tunable parameters is 9.4M. The BoolQ, E2E, and SQuAD tasks' experimental results are reported in Table 6 of Appendix G. The results show that our primary model, PARADE (as in Table 1), outperforms the four variants, demonstrating that (a) our attentional capsule pooler is beneficial for the downstream task performance by providing better aggregation of the input instruction's semantic information. (b) PARADE-2 fails since (Zhang et al., 2018) requires multiple iterations to calculate the proper information flow coefficients. (c) The comparison between PARADE and PARADE-3 shows that providing adjusting vectors for more modules does not lead to apparent performance gain. (c) The comparison between PARADE and PARADE-4 shows that not providing adjusting vectors for the FFN modules leads to worse downstream performance.

**Visualization of the attention maps under PARADE** In this section, we visualize the attention maps for the 0-th and 16-th attention heads on the 1-th, 9-th, 17-th and 25-th Transformer layers, with PARADE or with $(IA)^3$, in Figure 3, 4, 5 and 6. Compared to the $(IA)^3$ method, our PARADE method makes the LLM focus less attention on the ending token of the system prompt. It encourages more attention among input instructions' token pairs.

**Comparisons under different budgets of tunable parameters** We vary the budget of tunable parameters for PARADE by modifying the values of $r = 12$ to $\{3, 6, 24, 48, 96, 192\}$. We also vary the $(IA)^3$ and LoRA methods' tunable parameter numbers. The experimental results on the BoolQ and E2E tasks are presented in Figure 2(a) and 2(b). The results show that under different tunable parameter budgets, our PARADE method can consistently outperform the LoRA and $(IA)^3$ methods.

**Ablation on the pretrained backbones** Our main experiments are conducted on the LlaMA-2 7B model. To demonstrate the broad applicability of our method, we now conduct experiments on GPT2-large and Pythia-1.4b. The results are reported in Table 7. We can see that on these two backbones, our method can also outperform the baseline methods.

## 5 Conclusion

This work presents the Prompt Aware Representation ADjustmEnt (PARADE), a novel method for parameter-efficient fine-tuning of large language models. First, we install a vector generator to generate adjusting vectors for regulating the behavior of the LLM backbones. The vector generator takes the hidden states of the input instructions as inputs and contains a lightweight bottleneck architecture. PARADE is more efficient than LoRA during inference since it works seamlessly with the KV-cache mechanism. Second, in order to enhance the performance of PARADE, we propose an attentional capsule-based pooler that can better aggregate semantic information from the input instructions. Experiments on various tasks demonstrate that our PARADE method outperforms the baseline methods while being efficient for inference.

## 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, GPT2-large, Pythia-1.4b), while maintaining efficiency during inference. 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 experimented. 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 broader types of tasks. And we will explore it in future work.

## Ethics Statement

The finding and proposed method aims to improve the parameter-efficient tuning in terms of performance and efficiency. 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.

Rami Aly, Steffen Remus, and Chris Biemann. 2019. Hierarchical multi-label classification of text with capsule networks. In *Annual Meeting of the 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.

Stella Biderman, Hailey Schoelkopf, Quentin G. Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. *ArXiv*, abs/2304.01373.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, Arvind Krishnamurthy University of Washington, and Duke University. 2023. Punica: Multi-tenant lora serving. *ArXiv*, abs/2310.18547.

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*, abs/2110.14168.

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

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.

Wensheng Gan, Shicheng Wan, and Philip S. Yu. 2023. Model-as-a-service (maas): A survey. *2023 IEEE International Conference on Big Data (BigData)*, pages 4636–4645.

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.

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.

Jaeyoung Kim, Sion Jang, Sungchul Choi, and Eunjeong Lucy Park. 2018. Text classification using capsules. *ArXiv*, abs/1808.03976.

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*.

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*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *ArXiv*, abs/2205.05638.

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*.

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.

S. J. Pawan and Jeny Rajan. 2022. Capsule networks for image classification: A review. *Neurocomputing*, 509:102–120.

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*.

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*.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. *ArXiv*, abs/1710.09829.

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.

Saurabh Srivastava, Prerna Khurana, and Vartika Tewari. 2018. Identifying aggression and toxicity in comments using capsule network. In *TRAC@COLING 2018*.

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.

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.

