IAPT: Instruction-Aware Prompt Tuning for Large Language Models

Anonymous ACL submission

Abstract

Soft prompt tuning is a widely studied parameter-efficient fine-tuning method. However, it has a clear drawback: many soft tokens must be inserted into the input sequences to guarantee downstream performance. As a result, soft prompt tuning is less considered than Low-rank adaptation (LoRA) in the large language modeling (LLM) era. In this work, we propose a novel prompt tuning method, Instruction-Aware Prompt Tuning (IAPT), that requires only four soft tokens. First, we install a parameter-efficient soft prompt generator at each Transformer layer to generate idiosyncratic soft prompts for each input instruction. The generated soft prompts can be seen as a semantic summary of the input instructions and can effectively guide the output generation. Second, the soft prompt generators are modules with a bottleneck architecture consisting of a self-attention pooling operation, two linear projections, and an activation function. Pilot experiments show that prompt generators at different Transformer layers require different activation functions. Thus, we propose to learn the idiosyncratic activation functions for prompt generators automatically with the help of rational functions. We have conducted experiments on various tasks, and the experimental results demonstrate that (a) our IAPT method can outperform the recent baselines with comparable tunable parameters. (b) Our IAPT 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 many challenging evaluation tasks (Huang et al., 2023; Li et al., 2023) like question answering, 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 (Xu et al., 2023; Ding et al., 2022; Xin et al., 2024). Although LoRA is effective and can bring stable performance with the original setting in Hu et al. (2021), it still brings inconvenience under the multi-tenant setting (Chen et al., 2023): it has to add LoRA modules to multiple weights of the Transformer layer and introducing significant additional latency in every generation steps 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 Instruction-Aware Prompt Tuning (IAPT). We fine-tune the LLMs by inserting instruction

[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`.

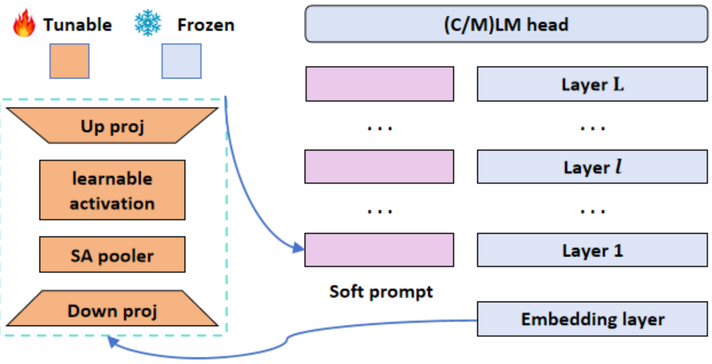


Figure 1: Schematic illustration of our IAPT method. **Left**: The prompt generator which consists of a down-projection, a self-attention based pooler (SA pooler), a learnable activation whose curvature is learned in the downstream task, and a up-projection. **Right**: The prompt generator uses the instructions' hidden states as the input tensors, and output the generated soft tokens which will be concatenated to the next layer's hidden states.

aware soft prompts to each Transformer layer (Figure 1). To flexibly regulate the attentions of LLMs, we install a prompt generator before each Transformer layer to generate the soft prompts by taking the input instruction's hidden states as input. The prompt generator is a lightweight module containing a down-projection layer, a self-attention based pooling layer, an activation function, and an up-projection layer. To enhance the expressiveness of prompt generators, we propose to automatically learn the idiosyncratic activation functions for different prompt generators with the help of rational functions.

We conduct extensive experiments on a wide collection of tasks, including sentiment classification, natural language inference, question answering, constrained sentence generation, math reasoning, SQL query generation, and instruction tuning, to demonstrate the effectiveness of our IAPT method. Notably, our method can consistently outperform strong PEFT baselines with comparable tunable parameter budgets, especially the recent LoRA variants (Zhang et al., 2023b; Ding et al., 2023) and SOTA prompt tuning methods (Liu et al., 2022c; Wu et al., 2022; Liu et al., 2022b). We also use experiments and 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) We proposed adding a self-attention pooling module in the prompt generator, which can help the different transformer layers share the projection layers, thus improving the parameter efficiency. (c) The activation functions are learned during fine-tuning, which improves the downstream performance.

Our contributions are summarized as follows:

- we propose a novel PEFT method, IAPT, that generates soft prompts conditioned on the input instructions received by the LLMs.
- We propose to employ the self-attention mechanism to obtain high-quality information aggregation of the input instructions, thus generating better soft prompts.
- We propose to learn the activation functions for different prompt generators at different Transformer layers, improving the downstream fine-tuning performance.
- We have conducted extensive experiments and analysis showing that our IAPT framework is (a) practical and outperforms the baselines under comparable tunable 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., 2022c). 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).

