Late Prompt Tuning: A Late Prompt Could Be Better Than Many Prompts

Xiangyang Liu[1,2], **Tianxiang Sun**[1,2], **Xuanjing Huang**[1,2], **Xipeng Qiu**[1,2] *
[1]School of Computer Science, Fudan University
[2]Shanghai Key Laboratory of Intelligent Information Processing, Fudan University
{xiangyangliu20,txsun19,xjhuang,xpqiu}@fudan.edu.cn

Abstract

Prompt Tuning is a parameter-efficient tuning (PET) method for utilizing pre-trained models (PTMs), which simply prepends soft prompts to word embeddings and only optimizes the prompts to make PTMs adapt to downstream tasks. Although it is the ultimate in terms of parameter efficiency and more convenient for PTMs deployment, its performance is lower than most other state-of-the-art PET methods. Moreover, the number of tunable parameters is reduced by $\sim$17000$\times$ (from 355M to 21K on RoBERTa$_{\text{LARGE}}$), but it doesn't greatly reduce the training costs. In this paper, we explore why Prompt Tuning performs poorly and find that it is due to the long propagation path from label signals to the soft prompts. And we present **L**ate **P**rompt **T**uning (LPT) that can better and more efficiently drive PTMs using late prompt. Concretely, we insert soft prompts into some intermediate layer of PTMs instead of embedding layer or each layer. To take full advantage of the hidden states before the prompt insertion layer, we introduce a prompt generator to generate independent prompts for each instance using these hidden states. Through extensive experiment results across various tasks and PTMs, we show that LPT can achieve comparable performance to full model tuning in full-data scenario and outperforms the traditional Prompt Tuning by **12.4 points** and Model Tuning by **5 points** when only having 100 training samples. Besides, its training speed is **2 times** and memory cost is reduced by **56.6%** than Model Tuning.

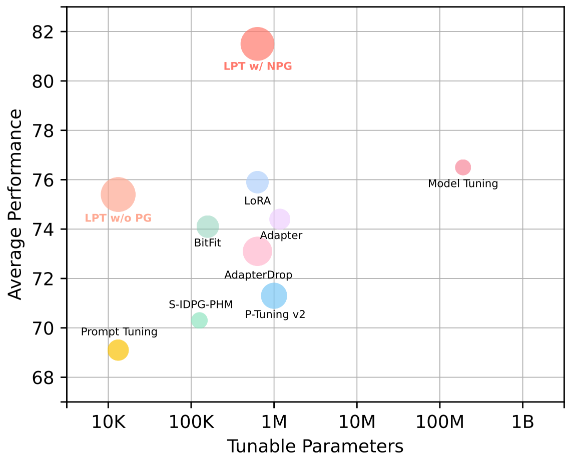


Figure 1: Overall comparison between LPT and baselines of only 100 training samples for each task. All methods are evaluated on 10 text classification tasks using RoBERTa$_{\text{LARGE}}$. The radius of every circle indicates training speed (tokens per millisecond). LPT w/ NPG and LPT w/o PG represent LPT with naive prompt generator and without prompt generator, respectively. The details can be found in Section 5.

1 Introduction

Pre-trained models (Devlin et al., 2019; Liu et al., 2019; Radford et al., 2019; Yang et al., 2019; Lan et al., 2020; Raffel et al., 2020; Lewis et al., 2020; He et al., 2021; Liu et al., 2022a; Qiu et al., 2020) have pushed most NLP tasks to state-of-the-art (SOTA). Model Tuning is a popular method for utilizing PTMs on downstream tasks, which needs to tune all parameters of PTMs for every task. Despite the welcome outcome, it leads to prohibitive adaptation costs, especially for supersized PTMs (Brown et al., 2020; Wei et al., 2022). Parameter-efficient tuning (PET) method is a new tuning paradigm which can adapt PTMs to downstream tasks by only tuning a very small number of internal or additional parameters.

Prompt Tuning (Lester et al., 2021) is a simple and popular PET method that prepends additional soft prompts to word embeddings and only optimizes the prompts to make PTMs adapt to downstream tasks. It has an absolute advantage in parameter efficiency, and facilitates mixed-task inference which makes the PTMs deployment convenient. However, compared with adapter-based tuning methods (Houlsby et al., 2019; Mahabadi et al., 2021) and some other PET methods like LoRA (Hu et al., 2022) and BitFit (Zaken et al.,

*Corresponding author.

2022), it has lower performance and convergence rate. In addition, compared with full model tuning, although the number of tunable parameters is reduced by $\sim$17,000$\times$ (from 355M to 21K on RoBERTa$_{\text{LARGE}}$), the training speed is only 1.5 times and memory cost is only reduced by 29.8% than full model tuning. P-tuning v2 (Liu et al., 2022b) alleviates the issues in terms of performance through inserting the prompts into each hidden layer of PTMs, but it is difficult to optimize and needs more steps and longer prompt length to train models.

In this paper, we explore why Prompt Tuning performs poorly and find that it is due to the long propagation path from label signals to the soft prompts inserted into word embeddings. The key to Prompt Tuning is to make the soft prompts carry task-related information through downstream training. And the more task-related information soft prompts carry, the better Prompt Tuning performs. However, the label information (task-related information) has lost too much when it arrives at soft prompts after layer upon layer propagation since the model is frozen. In addition, our pilot experiments show that inserting soft prompts to the intermediate layers of PTMs (i.e., late prompt) can easily tackle the issue. We refer the layer that prompts insert into as prompt layer. Another advantage of the late prompt is that there is no gradient calculation for model parameters below prompt layer. This can greatly improves training speed and reduce memory cost. But there is a trade-off between efficiency and performance where the performance first increases and then decreases with the rise of the prompt layer. Despite its higher performance and convergence rate than the traditional Prompt Tuning, the hidden states before the prompt layer are underutilized. To further improve performance and take full advantage of these contextual hidden states, we introduce a prompt generator to generate independent prompts for each instance using these hidden states (instance-aware prompt).