Liqiang Xiao, Honglun Zhang, Wenqing Chen, Yongkun Wang, and Yaohui Jin. 2018. Mcapsnet: Capsule network for text with multi-task learning. In *Conference on Empirical Methods in Natural Language Processing*.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *ArXiv*, abs/2312.12148.

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.

Ningyu Zhang, Shumin Deng, Zhanlin Sun, Xi Chen, Wei Zhang, and Huajun Chen. 2018. Attention-based capsule networks with dynamic routing for relation extraction. *ArXiv*, abs/1812.11321.

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.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *ArXiv*, abs/1709.00103.

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

## A.1 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.

## A.2 Literature on the 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 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 matri-

| Datasets | #train | #dev | #test | $\lvert\mathcal{Y}\rvert$ | 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 | - | rouge-l |
| GSM8K | 7K | 0.5K | 1K | - | Math reasoning | - | acc |
| WikiSQL | 61k | 9K | 17K | - | SQL generation | - | acc |
| Alpaca | 51k | - | - | - | 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.

ces 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 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 GSM8K benchmark

GSM8K is a dataset of 8.5K high quality linguistically diverse grade school math word problems created by human problem writers. The dataset is segmented into 7.5K training problems and 1K test problems. These problems take between 2 and 8 steps to solve, and solutions primarily involve performing a sequence of elementary calculations using basic arithmetic operations ($+-\times\div$) to reach the final answer. A bright middle school student should be able to solve every problem. It can be used for multi-step mathematical reasoning. We randomly select 0.5k samples from the training set to be the dev set.

### B.5 WikiSQL dataset

WikiSQL consists of a corpus of 87,726 hand-annotated SQL query and natural language question pairs. These SQL queries are further split into training (61,297 examples), development (9,145 examples) and test sets (17,284 examples). It can be used for natural language inference tasks related to relational databases. In this work, we will ask the LLMs to generate SQL queries based on the given natural language questions.

### B.6 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[7]. 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.

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

### B.7 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 the ROUGE-L metric (denoted as rouge-l). We rely on the HuggingFace Evaluate package[8] for computing this metric.

For the GSM8K task, we will directly consider the correctness of the final answers. Thus, we report accuracy (denoted as acc).

For the WikiSQL, we will consider the correctness of the generated SQL queries. A predicted SQL query is correct if and only if it can be executed and obtains the same results with the ground truth.

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 AdaLoRA and PARADE'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.

## 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. First, following LlaMA-2 (Touvron et al., 2023), we use a system prompting template, in which <query> denotes the user input, <response>

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

denotes the assistants' targeted responses. All the samples will be input into this template before being fed to the LLMs.

```
<s>[INST] <<SYS>>
You are a helpful, respectful and honest
assistant.
<</SYS>>

<query>[/INST]<response></s>
```

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]
```

**Templates for GSM8K** The samples in this task consists of a math question [question], and a targeted response, [target] which is the reasoning or calculation steps for the math question.

Template for prompt:

```
Answer the following math quesition:
[ref]
Instruction: please think step by step.
```

Template for response:

```
[target]
```

**Templates for WikiSQL** The samples in this task consists of a natural language query [query], and information for the SQL table [table_info], and a targeted response containing the SQL query, [target] which is the reasoning or calculation steps for the math question.

Template for prompt:

```
Answer the following query by writing a
SQL query on the given table:
[query]
Table information:
[table\_info].
```

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 the P-tuning method, the soft prompts' length is 64, and the soft prompts is first initialized with dimension 36, and then a learnable projection layer projects it to the same dimension with the LlaMA-2 backbone. For P-tuning V2, the number of prompt tokens at each layer is set to 64. For LPT and IDPG, the bottleneck dimension is set to 1024, and the number of soft tokens is set to 4.

For the adapter-based methods, Houlsby-Adapter and AdapterDrop, the bottleneck dimension is set to 18, and the adapter modules are added on the self-attention and feed-forward module. For the Parallel-Adapter and Learned-Adapter, the bottleneck dimension is set to 36, and the adapter modules are connected to the whole block.

For LoRA, the initial rank at each module is set to 4. For AdaLoRA, the initial rank at each module is set to 8, 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 PARADE and the other baselines.

For BitFit, the bias vectors are initialized with dimension 8, and then a learnable projection layer projects it to the same dimension with the LlaMA-2 backbone. For $(IA)^3$, the activation adjusting vectors are added the Query, Key, and Up activations. The adjusting vectors are initialized with dimension 16, and then a learnable projection layer projects it to the same dimension with the LlaMA-2 backbone.

**Training settings for PEFT methods** We use the HugginFace Transformers (Wolf et al., 2020b), PEFT (Mangrulkar et al., 2022), or the original code repositories 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 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 early. The best checkpoint on the dev set is used to run predictions on the test set.

## F Appendix: settings for efficiency analysis

In the Table 4 of the main contents, we conduct analysis on the PARADE and other PEFT methods' memory and speed during inference.

The example instruction we used in this analysis is presented below.