2.2 Prompt tuning methods

Prompt tuning (Lester et al., 2021) and P-tuning (Liu et al., 2022c) insert soft prompts 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 PTMs. IDPG (Wu et al., 2022) uses parameterized hypercomplex multiplication (Le et al., 2021) to parameterize soft prompts, improving the parameter efficiency. LPT (Liu et al., 2022b) improves upon IDPG by selecting an intermediate layer to insert soft prompts. SPT (Zhu and Tan, 2023) designs a mechanism to automatically decide which layers to insert new soft prompts or keep the prompts propagated from the previous layer. Our work is different and compliments the existing literature in the following aspects: (a) The above works do not work with the current SOTA large language models and only experiment with relatively simple classification tasks. In comparison, by generating soft prompts conditioned on the input instructions received by the LLMs, our IAPT method works well with the currently best decoder-based LLMs in a wide collection of downstream tasks. (b) Our work can reduce the number of soft tokens from 32-128 to 4 by improving the architectural design of the prompt generators. (c) our work improves the parameter efficiency by sharing the parameters of prompt generators across Transformer layers.

3 Methods

3.1 Preliminaries

Transformer model Currently, the most widely used open-sourced (large) language models adopt the stacked Transformer architecture (Vaswani et al., 2017). Denote the total number of Transformer layers in the LLM as L. 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.

Task format 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. Moreover, 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

Soft prompt tuning is a widely studied PEFT method, which can achieve downstream task adaptations through a minimal number of tunable parameters compared to the (L)LM backbones. However, soft prompt tuning is less applied or studied in the era of large language models (Xu et al., 2023; Ding et al., 2022; Xin et al., 2024) due to the following drawbacks:

- Soft prompt tuning needs to concat a sequence of soft prompts to the input sequence, which inevitably increases the model complexity significantly. According to the experiments in (Zhu et al., 2023), P-tuning V2 (Liu et al., 2022c) has to set the soft prompts' sequence length to at least 32 to make the LLM's downstream task performance to be close to LoRA (Hu et al., 2021).

- Soft prompt tuning can not flexibly adjust the tunable parameter numbers if the soft prompt length is fixed, thus can not conveniently make a tradeoff between the PEFT parameters' expressiveness and the parameter efficiency.

- Most existing work on soft prompt tuning methods assumes that the soft prompt is fixed and shared across all samples within a task or dataset. However, a task may encompass a diverse range of samples, some of which are easy while others are challenging. Therefore, prompts need to exhibit sufficient diversity to effectively modulate the attention patterns of LLMs across different samples, thereby achieving better fine-tuning performance.

Thus, in this work, we propose dynamically generating prompt generators based on the given instructions/prompts to address the above issues.

3.3 Instruction aware prompt generators

In this work, the prompt generators generate soft prompts based on the input instructions received by the LLMs. As shown in Figure 1, 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 l_{ins} right before the current Transformer layer as $\mathbf{h}$.[3] The prompt generator first down-projects $\mathbf{h}$ from dimension d to dimension m ($m \ll d$) via a linear layer MLP_{down}. Then, it obtains the prompt $\mathbf{p}$ with a fixed length l_{sp} through a pooling operation $\text{Pooler}()$. The pooled prompt will go through an activation function g and be up-projected to dimension d via another linear layer MLP_{up}. Formally,

$$\mathbf{p} = \text{MLP}_{up}(g(\text{Pooler}(\text{MLP}_{down}(\mathbf{h})))). \quad (1)$$

Then, the generated soft prompt $\mathbf{p}$ will be concatenated to $\mathbf{h}$ and go through the calculations of the next Transformer layer.

Note that the decoder-based causal language models (CLM) usually employ the KV cache mechanism[4] during generation to reduce computation complexity. Our prompt generators work seamlessly with the KV cache mechanism since the soft prompts are generated when the input instruction (or prompt) is passed through the LLM for the first time. In the subsequent generation steps, the generated soft prompts will be combined into the KV caches and will be reused without repetitively calling the prompt generators. In comparison, the LoRA method provides reparameterizations to the model parameters, and it has to participate in the calculations during each generation step.

[3]Note that the soft prompts propagated from the previous layer will be discarded. That is, each Transformer layer uses newly generated prompts. We will use experiments to demonstrate the validity of this setting.