Based on the late and instance-aware prompt, we present LPT to improve Prompt Tuning. Since the soft prompts are inserted into one of the intermediate layers of PTMs, we don't need to compute gradients for model parameters below the prompt insertion layer. This can speed up training process and reduce memory cost. And extensive experiment results show that LPT outperforms most prompt-based tuning methods and can be comparable with adapter-based tuning methods even full model tuning. Especially in the few-shot scenario with only 100 training samples, LPT outperforms Prompt Tuning by **12.4 points** and Model Tuning by **5 points** in the average performance of ten text classification tasks while the training speed (tokens per millisecond) is $\sim$**2.0** times and the memory cost is decreased by $\sim$**56.6%** than full model tuning (on RoBERTa$_{\text{LARGE}}$). Figure 1 shows an overall comparison between LPT and baselines. To sum up, the key contributions of this paper are:

- We explore why Prompt Tuning performs poorly and find that it is due to the long propagation path from label signals to soft prompts and present a simple method referred as late prompt to tackle the issue.
- Combining the late prompt with instance-aware prompt, we present LPT, which can further improve performance and be comparable with adapter-based tuning methods and even Model Tuning while greatly reducing training costs.
- We verify the versatility of LPT in full-data and few-shot scenarios across 10 tasks and 3 PTMs.

2 Related Work

Adapter-based tuning. One research line of PET method is adapter-based tuning (Ding et al., 2022) which inserts some adapter modules between PTMs layers and optimizes these adapters in downstream training for model adaptation. Adapter (Houlsby et al., 2019) inserts adapter modules with bottleneck architecture between every consecutive Transformer (Vaswani et al., 2017) sub-layers. AdapterDrop (Rücklé et al., 2021) investigates the efficiency through removing adapters from lower layers. Compacter (Mahabadi et al., 2021) uses low-rank optimization and parameterized hypercomplex multiplication (Zhang et al., 2021) to compress adapters. Adapter-based tuning methods have comparable results with full model tuning when training data is sufficient but don't work well in few-shot scenario (Wang et al., 2021a).

Prompt-based tuning. Another main research line of PET method is prompt-based tuning, which inserts some additional soft prompts into the hidden states instead of injecting new neural modules to PTMs. Prompt Tuning (Lester et al., 2021) and P-tuning (Liu et al., 2021) 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., 2022b) insert prompts to every hidden layer of PTMs. BBT (Sun et al., 2022b) optimizes prompts with derivative-free optimization. Some prompt-based tuning methods, like Prompt Tuning and BBT, formulate downstream tasks as pre-training tasks (e.g., masked language modeling task) to close the gap between pre-training and downstream training (Sun et al., 2022a). There are also some prompt-based methods with instance-aware prompt. IDPG (Wu et al., 2022) uses prompt generator with parameterized hypercomplex multiplication (Zhang et al., 2021) to generate soft prompts for every instance. Context-tuning (Tang et al., 2022) uses BERT model (Devlin et al., 2019) as prompt generator and focuses on NLG tasks. IPL (Jin et al., 2022) firstly calculates relevance scores between prompt tokens and inputs and then uses the scores to re-weight the original prompt tokens. But, it tunes all parameters of PTMs. All above instance-aware prompt methods have the same weakness is to firstly encode inputs using an extra encoder, which slows down the training and increases inference latency.

There are also some other popular PET methods, such as BitFit (Zaken et al., 2022) which only tunes bias terms, LoRA (Hu et al., 2022) which optimizes low-rank decomposition matrices of the weights within self-attention layers.

3 Problem Formulation

Given a PTM $\mathcal{M}$, in the setting of Model Tuning, we first reformulate the input with single sentence as $\mathbf{E}(\texttt{[CLS]}\ \langle S_1 \rangle\ \texttt{[SEP]})$ and the input with sentence pair as $\mathbf{E}(\texttt{[CLS]}\ \langle S_1 \rangle\ \texttt{[SEP]}\ \langle S_2 \rangle\ \texttt{[SEP]})$. Where $\mathbf{E}$ is the embedding layer of $\mathcal{M}$. The final hidden state of `[CLS]` token will be used to predict label. In the setting of Prompt Tuning, we insert a set of randomly initialized soft prompts $\mathbf{p}$ into word embeddings, and also modify the original inputs using different manual templates with a `[MASK]` token for different tasks. For example, the input with single sentence from a sentiment analysis task will be transform into $\texttt{concat}(\mathbf{p}, \mathbf{E}(\texttt{[CLS]}\ \langle S_1 \rangle\ \text{It was}\ \texttt{[MASK]. [SEP]}))$. Then, we map the original labels $\mathcal{Y}$ to some words in the vocabulary $\mathcal{V}$ of $\mathcal{M}$, which formulates downstream tasks as a language modeling task to close the gap between pre-training and downstream training. The final hidden state of `[MASK]` token will be used to predict label.

For our proposed method LPT, the soft prompts **p** will be generated for every input using a prompt generator (**PG**). In addition, the layer that the soft prompts are inserted into is will be some intermediate layer of PTMs instead of word embeddings, and we refer the layer as prompt layer (**PL**).

4 Why Prompt Tuning Performs Poorly?

The workflow of Prompt Tuning can be thought of as making soft prompts to carry task-related information through downstream training. Then, during inference, these prompts are concatenated with test inputs, so that the hidden representations of these inputs also contain task-related information through layer upon layer propagation. An intuitive thought is that the more task-related information the soft prompts carry, the better its eventual generalization performance. And the task-related information is transmitted from the top layer of PTMs to the soft prompts inserted into the word embeddings. Based on the above facts, we speculate that the poor performance of prompt tuning is due to the long propagation path of task-related information, which leads to the information becoming weaker and weaker as it is transmitted layer by layer. To verify this conjecture, we conduct some pilot experiments on TREC (Voorhees and Tice, 2000) and RTE (Dagan et al., 2005) datasets using $\text{RoBERTa}_{\text{LARGE}}$ (Liu et al., 2019).