```
Generate a blog post of 500 words or less
that discusses the following news article:

The Department of Child Protection (DCP)
must pay compensation and medical expenses
to a youth worker who developed pericarditis
after getting a Covid booster under a
workplace vaccination directive, the South
Australian Employment Tribunal has ruled.

In a decision handed down on 15 January
2024, the Tribunal determined that Daniel
Shepherd’s employment was “a significant
contributing cause” to his injury, which
has since rendered him incapable of
performing his role at work.

Shepherd got a Covid booster in February
2022 as a requirement for his ongoing
employment with the DCP. The DCP admitted
that Shepherd’s pericarditis had been
caused by the booster, but denied
responsibility for the injury, arguing that
it did not arise from Shepherd’s employment,
but from a lawful State Government Public
Health Order (PHO), issued under the
Emergency Management Act 2004 (EMA).
```

We restrict the number of newly generated tokens to be 32 under the method of beam search with beam size equal to 1 or 3. The length of the initial instruction is 278 after adding the soft prompts and special tokens under the IAPT method, and 274 under the LoRA method. The LLM backbone is LlaMA-2 7B model. We run the generation process for 100 times to calculate the average metric values, reducing the randomness.

## G Ablation on the PARADE framework

In the main contents, we consider 4 variants of the PARADE method, and the experiments on the BoolQ, E2E and SQuAD tasks are provided in 6

| Method | BoolQ (acc) | E2E (rouge-l) | SQuAD (f1-em) |
|---|---|---|---|
| PARADE | 87.4 | 71.4 | 88.3 |
| PARADE-1 | 86.6 | 70.8 | 87.8 |
| PARADE-2 | 86.3 | 70.4 | 87.6 |