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

3.4 Self-attention based pooler

Our prompt generator must pool the input instructions of variable lengths to a fixed length. For the pooling operation, the previous literature often chooses average pooling or max pooling (Kim, 2014; Zhu et al., 2021; Zhu, 2021a), which are pointed out by the literature (Zhu, 2021b) that they are prone to weaken important words when the input sequence is long, thus dropping useful information during pooling. Thus, in this work, we utilize the self-attention mechanism in our pooling module $\text{Pooler}()$. Self-Attention assigns each token in the input instruction a weight to indicate the importance of the token. A few crucial tokens to the task will be emphasized, while the less important tokens are ignored. Formally, we initialize a learnable weight matrix $W_{sa} \in \mathbb{R}^{m \times l_{sp}}$, then the self-attention based pooler's calculation processes are:

$$\begin{aligned}
\mathbf{U} &= \mathbf{h}W_{sa}, \\
\mathbf{A} &= \text{Softmax}(\mathbf{U}), \\
\mathbf{p} &= \mathbf{A}^{\intercal}\mathbf{h}, \quad (2)
\end{aligned}$$

where Softmax is the softmax function along the first dimension, and $\intercal$ denotes matrix transpose. In the above equations, each column of W_{sa} is a trainable query vector designated to determine the self-attention weights via dot products between this query and each token. Then, the weights are normalized across the sequence dimension via the softmax normalization function. Corresponding to different soft tokens, different query vectors in W_{sa} can aggregate the input instructions in different aspects, thus providing a high-quality summarization of the instruction's semantic information.

3.5 Learned activation functions

The previous PEFT literature usually set the activation functions in a PEFT module to be ReLU (Mahabadi et al., 2021; Pfeiffer et al., 2021; Liu et al., 2022b) and does not discuss whether this setting is optimal. In addition, the PEFT modules' activation functions in different Transformer layers are usually set to be identical. In our initial exploratory experiments (presented in Appendix H), we find that (a) different downstream tasks require different activation functions for the prompt generators in Equation 1. (b) it is beneficial for prompt generators of different depths to have different activation functions. Thus, how can we find an optimal setting for the prompt generators' activation functions? Exhaustive hyper-parameter search is time and GPU-consuming. Thus, we are motivated to set the activation function to be learnable during training.

We resort to rational activation functions to make the activation functions learnable. Empirically introduced as Padé Activation Units (Molina et al., 2019), rational activation functions are learnable and can approximate common activation functions and learn new ones. The rational activation function $R(x)$ of order m, n is defined as follows:

$$R(x) = \frac{\sum_{j=0}^{m} a_j x^j}{1 + \| \sum_{i=1}^{n} b_i x^i \|}, \quad (3)$$

where a_j and b_i are learnable parameters. The rational activation functions are integrated in image classification models (Molina et al., 2019), sequence modeling (Delfosse et al., 2021a), the policy and critic networks in reinforcement learning (Delfosse et al., 2021b), and Generative Adversarial Networks (Boull'e et al., 2020).

Inspired by the above literature, we propose learning the activation functions in prompt generators via the rational activation functions when finetuning a downstream task. Denote the set of parameters in the learnable activations as Θ and the other parameters in the prompt generators as Ω. Following DARTS (Liu et al., 2019), we consider Θ as architectural parameters and optimize them along with the prompt generators' parameters Ω via bi-level optimization. Due to limited length, we introduced bi-level optimization to Appendix B.

3.6 Cross-layer parameter sharing

To improve our IAPT method's parameter efficiency, we propose sharing the parameters of prompt generators across Transformer layers. Denote the total number of Transformer layers in the LLM as L. We ask every $L_s > 0$ prompt generators to (a) share the parameters in MLP_{up}, MLP_{down}, and the learnable activations, (b) but not to share the parameters in the self-attentional Pooler. We will use experiments to show that the self-attention Pooler is the key that parameters of prompt generators can be shared across layers.

4 Experiments

In this section, we conduct a series of experiments and analysis to evaluate our IAPT method.

4.1 Baselines

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

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).

LoRA and its variants we consider the following LoRA variants as baselines: (a) the original LoRA (Hu et al., 2021); (b) AdaLoRA (Zhang et al., 2023b), which adaptively adjust the LoRA ranks among different Transformer modules.

Prompt-based tuning For prompt-based tuning methods, we compare with (a) P-tuning (Liu et al., 2021). (b) P-tuning v2 (Liu et al., 2021). (c) IDPG (Wu et al., 2022). (d) LPT (Liu et al., 2022b). We adjust the tunable parameters of these methods via reparameterization so that their tunable parameters are comparable to our IAPT methods.

Other PEFT methods We also compare: (a) BitFit (Ben-Zaken et al., 2021), which fine-tunes the model by adding tunable bias terms to the linear layers of LLMs. (b) $(\text{IA})^3$ (Liu et al., 2022a), which multiplies learnable vectors to the hidden states in different modules of the Transformer layer. (c) SSP (Hu et al., 2022), which is a representative work on combining different PEFT methods, including LoRA and BitFit.

The baselines are implemented using their open-sourced codes. We only adjust the hyper-parameters related to tunable parameter numbers to compare the baseline methods and our IAPT method fairly. The hyper-parameter settings for the baselines are detailed in Appendix F.