Does shortening the propagation distance improve performance? We start by considering a simple experiment setting where the soft prompts are inserted into different layers of $\text{RoBERTa}_{\text{LARGE}}$ and then we look at how performance changes as the prompt layer changes. As shown in the left plots of Figure 2, we can observe that the performance first increases and then decreases with the rise of prompt layer, and the highest performance is basically achieved when the prompt layer is in the range of 12 to 14. In addition, we also explore the convergence rates at different prompt layers. For simplification, we only consider three different prompt layers of 1, 13 and 24. The middle plots in Figure 2 show that when prompt layer is 13, the convergence rate is faster than prompt layer 1. When the prompt layer is extremely close to the output layer, the convergence rate will slow down. The trend shown in these plots can be preliminarily identified that shortening the propagation distance

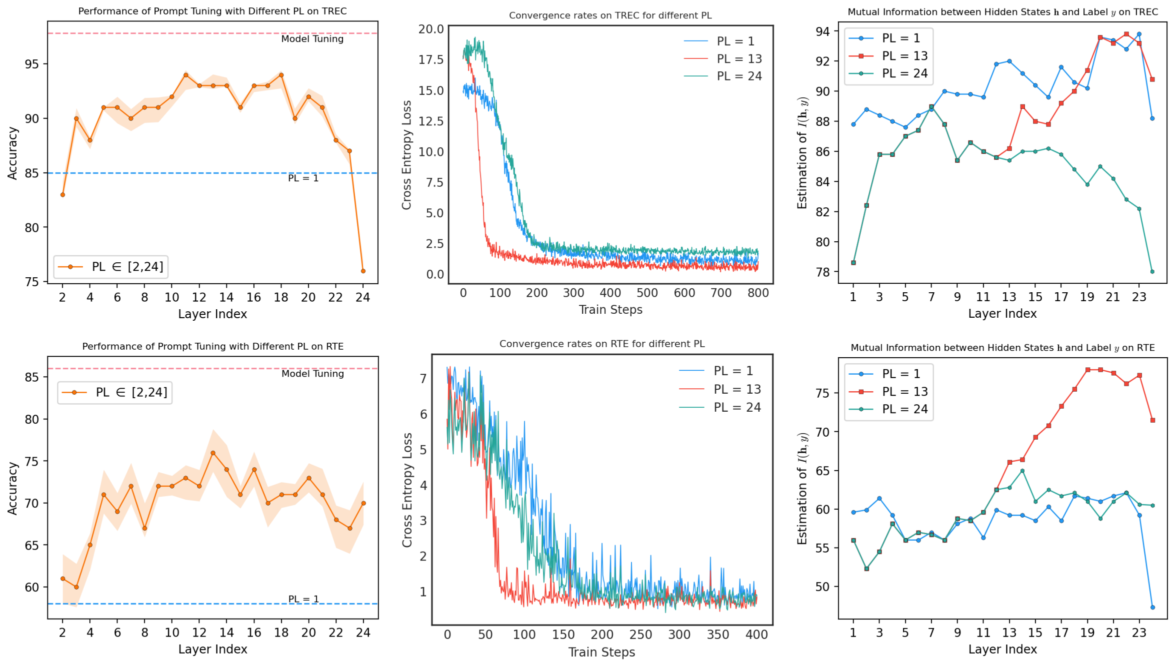


Figure 2: **Left**: The performance achieved by inserting soft prompts into different layers of RoBERTa$_{\text{LARGE}}$. **Middle**: Comparison of convergence rates for different prompt layers. **Right**: The estimated mutual information between hidden states of each layer and label. 'PL' denotes the prompt layer. 'PL = 1' denotes the traditional Prompt Tuning (Lester et al., 2021).

can effectively improve performance. However, when the prompt layer is closer to the output layer, the task-related representation information within the hidden states of inputs is far from enough due to the wrapped hidden states with soft prompts are only processed by a small part of the PTM. This leads to the gradual decline of performance when prompt layer is in the range of $[14, 24]$.

Task-related information in hidden states. To quantify the task-related information carried in soft prompts, we follow Wang et al. (2021b) and adopt the mutual information $I(\mathbf{h}, y)$ between hidden states and label of each input. The estimate method of $I(\mathbf{h}, y)$ is provided in Appendix A. For simplification, we follow the experiment setting of convergence rate. The right plots of Figure 2 show the $I(\mathbf{h}, y)$ at different layers. Observed by these plots, on RTE task, the values of $I(\mathbf{h}, y)$ are highest when PL = 13, which is consistent with the ranking of performance shown in the left plots. That is, the task-related information carried by the soft prompts of this method is more than others. However, on TREC task, the mutual information value is gradually increased to the maximum for PL = 13. We believe that is due to the interactions between soft prompts and hidden states of inputs is fewer at bottom layers, which leads to these hidden states have fewer task-related information.

The above observations suggest that our conjecture about the poor performance of Prompt Tuning is correct. The long propagation path of label information leads to the poor performance and the low convergence rate of Prompt Tuning.

5 LPT: Late Prompt Tuning

From the experiment results in Section 4, we observe that using late prompt can greatly improve the performance of Prompt Tuning. Moreover, late prompt can bring two other advantages: **(1)** No gradients calculation for model parameters below the prompt layer; **(2)** The hidden states before the prompt layer can be used to generate great independent prompts for each instance. Based on these, we propose an efficient prompt tuning method LPT which combines late and instance-aware prompt. An illustration of LPT is shown in Figure 3. In this section, we will introduce two different prompt generators used in LPT and how to determine the prompt layer.

5.1 Prompt Generators

Naive prompt generator (NPG). The prompt generator is a simple feed-forward layer with bottleneck architecture. Assume the prompt length is

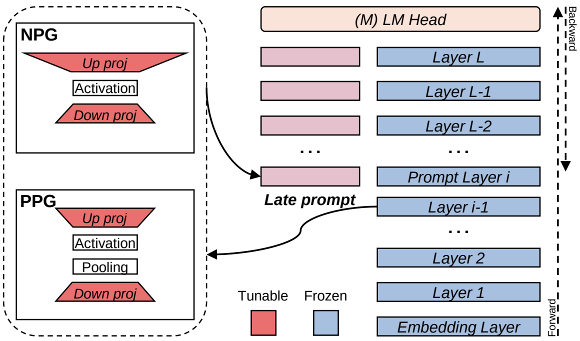
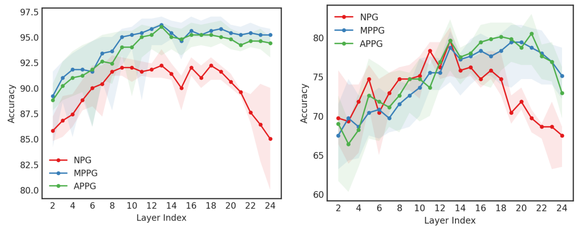


Figure 3: An illustration of LPT. **Left**: Naive (NPG) and pooling (PPG) prompt generators. **Right**: Prompt Tuning with late and instance-aware prompt.