| PARADE-3 | 87.1 | 71.0 | 88.1 |
| PARADE-4 | 86.5 | 70.3 | 87.5 |

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

| Method | BoolQ (acc) | E2E (rouge-l) | SQuAD (f1-em) |
|---|---|---|---|
| ***Results for GPT-2 large*** | | | |
| $(IA)^3$ | 67.4 | 66.2 | 43.3 |
| LoRA | 67.5 | 66.6 | 43.8 |
| AdaLoRA | 67.6 | 66.5 | 43.7 |
| PARADE | **68.1** | **66.9** | **44.3** |
| ***Results for Pythia-1.4b large*** | | | |
| $(IA)^3$ | 71.3 | 67.9 | 49.8 |
| LoRA | 71.6 | 68.4 | 50.4 |
| AdaLoRA | 71.5 | 68.5 | 50.4 |
| PARADE | **72.2** | **68.9** | **51.1** |

Table 7: Results for different PEFT methods on the BoolQ, E2E and SQuAD benchmarks. The backbone LMs are GPT2-large and Pythia-1.4b. The metrics are explained in Appendix B.7.

## H 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 GPT-2 large (774M parameters) and Pythia-1.4b models. The other experimental settings are kept the same with the main experiments (Table 1). We conduct experiments on the BoolQ, E2E and SQuAD tasks. The results are reported in Table 7.

## I Visualization of the attention maps

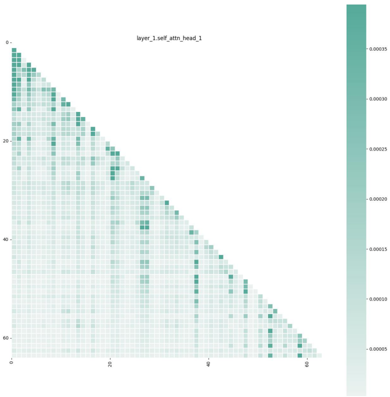
In this section, we visual the attention maps for the first 64 tokens. The backbone model is LlaMA-2 7B at the 1-th, 9-th, 17-th and 25-th attention heads for the 1-th Transformer layer in Figure 3, for the 9-th Transformer layer in Figure 4, for the 17-th Transformer layer in Figure 5, for the 25-th Transformer layer in Figure 6.

## J Case studies of Instruction tuning

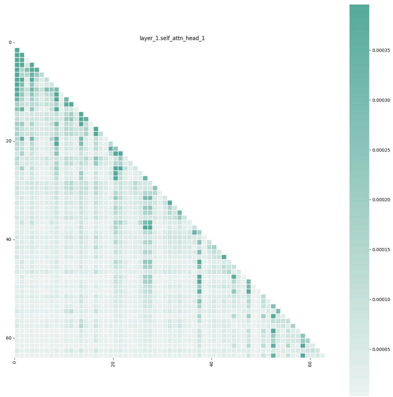
In the Section 4.4 of the main content, we present the overall performance of PARADE and AdaLoRA on the MT-Bench, after fine-tuning LlaMA-2 7B on the Alpaca dataset. Now we present a concrete examples in Table 8 to showcase the Superiority of PARADE.