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., 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-

Method	Tunable Params	SST-2 (acc)	RTE (acc)	QNLI (acc)	BoolQ (acc)	COPA (acc)	ReCoRD (f1-em)	SQuAD (f1-em)
Baselines								
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
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
SSP	8.6M	93.5	82.6	92.6	86.4	91.1	90.0	87.4
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
Our proposed methods								
IAPT	8.4M	**94.3**	**83.9**	**93.2**	**87.5**	**91.9**	**91.2**	**88.5**

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 C.7.

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 C. The above tasks' evaluation metrics or protocols are in Appendix C.7.

4.3 Experiment Settings

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

Pretrained backbones The main experiments use the 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 the 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 IAPT framework In our experiments, unless otherwise specified, we set: (a) the length of soft prompts is $l_{sp} = 4$, (b) the bottleneck dimension m of the IAPT prompt generator is 256, (c) every $L_s = 4$ layers share the prompt generators' parameters except for the self-attention poolers, and (d) the hyper-parameters of the rational activation are $m = 6$, $n = 5$, and the learnable parameters a_j and b_i are initialized by approximating the GeLU activation function. Under the above settings, our IAPT method will introduce 8.4M tunable parameters to the LlaMA-2 7B backbone. The hyper-parameters for training are specified in Appendix F.

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 F.

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 IAPT method outperforms the baseline methods across all seven tasks, with comparable or fewer tunable parameters. In particular, IAPT outperforms the previous SOTA prompt tuning methods like P-tuning V2 and LPT and the strong LoRA style baselines like LoRA and AdaLoRA with comparable parameters. These results demonstrate that our method is good at downstream task adaptation of large language models.

Method	E2E (rouge-l)	GSM8K (acc)	WikiSQL (acc)
LPT	70.4	34.2	84.3
LoRA	70.7	35.1	85.4
AdaLoRA	70.8	35.2	85.2
IAPT	71.3	36.4	85.9

Table 2: Results for different PEFT methods on the E2E, GSM8K, and WikiSQL benchmark. The backbone LM is LlaMA-2 7B.

Method	Avg GPT-4 score (↑)	ROUGE-L (↑)
AdaLoRA	6.95	51.1
IAPT	7.19	52.8

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

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 IAPT method outperforms LoRA, AdaLoRA, and LPT on the three tasks.

Results for general-purpose instruction tuning After the LlaMA-2 7B is fine-tuned on the Alpaca dataset with our IAPT 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 C.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 IAPT method outperforms the AdaLoRA method in terms of the GPT-4 evaluation scores and ROUGE-L, demonstrating that IAPT can enhance the instruction tuning quality of large language models. A case study of answers generated by different methods is presented in Table 9 of Appendix M, showcasing that IAPT leads to better instruction-tuned LLMs.

4.5 Ablation studies and analysis

Analysis of the inference efficiency To demonstrate the inference efficiency of our IAPT method, we now compare the GPU memory and generation speed of IAPT and LoRA. 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 G. 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.

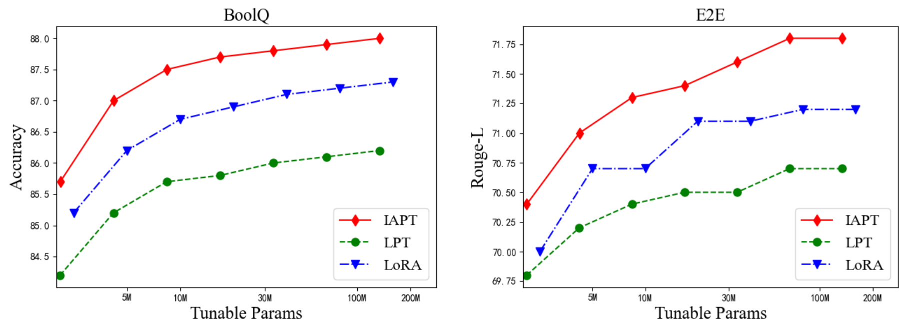
Method	Beam size	Speed (tps)	Memory cost (MiB)
LoRA	1	25.1	14616
	3	21.9	16104
IAPT	1	34.4	14490
	3	27.9	15946

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

From Table 4, one can see that under beam sizes 1 and 3, the IAPT method has a comparable memory cost with LoRA, but the generation speed is significantly higher. The speed advantages of IAPT come from the following factors: (a) our method only adds four soft tokens, which is relatively short compared to the instructions received by the modern LLMs (Ouyang et al., 2022). (b) our prompt generators are lightweight and efficient during inference. (c) The soft prompts are only generated once when an input instruction is passed to the LLM and right before generating the first new token. The soft prompts are integrated into the KV cache in the following generation steps. In contrast, the LoRA method requires the model to call the LoRA modules at each generation step, resulting in higher latency.