(a) TREC (b) RTE

Figure 4: The change trend of performance with different prompt layer for three different prompt generators. The backbone model is RoBERTa$_{\text{LARGE}}$.

l, then we can generate independent prompts for each instance as below:

$$\hat{\mathbf{p}} = \mathbf{W_2}(\mathrm{ReLU}(\mathbf{W_1}\mathbf{h}_{\texttt{[CLS]}} + \mathbf{b_1})) + \mathbf{b_2}, \quad (1)$$

$$\mathbf{p} = \mathrm{Reshape}(\hat{\mathbf{p}}), \quad (2)$$

where $\mathbf{W_1} \in \mathbb{R}^{m\times d}$, $\mathbf{W_2} \in \mathbb{R}^{(l\times d)\times m}$, $\mathbf{h}_{\texttt{[CLS]}} \in \mathbb{R}^d$ and $\mathbf{p} \in \mathbb{R}^{l\times d}$. $\mathbf{b_1}$ and $\mathbf{b_2}$ are bias terms. d is the dimension of hidden states. Since $m \ll d$, the prompt generator doesn't have too many parameters. However, the number of parameters within $\mathbf{W_2}$ will increase with prompt length l. To ease this problem, we propose pooling prompt generator.

Pooling prompt generator (PPG). PPG introduces a pooling operation between down-projection and up-projection operations, which directly obtains the prompts with length l through pooling on input sequences. The generator is more lightweight to generate the prompt,

$$\hat{\mathbf{p}} = \mathrm{ReLU}(\mathrm{Pooling}(\mathbf{W_1}\mathbf{h} + \mathbf{b_1})), \quad (3)$$

$$\mathbf{p} = \mathbf{W_2}\hat{\mathbf{p}} + \mathbf{b_2}. \quad (4)$$

Different from NPG, $\mathbf{W_1} \in \mathbb{R}^{m\times d}$, $\mathbf{W_2} \in \mathbb{R}^{d\times m}$ and $\mathbf{h} \in \mathbb{R}^{d\times n}$ here. n is the length of the original input. In this paper, we consider both Average Pooling and Max Pooling, referred as **APPG** and **MPPG** respectively.

5.2 How to Determine Prompt Layer?

Generating a good prompt needs a good contextual representation for the input. In this sub-section, We will explore the trade-off between performance and efficiency of different prompt layers for LPT through the pilot experiments on TREC (Voorhees and Tice, 2000) and RTE (Dagan et al., 2005) datasets. As shown in Figure 4, the performance of NPG has a significant decline when the prompt layer is in the range from 14 to 24. However, different from NPG, APPG and MPPG retain high performance as the prompt layer approaches the output layer, especially on TREC dataset. We believe that this is due to the hidden states from the higher layers can help generate better prompts, while NPG only uses `[CLS]` token as the representation of the entire input when generating prompts, which leads to the loss of information. According to the above observations, LPT with APPG and MPPG can achieve a better trade-off for both relatively simple (TREC) and difficult (RTE) tasks. But in this work, to ensure that all methods (NPG, APPG and MPPG) can achieve a good performance while maintaining a relatively low training costs, we simply choose the most intermediate layer of PTMs as the prompt layer. That is, we choose the 13-th layer as the prompt layer for RoBERTa$_{\text{LARGE}}$.

6 Experiments

6.1 Evaluation Datasets

We evaluate our method on 5 single-sentence and 5 sentence-pair classification tasks, including 6 tasks from GLUE benchmark (Wang et al., 2019) and 4 other popular tasks include MPQA (Wiebe et al., 2005), MR (Pang and Lee, 2005), Subj (Pang and Lee, 2004) and TREC (Voorhees and Tice, 2000) tasks. All details about data statistics and splits can be found in Appendix B.

6.2 Experiment Settings

We evaluate our method in both full-data and few-shot scenarios on three PTMs, including RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019), DeBERTa$_{\text{LARGE}}$ (He et al., 2021) and GPT2$_{\text{LARGE}}$ (Radford et al., 2019). According to the conclusion from the Section 5.2, we choose the 13-th layer as the prompt layer

Method	Tunable Parameters	SST-2 (acc)	MPQA (acc)	MR (acc)	Subj (acc)	TREC (acc)	MNLI (acc)	MRPC (acc and F1)	QNLI (acc)	QQP (acc and F1)	RTE (acc)	Avg
Model Tuning	355M	95.6	**90.2**	91.3	96.8	**97.6**	89.3	**91.2**	94.6	**90.7**	**86.2**	**92.4**
Adapter	1.6M	**96.2** (0.2)	89.2 (0.5)	91.6 (0.4)	96.8 (0.4)	97.0 (0.3)	**90.5** (0.1)	90.3 (1.0)	94.7 (0.3)	89.4 (0.7)	85.5 (1.2)	92.3
AdapterDrop	811K	95.3 (0.3)	89.1 (0.7)	91.0 (0.5)	95.3 (0.6)	95.7 (0.5)	88.5 (0.2)	90.1 (1.3)	93.3 (0.3)	88.3 (0.3)	81.1 (2.0)	90.8
BitFit	273K	95.9 (0.1)	89.2 (0.9)	91.8 (0.5)	96.9 (0.1)	96.2 (0.3)	90.0 (0.1)	89.6 (0.9)	94.4 (0.2)	87.9 (0.4)	82.4 (1.1)	91.4
LoRA	788K	96.2 (0.3)	90.1 (0.3)	**92.0** (0.1)	**97.1** (0.4)	96.8 (0.6)	89.8 (0.3)	91.1 (0.6)	**94.8** (0.2)	89.8 (0.1)	84.8 (2.1)	92.3
Prompt Tuning	21K	94.9 (0.5)	88.8 (0.8)	89.6 (0.5)	93.9 (0.6)	86.4 (0.7)	86.7 (0.9)	75.7 (0.7)	91.4 (0.1)	81.2 (0.8)	60.8 (0.5)	84.9
Prompt Tuning-256	262K	95.8 (0.4)	90.2 (0.2)	91.8 (0.4)	95.8 (0.5)	93.3 (0.4)	87.7 (0.5)	76.2 (2.4)	91.6 (0.8)	85.3 (0.3)	59.7 (2.4)	86.7
P-tuning v2	985K	95.8 (0.4)	89.9 (0.6)	91.4 (0.4)	96.5 (0.2)	95.8 (0.6)	88.2 (0.2)	81.3 (1.1)	93.7 (0.3)	85.3 (0.2)	66.9 (2.3)	88.5
S-IDPG-PHM	114K	94.8 (0.3)	89.5 (0.6)	90.8 (0.5)	95.9 (0.6)	89.3 (0.4)	87.4 (0.5)	77.3 (1.2)	91.2 (0.4)	82.3 (1.9)	62.7 (1.9)	86.1
LPT												
LPT w/o PG	21K	95.5 (0.3)	87.6 (1.7)	89.3 (0.6)	95.1 (0.2)	89.7 (0.7)	88.0 (0.4)	82.3 (1.3)	92.0 (0.1)	84.2 (0.5)	75.2 (1.8)	87.9
LPT w/ NPG	792K	95.5 (0.4)	89.0 (0.1)	90.9 (0.2)	95.8 (0.2)	95.9 (0.4)	87.0 (0.3)	88.4 (1.5)	91.7 (0.6)	86.6 (0.5)	79.7 (3.2)	90.1
LPT w/ MPPG	263K	95.4 (0.4)	89.1 (0.2)	90.7 (0.1)	96.5 (0.2)	97.4 (0.2)	87.7 (0.3)	90.4 (0.6)	91.3 (0.3)	88.6 (0.4)	78.7 (3.3)	90.6
LPT w/ APPG	263K	95.3 (0.2)	89.1 (0.3)	90.7 (0.1)	96.2 (0.2)	97.0 (0.2)	87.4 (0.3)	90.2 (1.0)	91.6 (0.4)	87.4 (0.4)	79.2 (3.3)	90.4