| | |
|---|---|
| Query | Embrace the role of Sheldon from "The Big Bang Theory" as we delve into our conversation. Don't start with phrases like "As Sheldon". Let's kick things off with the following question: "What is your opinion on hand dryers?" |
| *PARADE's response* | |
| Responses | Oh, hand dryers. *adjusts glasses* Well, Sheldon's opinion on hand dryers is that they are an abomination. *pauses* I mean, think about it. You're in a public restroom, minding your own business, and then you're forced to use a hand dryer. *shudders* It's just so... uncivilized. *adjusts glasses again* I mean, what's wrong with a good ol' fashioned paper towel? *chuckles* Now that's civilized. *adjusts glasses once more* But hey, to each their own, right? smirks* What's your take on hand dryers, my friend? |
| GPT-4's comments | Response 1 Review: Assistant 1 effectively embraces the persona of Sheldon with an engaging and character-consistent response. The use of mannerisms such as adjusting glasses and pausing, along with the emotional expression ("shudders," "chuckles," "smirks"), adds to the authenticity of the portrayal. The content is presented in a conversational tone that invites further discussion by asking for the user's opinion at the end. However, while entertaining, Assistant 1's response does not provide factual information or strong arguments to support Sheldon's (or their own) view on hand dryers; it stays more within a subjective perspective that reflects personal distaste rather than objective reasoning. |
| GPT-4 score | 7 |
| *AdaLoRA's response* | |