Visualization of the learned activation functions In Figure 4 of Appendix L, we visualize the learned activation functions of the prompt generators for every $L_s = 4$ Transformer layers after fine-tuning on the Alpaca dataset. Rational GeLU is the rational function approximating the GeLU activation and is used to initialize the learnable activation functions for the prompt generators. Rational GeLU and GeLU are overlapping with each other. As shown in Figure 4, we can see that (a) the learned activation function differs from the GeLU activation function but still has a similar shape to GeLU. (b) The learned activation functions are different across different Transformer layers. We can see that the learned activations adapted to the fine-tuning dataset and can extract suitable features for providing suitable soft prompts.

Ablation study of IAPT framework We now consider the following variants of IAPT: (a) IAPT-1 substitutes the self-attention pooler to average pooling. (b) IAPT-2 sets $L_s = 16$ and $m = 1024$. (c) IAPT-3 sets $L_s = 1$ and $m = 64$. (d) IAPT-4 uses



(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.

the GeLU activation function for every prompt generator. (e) IAPT-5 uses the ReLU activation function for every prompt generator. (f) IAPT-6 uses ReLU for the first 16 layers' prompt generators, and GeLU for the deeper 16 layers'. (h) IAPT-7 uses GeLU for the first 16 layers' prompt generators, and ReLU for the deeper 16 layers'. The experimental results on the BoolQ, E2E, and SQuAD tasks are reported in Table 7 of Appendix I. The results show that IAPT under the default settings (as in Table 1) outperforms the four variants. In addition, (a) comparing IAPT-1 to IAPT shows that the self-attention poolers provide more practical information aggregation. In addition, self-attention poolers provide adaptive feature extraction for cross-layer parameter sharing. (b) Comparing IAPT to IAPT-2 and IAPT-3 demonstrates that under the comparable tunable parameters, cross-layer parameter sharing of prompt generators allows for higher values of m, thus improving the capacity of IAPT. However, aggressively sharing prompt generators across layers could hurt downstream performance. (c) Comparing IAPT to IAPT-4, IAPT-5, IAPT-6 and IAPT-7 demonstrates the necessity of learning activation functions for the prompt generators.

Comparisons under different budgets of tunable parameters We vary the budget of tunable parameters for IAPT by modifying the values of $m = 256$ to $\{64, 128, 512, 1024, 2048, 4096\}$. We also vary the LPT 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 IAPT method can consistently outperform the LoRA and LPT methods.

Effects of different lengths of soft prompts We vary the length l_{sp} from 4 to $\{1, 2, 8, 16, 32\}$ for IAPT and LPT. The experimental results on the BoolQ task are presented in Figure 3 of Appendix J. The results show that our IAPT method is less sensitive to the prompt length in terms of downstream performance and performs better than the LPT baseline under different prompt lengths. IAPT effectively aggregates the semantic features of the input instructions with the help of the self-attention pooler and learnable activations, thus obtaining better downstream performances.

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 8. We can see that on these two backbones, our method can also outperform the baseline methods.

5 Conclusion

This work presents the instruction aware prompt tuning (IAPT) method, an innovative method for the parameter-efficient fine-tuning of large language models. Upon the hypothesis that different input instructions require different soft prompts, we propose to generate soft prompts from the input instructions. We propose three recipes for improving our framework's downstream performance: (a) self-attention pooling; (b) learning different activation functions during fine-tuning for different prompt generators of different depth; (c) cross-layer parameter sharing of prompt generators. Our method is convenient to implement and off-the-shelf. Experiments on various tasks demonstrate that our IAPT 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, 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 soft prompt based tuning in terms of better downstream performances whiling pursuing 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 tests 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.

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.

Nicolas Boull'e, Yuji Nakatsukasa, and Alex Townsend. 2020. Rational neural networks. *ArXiv*, abs/2004.01902.

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.

Quentin Delfosse, Patrick Schramowski, Alejandro Molina, and Kristian Kersting. 2021a. Recurrent rational networks. *arXiv preprint arXiv:2102.09407*.

Quentin Delfosse, Patrick Schramowski, Martin Mundt, Alejandro Molina, and Kristian Kersting. 2021b. Adaptive rational activations to boost deep reinforcement learning.

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, 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.

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.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*.

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*.

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

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.

Xiangyang Liu, Tianxiang Sun, Xuanjing Huang, and Xipeng Qiu. 2022b. 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. 2022c. 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*.

Alejandro Molina, Patrick Schramowski, and Kristian Kersting. 2019. Padé activation units: End-to-end learning of flexible activation functions in deep networks. *ArXiv*, abs/1907.06732.

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*.

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.

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*.

Yi Xin, Siqi Luo, Haodi Zhou, Junlong Du, Xiaohong Liu, Yue Fan, Qing Li, and Yuntao Du. 2024. Parameter-efficient fine-tuning for pre-trained vision models: A survey. *ArXiv*, abs/2402.02242.