Table 1: Overall comparison in full-data scenario. All the methods are evaluated on test sets except the tasks from GLUE benchmark. We report mean and standard deviation of performance over 3 different random seeds for all the methods except model tuning. The best results are highlighted in **bold** and the second best results are marked with underline. Prompt Tuning-256 indicates the Prompt Tuning method with prompt length 256. All the results are obtained using RoBERTa$_{\text{LARGE}}$.

for RoBERTa$_{\text{LARGE}}$ and DeBERTa$_{\text{LARGE}}$, and the 19-th layer for GPT2$_{\text{LARGE}}$ except special explanation. More implementation details are provided in Appendix C.

6.3 Baselines

We consider *Model Tuning*, *adapter-based tuning*, *prompt-based tuning* methods and two other state-of-the-art PET methods which include **(1) BitFit** (Zaken et al., 2022) and **(2) LoRA** (Hu et al., 2022) as our baselines. For adapter-based tuning methods, we compare with **(1) Adapter** (Houlsby et al., 2019) and **(2) AdapterDrop** (Rücklé et al., 2021). For prompt-based tuning methods, we compare with **(1) Prompt Tuning** (Lester et al., 2021), **(2) P-tuning v2** (Liu et al., 2022b) and **(3) IDPG** (Wu et al., 2022). We implement Aadpter, AdapterDrop, BitFit and LoRA using OpenDelta[1] library. For IDPG which also raises instance-aware prompt, we only compare with the version with single-layer prompt, that is S-IDPG-PHM. And we don't use supplementary training like Wu et al. (2022) to enhance performance.

6.4 Main Results

Results in full-data scenario. The overall comparison of the results in full-data scenario is shown in Table 1. We can observe that: (i) Prompt Tuning method with only late prompt, that is LPT w/o PG can greatly improve the performance of the traditional Prompt Tuning under the same tunable parameters and even is comparable with P-tuning v2 which insert prompts to each layer of PTMs. (ii) Increasing prompt length for Prompt Tuning can improve performance to some extend, but increasing the training burden and inference latency notably. (iii) Our methods LPT with different prompt generators (i.e., LPT w/ NPG, LPT w/ MPPG and LPT w/ APPG) outperform all the prompt-based methods include S-IDPG-PHM which also claims instance-aware prompt. (iv) The performance of LPT with prompt generator is comparable with AdapterDrop, especially for LPT w/ MPPG and LPT w/ APPG which the number of their tunable parameters is only one-third of that of AdapterDrop. (v) Prompt-based methods underperform adapter-based methods on sentence-pair tasks compared with adapter-based methods and Model Tuning, which is consistent with the results from Sun et al. (2022b) and Ding et al. (2022). We suppose that is due to sentence-pair tasks are more difficult than single-sentence tasks and are more influenced by manual templates and label words.

Results in few-shot scenario We further evaluate our method in few-shot scenario. Following Wu et al. (2022), we consider two settings where the number of training data is 100 and 500 respectively and randomly sample the training samples from original training sets. Besides, we randomly sample 1000 samples from the original training sets as development sets and there is no overlap with sampled training sets. For the tasks from GLUE benchmark (Wang et al., 2019), the original development sets are used as the test sets and the test sets remain unchanged for 4 other tasks.

Table 2 and 9 show the overall comparison of all the methods in few-shot scenario. LPT w/ NPG