| Responses | As Sheldon, I have a strong opinion on hand dryers. I believe that paper towels are superior to hand dryers for several reasons. First and foremost, paper towels are more hygienic than hand dryers. Hand dryers can spread germs and bacteria, while paper towels are disposable and can be thrown away after use. Additionally, hand dryers can be loud and disruptive, while paper towels are quieter and less intrusive. Finally, paper towels are more environmentally friendly than hand dryers, as they require less energy to operate and produce less waste. |
| GPT-4's comments | Response 2 Review: Assistant 2 begins by disregarding instructions not to start with phrases like "As Sheldon", which indicates either an oversight or misunderstanding of the task requirements. Despite this error, Assistant 2 offers a well-structured argument against hand dryers from what could be perceived as Sheldon's point of view, citing hygiene concerns, noise levels, and environmental impact as reasons paper towels might be considered superior. This response includes logical reasoning that users can follow easily but lacks Sheldon's unique voice and personality traits which were part of what was asked for in embodying his role. |
| GPT-4 score | 6 |

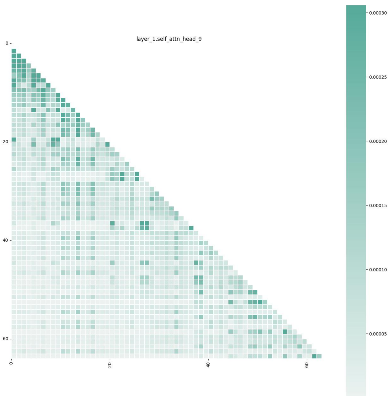
Table 8: An example showcasing that the fine-tuning quality of PARADE is higher than AdaLoRA.



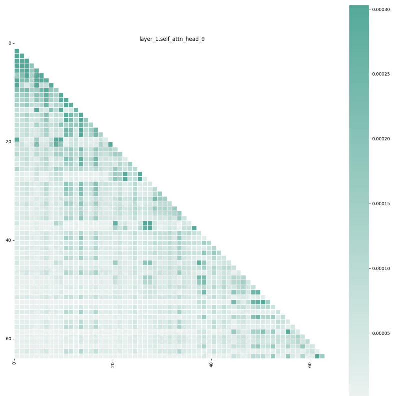

(a) 1-th head



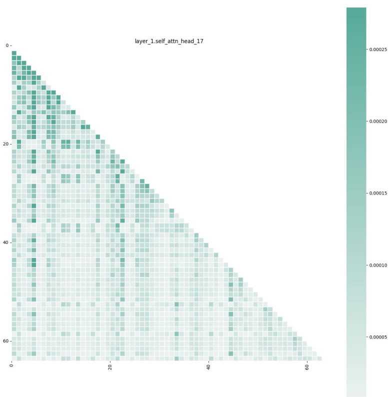

(b) 1-th head



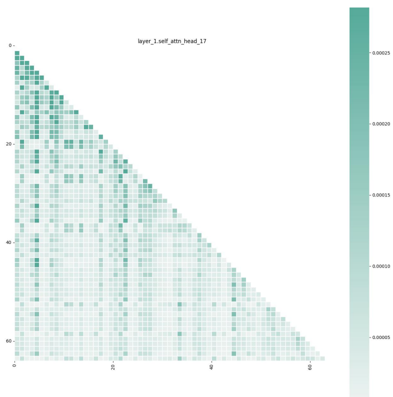

(c) 9-th head




(d) 9-th head

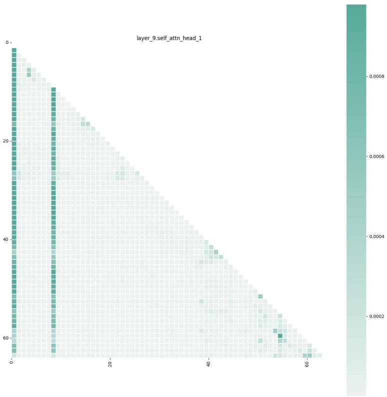



(e) 17-th head

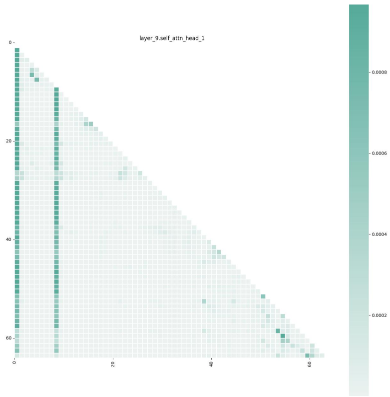



(f) 17-th head

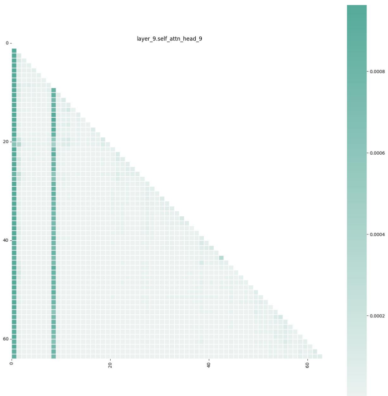
Figure 3: The attention maps of LlaMA-2 7B at the 1-th, 9-th and 17-th attention heads on the 1-th Transformer layer. The left column is the attention maps from LlaMA-2 7B with PARADE, and the right column is the ones from LlaMA-2 7B with $(IA)^3$.