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.

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. 2021a. Autonlu: Architecture search for sentence and cross-sentence attention modeling with redesigned search space. In *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part I 10*, pages 155–168. Springer.

Wei Zhu. 2021b. AutoRC: Improving BERT based relation classification models via architecture search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 33–43, Online. Association for Computational Linguistics.

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, Yuan Ni, and Guotong Xie. 2021. Autotrans: Automating transformer design via reinforced architecture search. In *Natural Language Processing and Chinese Computing*, pages 169–182, Cham. Springer International Publishing.

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 matrices across layers are also investigated by VERA.

B Appendix: introduction to bi-level optimization

The bi-level optimization (Liu et al., 2019) optimize Θ conditioned on the optimized parameters of Ω^*. Denote the training set as $\mathcal{D}_{train}$, and the validation set as $\mathcal{D}_{val}$. The inner and outer levels of optimization are conducted on these two separate splits of the task dataset, which is analogous to validating architectures trained on $\mathcal{D}_{train}$ using a different split $\mathcal{D}_{val}$ to avoid over-fitting. Thus the optimization objective is:

$$\begin{aligned} & \min_{\Theta} \mathcal{L}(\mathcal{D}_{val}, \Omega^*, \Theta), \\ & s.t. \ \ \Omega^* = \arg\min_{\Omega} \mathcal{L}(\mathcal{D}_{train}, \Omega, \Theta), \end{aligned} \quad (4)$$

where $\mathcal{L}()$ is the objective function on a given downstream task, such as cross entropy loss. The above bi-level optimization problem is approximated with an alternating optimization strategy. The gradients of Ω are calculated with batches of samples from $\mathcal{D}_{train}$, and the gradients of Θ are calculated on $\mathcal{D}_{val}$.

C Appendix for the datsets and evaluation metrics

C.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.

C.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.

C.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

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	-	rouge-l
GSM8K	7K	0.5K	1K	-	Math reasoning	-	acc
WikiSQL	61k	9K	17K	-	SQL generation	-	acc
Alpaca	50k	1k	-	-	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.

key contents. This dataset has a 42061/4672/4693 train/dev/test split.

C.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.

C.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.

C.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[5]. 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. We set aside 1000 samples in the Alpaca dataset as the validation set.

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

C.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[6] 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 IAPT'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 E. ROUGE-L scores computed by considering the answers generated by GPT-4 as the ground truth.

D 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> 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:

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

```
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]
```