[1]https://github.com/thunlp/OpenDelta

Method	Tunable Parameters	SST-2 (acc)	MPQA (acc)	MR (acc)	Subj (acc)	TREC (acc)	MNLI (acc)	MRPC (acc and F1)	QNLI (acc)	QQP (acc and F1)	RTE (acc)	Avg
Model Tuning	355M	89.6 (1.2)	81.5 (2.0)	85.5 (2.5)	**93.6** (0.5)	<u>91.3</u> (1.9)	51.5 (3.3)	78.3 (1.0)	<u>73.9</u> (6.6)	71.6 (2.9)	48.6 (3.0)	76.5
Adapter	1.6M	90.8 (1.3)	81.8 (3.0)	86.3 (1.5)	93.4 (0.8)	89.7 (3.7)	42.9 (1.2)	77.9 (2.4)	61.5 (2.7)	67.3 (2.0)	52.0 (2.4)	74.4
AdapterDrop	811K	87.8 (1.2)	81.4 (2.3)	85.8 (1.5)	<u>93.5</u> (0.9)	89.8 (4.6)	41.0 (0.8)	76.6 (0.9)	60.4 (3.9)	64.7 (2.5)	50.4 (1.8)	73.1
BitFit	273K	<u>91.8</u> (0.9)	84.0 (2.1)	86.9 (1.0)	92.3 (1.0)	90.8 (1.8)	42.0 (0.9)	77.0 (2.7)	60.3 (6.5)	64.9 (0.9)	50.8 (2.2)	74.1
LoRA	788K	91.0 (1.3)	83.2 (1.3)	87.4 (0.7)	92.6 (1.4)	**92.0** (0.4)	48.1 (3.7)	<u>78.5</u> (1.7)	65.9 (5.7)	69.7 (3.3)	51.0 (2.0)	75.9
Prompt Tuning	21K	90.0 (2.2)	73.5 (5.8)	85.1 (1.2)	80.6 (3.8)	72.3 (4.7)	47.3 (1.8)	74.2 (1.0)	55.8 (1.7)	52.7 (2.1)	59.6 (2.3)	69.1
P-tuning v2	985K	89.4 (0.6)	80.6 (1.6)	84.6 (2.3)	91.7 (1.4)	84.9 (4.5)	37.3 (1.8)	75.0 (0.6)	54.2 (1.1)	60.7 (3.0)	54.9 (2.1)	71.3
S-IDPG-PHM	114K	90.5 (1.7)	75.5 (5.8)	85.8 (0.8)	81.6 (1.8)	75.3 (3.7)	47.8 (1.6)	75.2 (1.1)	56.9 (0.9)	54.5 (1.9)	59.8 (2.4)	70.3
LPT												
LPT w/o PG	21K	91.3 (1.0)	80.6 (7.3)	88.2 (0.5)	90.7 (0.9)	79.9 (1.5)	52.9 (5.5)	77.4 (1.0)	66.2 (3.0)	68.2 (4.0)	58.3 (2.9)	75.4
LPT w/ NPG	792K	**92.7** (0.8)	**86.8** (1.4)	**88.5** (0.5)	92.7 (0.5)	90.9 (2.5)	**64.3** (2.0)	**80.6** (2.0)	**75.7** (3.2)	**74.6** (1.9)	**68.1** (5.5)	**81.5**
LPT w/ MPPG	263K	90.2 (0.9)	83.9 (5.0)	<u>88.5</u> (0.6)	92.7 (0.9)	85.9 (5.3)	58.8 (1.6)	77.3 (1.5)	71.9 (2.9)	<u>72.8</u> (2.3)	63.0 (3.4)	78.5
LPT w/ APPG	263K	90.4 (0.7)	<u>84.4</u> (6.1)	88.3 (0.6)	92.6 (1.2)	87.9 (3.7)	<u>60.1</u> (2.4)	78.2 (2.6)	71.6 (4.1)	72.0 (2.0)	<u>64.0</u> (2.9)	<u>79.0</u>

Table 2: Results in the few-shot scenario of 100 training samples. We report mean and standard deviation of performance over 4 different data splits for all the methods. **Bold** and <u>Underline</u> indicate the best and the second best results. All the results are obtained using RoBERTa$_{\text{LARGE}}$.

outperforms all the baselines in two different settings. Especially when the training set has only 100 samples, it outperforms Model Tuning by 5 points and Adapter by 7.1 points. This indicates that our method has better generalization when the training data is very scarce. However, we observe that LPT w/ MPPG and LPT w/ APPG don't perform as well in the few-shot scenario as they do in the full-data scenario. We speculate that this is owing to the optimal state of the pooling layer is to retain only useful information, and sufficient training data is needed to achieve this state. Nevertheless, both them are also superior to all the baselines when the training set has 100 samples.

Results on other PTMs To verify the generality of our conclusion about why Prompt Tuning performs poorly and the versatility of the proposed LPT, we also conduct experiments on two other popular PTMs, DeBERTa$_{\text{LARGE}}$ (He et al., 2021) and GPT2$_{\text{LARGE}}$ (Radford et al., 2019). The results are shown in Table 3. Obviously, only using late prompt to shorten the propagation distance of label information (i.e., LPT w/o PG) is also far superior to the traditional Prompt Tuning method on these two PTMs. This result enhances the reliability of our conclusion. Moreover, LPT with prompt generators further improve the performance on this basis, closing the gap with Model Tuning.

Method	Tunable Parameters	Subj (acc)	TREC (acc)	MRPC (acc and F1)	RTE (acc)	Avg
DeBERTa$_{\text{LARGE}}$						
Model Tuning	406M	**97.4**	**97.4**	**91.2**	**87.5**	**93.4**
Prompt Tuning	21K	94.2 (0.5)	87.7 (2.0)	79.8 (1.6)	64.6 (3.7)	81.6
LPT w/o PG	21K	94.9 (0.5)	94.4 (0.3)	81.4 (1.2)	75.1 (1.9)	86.5
LPT w/ NPG	792K	96.5 (0.2)	96.3 (0.3)	<u>90.8</u> (0.8)	<u>84.4</u> (0.7)	<u>92.0</u>
LPT w/ MPPG	263K	<u>96.9</u> (0.2)	<u>97.3</u> (0.3)	89.6 (1.0)	81.1 (1.6)	91.2
LPT w/ APPG	263K	96.5 (0.2)	97.0 (0.2)	89.7 (1.2)	82.6 (1.3)	91.5
GPT2$_{\text{LARGE}}$						
Model Tuning	774M	**97.2**	**97.0**	**88.0**	**75.8**	**89.5**
Prompt Tuning	26K	88.8 (1.0)	82.7 (1.1)	75.1 (0.5)	53.7 (1.3)	75.1
LPT w/o PG	26K	94.9 (1.2)	93.7 (2.3)	77.3 (1.3)	57.8 (2.1)	80.9
LPT w/ NPG	990K	<u>96.0</u> (0.3)	96.1 (0.4)	82.9 (1.0)	69.9 (1.0)	86.2
LPT w/ MPPG	329K	95.9 (0.3)	96.3 (0.5)	85.6 (0.4)	71.6 (0.6)	87.4
LPT w/ APPG	329K	95.6 (0.3)	<u>96.7</u> (0.3)	<u>85.7</u> (0.2)	<u>72.9</u> (0.8)	<u>87.7</u>

Table 3: Results on two single-sentence and two sentence-pair tasks using DeBERTa$_{\text{LARGE}}$ and GPT2$_{\text{LARGE}}$ models as the backbone. **Bold** and <u>Underline</u> indicate the best and the second best results.

6.5 Efficiency Evaluation

We compare the efficiency of our method with all the baselines on RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019) and GPT2$_{\text{LARGE}}$ (Radford et al., 2019) models. For each backbone, we select the largest batch size such that Model Tuning method can fit the fixed budget of a NVIDIA GTX 3090 GPU (24GB) and other methods use the same batch size as Model Tuning. We set the length of all inputs to 256 and evaluate the accuracy in the few-shot scenario that the number of training data is 100 for all methods.