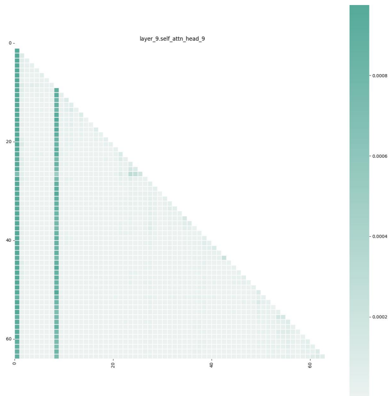

(a) 1-th head



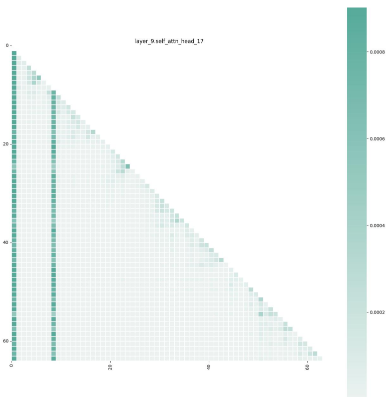

(b) 1-th head



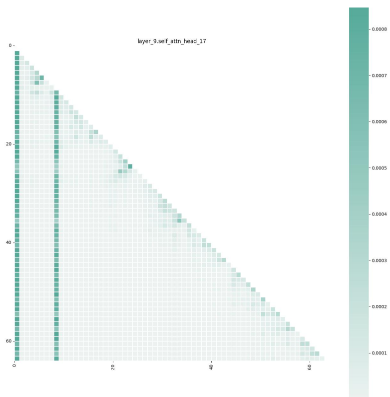

(c) 9-th head




(d) 9-th head

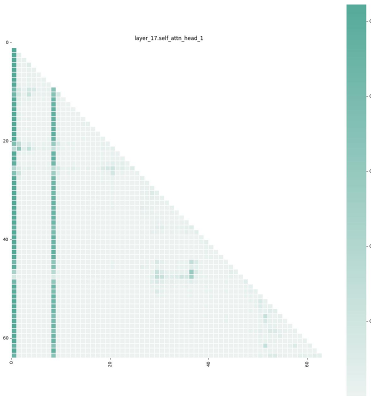



(e) 17-th head

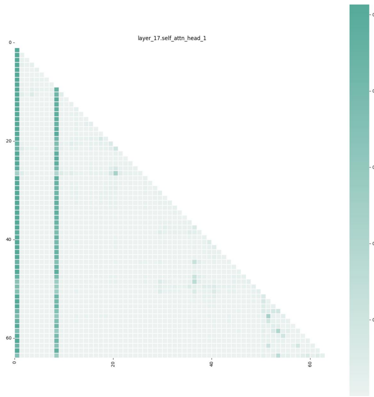



(f) 17-th head

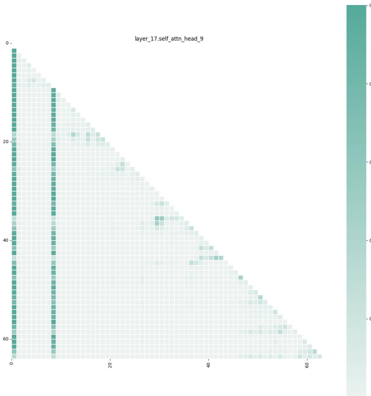
Figure 4: The attention maps of LlaMA-2 7B at the 1-th, 9-th and 17-th attention heads on the 9-th Transformer layer. The left column is the attention maps from LlaMA-2 7B with PARADE, and the right column is the ones from LlaMA-2 7B with $(IA)^3$.



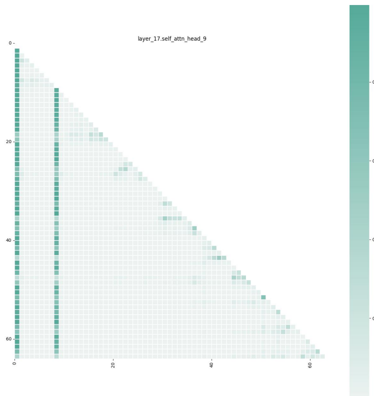

(a) 1-th head



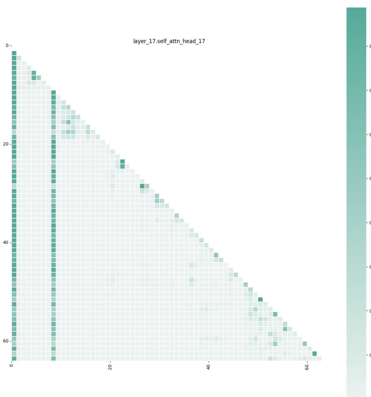

(b) 1-th head



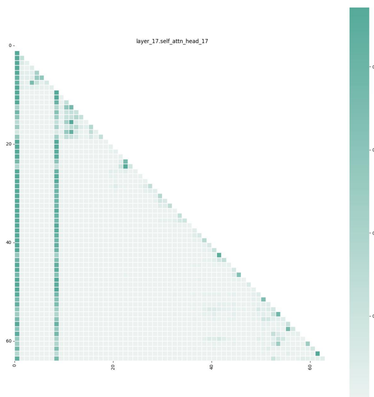

(c) 9-th head




(d) 9-th head

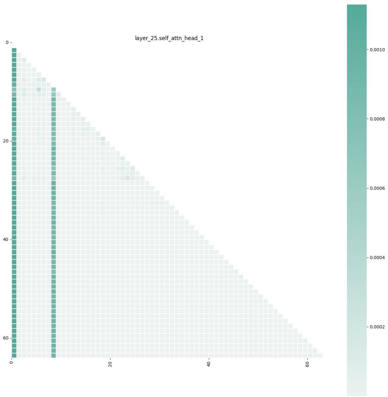



(e) 17-th head

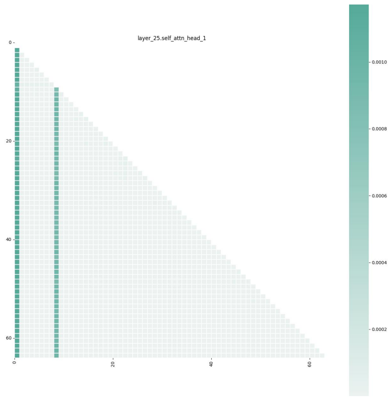



(f) 17-th head

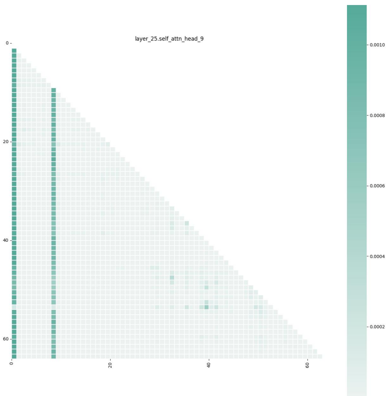
Figure 5: The attention maps of LlaMA-2 7B at the 1-th, 9-th and 17-th attention heads on the 17-th Transformer layer. The left column is the attention maps from LlaMA-2 7B with PARADE, and the right column is the ones from LlaMA-2 7B with $(IA)^3$.



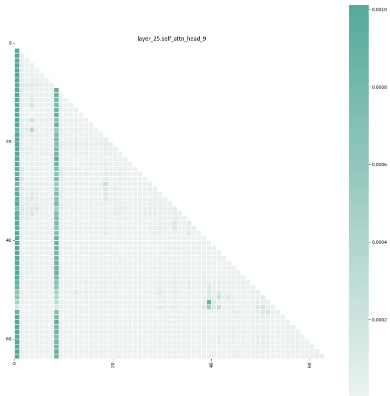

(a) 1-th head



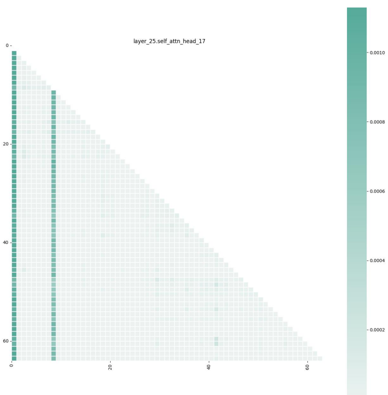

(b) 1-th head



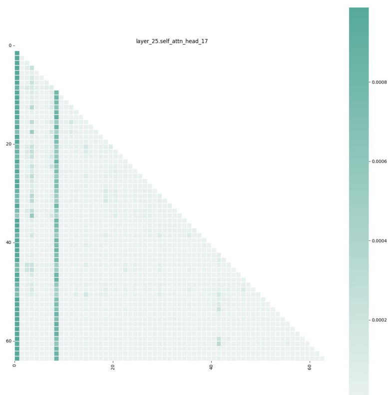

(c) 9-th head




(d) 9-th head




(e) 17-th head




(f) 17-th head

Figure 6: The attention maps of LlaMA-2 7B at the 1-th, 9-th and 17-th attention heads on the 25-th Transformer layer. The left column is the attention maps from LlaMA-2 7B with PARADE, and the right column is the ones from LlaMA-2 7B with $(IA)^3$.