E 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]
```

F 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 IAPT 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 $(\mathrm{IA})^3$, 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. For the bi-level optimizaiton of IAPT, the validation set is the same with the dev set. The hyper-parameters for calculating the gradients of the architectural parameters are the same with the normal training procedure, except that the learning rate is 1e-6. 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.

G Appendix: settings for efficiency analysis

In the Table 4 of the main contents, we conduct analysis on the IAPT 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.

H Appendix: pilot experiments

We now conduct pilot experiments on the BoolQ and E2E tasks to demonstrate the necessity of learning activation functions for the prompt generators. The other hyper-parameters or experimental settings are kept the same with Section 4.3 and F.

Method	BoolQ (acc)	E2E (rouge-l)	SQuAD (f1-em)
IAPT	**87.5**	**71.3**	**88.5**
IAPT-gelu	86.7	70.6	87.8
IAPT-relu	86.4	70.7	87.7
IAPT-relu-gelu	86.8	70.7	88.0
IAPT-gelu-relu	86.6	70.5	87.9

Table 6: The experimental results for the pilot experiments. The backbone model is LlaMA-2 7B.

We now compare three variants of IAPT: (a) IAPT-relu, which is to set the activation function of the prompt generators to ReLU. (b) IAPT-relu, which is to set the activation function of the prompt generators to GeLU. (c) IAPT-relu-gelu, which is to set the activation functions of the prompt generators on the lower 16 Transformer layers to ReLU, and set those on the 16 higher Transformer layers to GeLU. (d) IAPT-gelu-relu, which is to set the activation functions of the prompt generators on the lower 16 Transformer layers to GeLU, and set those on the 16 higher Transformer layers to ReLU. The results on the BoolQ and E2E tasks are presented in Table 6.

The results demonstrate that: (a) different downstream tasks may favor different activation functions for the prompt generators. (b) applying different activation functions for different Transformer layers may result in performance gains. The results demonstrate that there is room for improvements if we set the prompt generators' activation functions properly. However, such a setting is intractable to be set manually.

I Ablation on the IAPT framework

In the main contents, we consider seven variants of the IAPT method, and the experiments on the BoolQ, E2E and SQuAD tasks are provided in 7

J Ablation on the soft prompt length

We vary the prompt length l_{sp} from 4 to {1, 2, 8, 16, 32} for IAPT and LPT, and present the results on the BoolQ task in Figure 3.

K 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.

Method	BoolQ (acc)	E2E (rouge-l)	SQuAD (f1-em)
IAPT	**87.5**	**71.3**	**88.5**
IAPT-1	86.9	70.7	88.0
IAPT-2	86.2	70.2	87.3
IAPT-3	87.2	71.0	88.1
IAPT-4	86.7	70.6	87.8
IAPT-5	86.4	70.7	87.7
IAPT-6	86.8	70.7	88.0
IAPT-7	86.6	70.5	87.9

Table 7: The comparison of IAPT's variants on the BoolQ, E2E, and SQuAD tasks. The backbone model is LlaMA-2 7B.

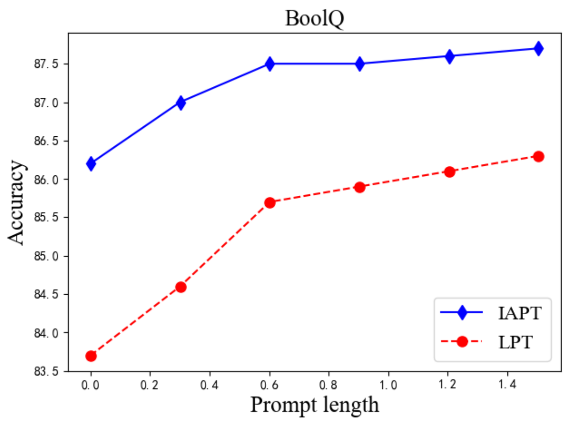


Figure 3: Performances under different soft prompt lengths.

Method	BoolQ (acc)	E2E (rouge-l)	SQuAD (f1-em)
Results for GPT-2 large			
LPT	67.2	66.1	43.2
LoRA	67.5	66.6	43.8
AdaLoRA	67.6	66.5	43.7
IAPT	**68.3**	**67.2**	**44.6**
Results for Pythia-1.4b large			
LPT	71.3	68.2	50.1
LoRA	71.6	68.4	50.4
AdaLoRA	71.5	68.5	50.4
IAPT	**72.3**	**69.1**	**51.3**

Table 8: Results for different PEFT methods on the BoolQ, E2E and SQuAD benchmarks. The backbone LMs are GPT2-large and Pythia-1.4b.

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 8.

L Visualization of the learned activation functions

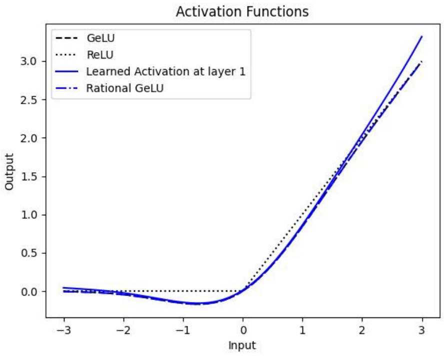