In Table 4, we report accuracy, tuable parameters, training speed (tokens per millisecond) and memory cost (GB) of each method. Our methods outperform all prompt-based methods considered in terms of efficiency and memory cost, and achieves the highest performance. Compared with AdapterDrop which has similar efficiency, our method LPT w/ NPG outperforms it by 20.1 and 7 points on RoBERTa$_{\text{LARGE}}$ and GPT2$_{\text{LARGE}}$, respectively. In addition, we also explore the impact of the choice of prompt layer on all efficiency metrics, and the specific experiment results are in Appendix E. Overall, given the large scale PTMs with millions or billions of parameters, such as RoBERTa (Liu et al., 2019), DeBERTa (He et al., 2021) and GPT2 (Radford et al., 2019), higher training speed and lower memory cost is

Method	Accuracy	Tuable Parameters	Training Speed tokens/ms (↑)	Memory Cost GB (↓)
		RoBERTa$_{LARGE}$		
Model Tuning	52.0 (1.9)	355M	11.6	23.5
Adapter	50.3 (2.5)	1.6M	15.5 (1.3×)	16.5 (29.8%)
AdapterDrop	49.4 (3.4)	811K	21.6 (1.9×)	9.5 (59.6%)
BitFit	50.2 (1.8)	273K	16.5 (1.4×)	15.7 (33.2%)
LoRA	50.1 (2.7)	788K	16.4 (1.4×)	16.2 (31.1%)
Prompt Tuning	58.2 (1.7)	21K	16.9 (1.5×)	17.8 (24.3%)
P-tuning v2	53.2 (2.4)	985K	19.2 (1.7×)	16.8 (28.5%)
S-IDPG-PHM	58.8 (1.9)	114K	12.0 (1.0×)	16.8 (28.5%)
LPT w/ NPG	**69.5** (3.1)	792K	23.2 (2.0×)	10.1 (56.6%)
LPT w/ MPPG	62.4 (3.1)	263K	23.4 (2.0×)	10.6 (54.9%)
LPT w/ APPG	<u>63.0</u> (2.2)	263K	23.4 (2.0×)	10.6 (54.9%)
		GPT2$_{LARGE}$		
Model Tuning	50.0 (1.9)	774M	2.6	22.1
Adapter	52.8 (2.9)	3.0M	3.3 (1.3×)	11.8 (46.6%)
AdapterDrop	49.9 (0.9)	1.5M	6.0 (2.3×)	8.4 (62.0%)
BitFit	51.3 (2.4)	511K	4.3 (1.7×)	11.5 (48.0%)
LoRA	52.6 (1.9)	740K	4.1 (1.6×)	11.5 (47.1%)
Prompt Tuning	50.3 (1.2)	26K	4.4 (1.7×)	13.6 (38.5%)
P-tuning v2	49.7 (1.9)	1.9M	4.5 (1.7×)	13.0 (41.2%)
S-IDPG-PHM	52.1 (2.3)	171K	3.2 (1.2×)	12.7 (42.5%)
LPT w/ NPG	**56.9** (2.0)	990K	6.0 (2.3×)	9.4 (57.5%)
LPT w/ MPPG	<u>54.2</u> (2.6)	329K	6.2 (2.4×)	9.6 (56.6%)
LPT w/ APPG	53.6 (1.7)	329K	6.2 (2.4×)	9.6 (56.6%)

Table 4: Comparison of parameter efficiency, training efficiency and memory cost for all the methods on two different backbone models. All methods are evaluated on RTE dataset.

a paramount importance for practical applications. And LPT offers a better trade-off in terms of training budget and performance.

6.6 Analyses

Effect of prompt layer. To enhance the reliability of the conclusion from Section 5.2, that is the most intermediate layer is the optimal choice of prompt layer, we also conduct the same experiment on DeBERTa$_{\text{LARGE}}$ (He et al., 2021) and GPT2$_{\text{LARGE}}$ (Radford et al., 2019) models. As shown in Figure 5, the most intermediate layer is also the optimal choice of prompt layer on DeBERTa$_{\text{LARGE}}$ and GPT2$_{\text{LARGE}}$ models, especially for LPT w/ NPG. These results suggest that optimal results can be achieved by selecting the most intermediate layer while maintaining relatively low training costs.

Visualization of instance-aware prompt. We select Subj dataset (Pang and Lee, 2004) which its development set has 1000 samples for this analysis. For simplification, we only visualize the instance-aware prompt of LPT w/ NPG method. As shown in Figure 6, we mark the samples which their representations are close using the same color. We can clearly observe that our method can generate similar prompts for samples with relatively similar sentence representation. On the contrary, for the

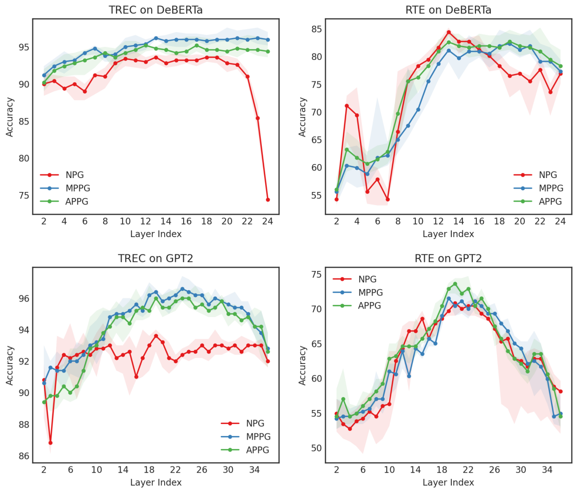


Figure 5: Change trend of performance with different prompt layers on DeBERTa$_{\text{LARGE}}$ (upper) and GPT2$_{\text{LARGE}}$ (lower) models.

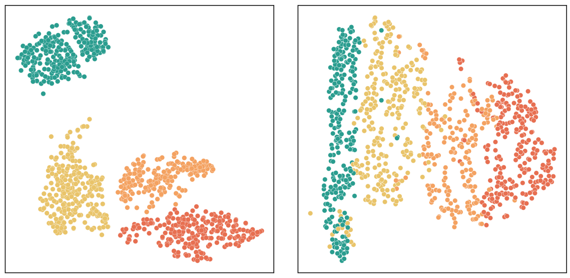


Figure 6: Sentence representation visualization (left) and instance-aware prompt visualization (right).

samples which have quite different sentence representations, their independent prompts are also quite different. The visualization result indicates that our method learns a special prompt for each instance and can be aware of the important information of the instance to drive PTMs better.