Now, we visualize the learned activation functions of the prompt generator at different Transformer layers in Figure 4.

M Case studies of Instruction tuning

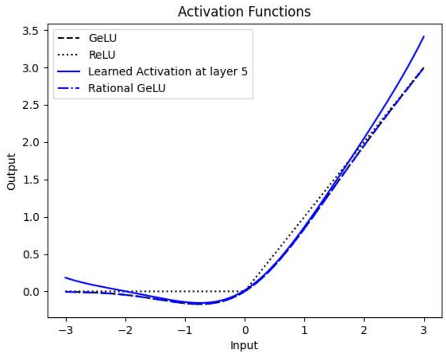
In the Section 4.4 of the main content, we present the overall performance of IAPT and AdaLoRA on the MT-Bench, after fine-tuning LlaMA-2 7B on the Alpaca dataset. Now we present concrete examples in Table 9 to showcase the Superiority of IAPT.

Query	One morning after sunrise, Suresh was standing facing a pole. The shadow of the pole fell exactly to his right. Can you tell me the direction towards which the shadow was pointing - east, south, west, or north? Explain your reasoning steps.
IAPT's response	
Responses	The shadow of the pole fell exactly to Suresh's right, which means that the shadow was pointing towards his left. Since the sun rises in the east and sets in the west, the shadow of the pole was pointing towards the west. Therefore, the direction towards which the shadow was pointing was west.
GPT-4's comments	Assistant 1 provides a clear and logical explanation. They correctly identify that since the shadow falls to the right of Suresh, it means that he is facing north, because the sun rises in the east and its light comes from that direction, casting shadows towards the west. The assistant's reasoning is accurate considering that sunrise has been mentioned, thus we can deduce that Suresh must be facing north for the shadow to fall on his right side (west). The explanation is easy to understand and would be helpful to human users looking for a clear answer with justification. However, there is an internal contradiction in Assistant 1's response; initially they state "the shadow was pointing towards his left" which does not align with their correct conclusion or premise. Score: 7/10 - While Assistant 1 reaches the correct conclusion (that the shadow points west), they mistakenly describe how shadows fall relative to Suresh's position first (claiming it falls towards his left), which could confuse users.
GPT-4 score	7
AdaLoRA's response	
Responses	The shadow of the pole fell exactly to Suresh's right, indicating that the sun was rising in the east.
GPT-4's comments	Assistant 2 starts well by indicating where Suresh must be looking based on where his shadow falls but fails to complete their reasoning. They point out correctly that if a pole's shadow falls exactly to someone's right after sunrise, it indicates that person must be facing north with sunlight coming from their left (east), casting a shadow toward their right (west). However, Assistant 2 does not provide this final step of information—identifying explicitly what direction this would make the shadow point—which makes their response incomplete when answering directly what was asked in terms of cardinal directions. Score: 5/10 - Assistant 2 identifies an important piece of information regarding how shadows are cast at sunrise but doesn't fully answer what direction toward which the shadow points nor do they offer any steps of reasoning beyond noting sun placement at sunrise. This may leave human users somewhat unclear about how they've reached any implicit conclusions or if additional reasoning should follow.
GPT-4 score	5

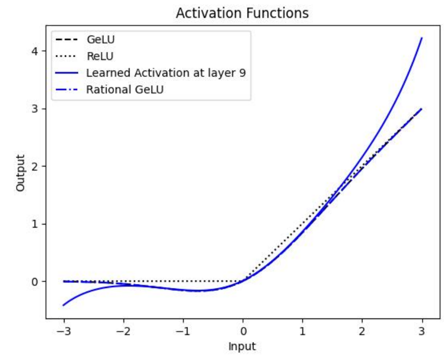
Table 9: An example showcasing that the fine-tuning quality of IAPT is higher than AdaLoRA.



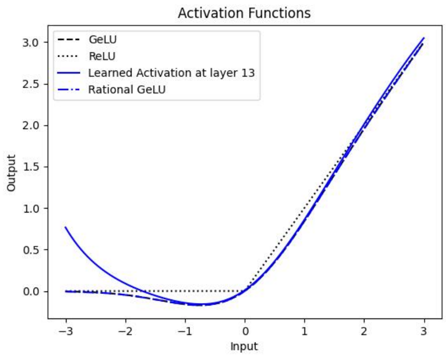
(a) 1st to 4-th layer



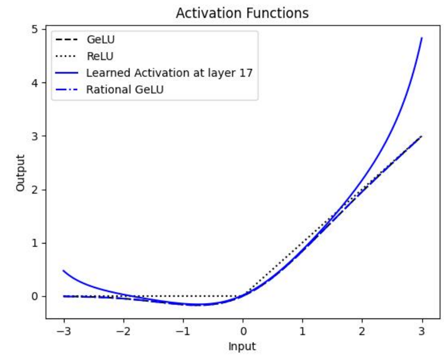
(b) 5th to 8-th layer



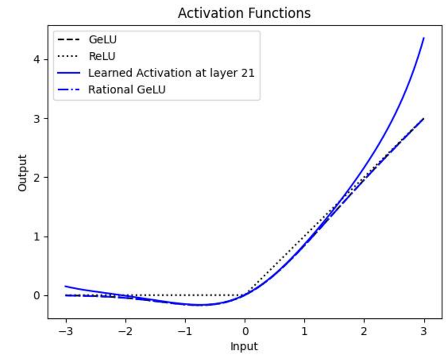
(c) 9th to 12-th layer



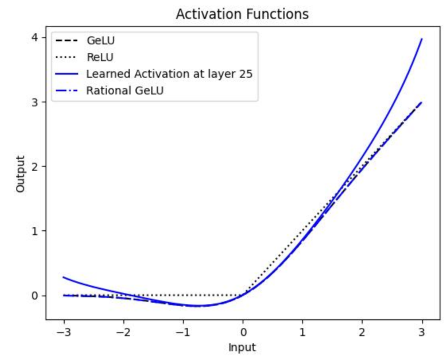
(d) 13th to 16-th layer



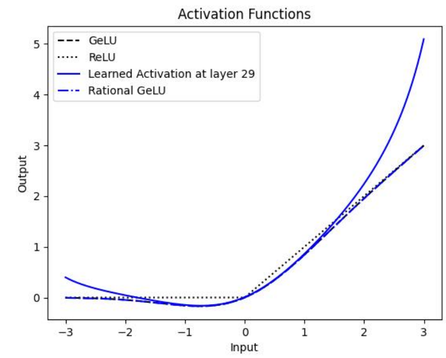
(e) 17th to 20-th layer



(f) 21th to 24-th layer



(g) 25th to 28-th layer



(h) 29th to 32-th layer

Figure 4: The learned activation functions for the prompt generators at different Transformer layers.