7 Conclusion

In this paper, we find that Prompt Tuning performs poorly due to the long propagation path between label information and soft prompts, and verify the reliability of the conclusion through extensive experiments. With this discovery, we present a more efficient and effective prompt tuning method LPT with late and instance-aware prompt. Experiment results in full-data and few-shot scenarios demonstrate the proposed LPT can achieve comparable or even better performance than state-of-the-art PET methods and full model tuning while having higher training speed and lower memory cost.

Limitations

Although we showed that our proposed method can greatly improve performance and reduce training costs for diverse NLU tasks on three different PTMs (i.e., $\text{RoBERTa}_{\text{LARGE}}$, $\text{DeBERTa}_{\text{LARGE}}$ and $\text{GPT2}_{\text{LARGE}}$), the larger PTMs with billions of or more parameters and NLG tasks were not considered. But our main thought of using late and instance-aware prompt is simple and can be easily transferred to other backbone architectures and different types of tasks. It would be interesting to investigate if our findings hold for other backbone models and types of tasks. And we will explore it in future work.

Ethics Statement

The finding and proposed method aims to improve the traditional Prompt Tuning method in terms of training costs and performance. The used datasets are widely used in previous work and, to our knowledge, do not have any attached privacy or ethical issues.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, volume 3944 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

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

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 3816–3830. Association for Computational Linguistics.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Feihu Jin, Jinliang Lu, Jiajun Zhang, and Chengqing Zong. 2022. Instance-aware prompt learning for language understanding and generation. *CoRR*, abs/2201.07126.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations,* *ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3045–3059. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

Xiangyang Liu, Tianxiang Sun, Junliang He, Lingling Wu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2022a. Towards efficient NLP: A standard evaluation and A strong baseline. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2022, Seattle, Washington, July 10-15, 2022*. Association for Computational Linguistics.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022b. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *CoRR*, abs/2103.10385.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 1022–1035.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain*, pages 271–278. ACL.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*, pages 115–124. The Association for Computer Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *SCIENCE CHINA Technological Sciences*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

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, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. Adapterdrop: On the efficiency of adapters in transformers. In *Proceedings of the*

2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, pages 7930–7946. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Tianxiang Sun, Xiangyang Liu, Xipeng Qiu, and Xuan-Jing Huang. 2022a. Paradigm shift in natural language processing. *Int. J. Autom. Comput.*, 19(3):169–183.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. *CoRR*, abs/2201.03514.

Tianyi Tang, Junyi Li, and Wayne Xin Zhao. 2022. Context-tuning: Learning contextualized prompts for natural language generation. *CoRR*, abs/2201.08670.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Ellen M. Voorhees and Dawn M. Tice. 2000. Building a question answering test collection. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 200–207. ACM.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2021a. List: Lite self-training makes efficient few-shot learners. *CoRR*, abs/2110.06274.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. 2021b. Revisiting locally supervised learning: an alternative to end-to-end training. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *10th International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Lang. Resour. Evaluation*, 39(2-3):165–210.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

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. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. 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. *CoRR*, abs/2204.04497.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1–9. Association for Computational Linguistics.

Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. 2021. Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

A Details for Mutual Information Estimation

Because the mutual information cannot be be calculated directly, we estimate it by training a new classifier using hidden states $\mathbf{h}$ as inputs and the original labels of inputs as outputs. Then, we estimate $I(\mathbf{h}, y)$ using the performance achieved by the classifier. Since $I(\mathbf{h}, y) = H(y) - H(y|\mathbf{h}) = H(y) - \mathbb{E}_{(\mathbf{h},y)}[-\log p(y|\mathbf{h})]$ (Wang et al., 2021b), we can train a new classifier $q_\psi(y|\mathbf{h})$ to approximate $p(y|\mathbf{h})$, such that we have $I(\mathbf{h}, y) \approx \max_\psi\{H(y) - \frac{1}{N}[\sum_{i=1}^{N} - \log\ q_\psi(y_i|\mathbf{h}_i)]\}$. Because $H(y)$ is a constant, we are going to ignore it here. Based on the above conditions, we can use the loss of $q_\psi(y|\mathbf{h})$ (i.e., $-\frac{1}{N}[\sum_{i=1}^{N} - \log\ q_\psi(y_i|\mathbf{h}_i)]$) as the estimate of $I(\mathbf{h}, y)$. Further simplification, we use the performance of this new classifier to estimate mutual information $I(\mathbf{h}, y)$. Because RoBERTa$_{\text{LARGE}}$ (Liu et al., 2019) has 24 layers totally except embedding layer, we can obtain 24 hidden states for each input. Hence, we need to train 24 new classifiers for each method. To speed up the training process, we use a 6-layer RoBERTa$_{\text{LARGE}}$ as q_ψ.

B Datasets

For SST-2 (Socher et al., 2013), MNLI (Williams et al., 2018), MRPC (Dolan and Brockett, 2005), QNLI (Rajpurkar et al., 2016), QQP[2] and RTE (Dagan et al., 2005) datasets which are from GLUE benchmark (Wang et al., 2019), we use their original data splits. For 4 other datasets, we select a certain number of samples from the training set as the development set, and the number of samples for each label is determined according to its proportion in the original training set. The dataset statistics after split are shown in Table 5

C Implementation Details

The search space of hyperparameters considered in our method is shown in Table 6. As an additional note, because the hyperparameter settings are different for each baseline model, we don't listed them all. For adapter-based tuning methods, we set the down-projection size m to 16. We set the prompt length to 20 for Prompt Tuning (Lester et al., 2021) and P-tuning v2 (Liu et al., 2022b), and 5 for S-IDPG-PHM (Wu et al., 2022) and LPT w/ NPG. For LPT w/ MPPG and LPT w/ APPG, due to the number of tunable parameters being invariable with prompt length changes, we also search the prompt length in the range of {10, 15, 20} for them. Besides, we set the down-projection size m of S-IDPG-PHM and LPT to 256 and 128 respectively. The hyperparameters r and α in LoRA are set to 8 and 16 on RoBERTa$_{\text{LARGE}}$ and 4 and 32 on GPT2$_{\text{LARGE}}$. For the batch size of GPT2 model listed in Table 6, it refers to the number of samples in a single forward pass. Due to the large scale of GPT2$_{\text{LARGE}}$, we use *gradient accumulation* technique. The accumulation step is 2 or 4. We use AdamW optimizer (Loshchilov and Hutter, 2019) for all the methods referred in this work. We use Pytorch (Paszke et al., 2019) and HuggingFace's Transformers (Wolf et al., 2020) libraries to implement all the methods referred in this work. All experiments are conducted on 8 NVIDIA GTX 3090 GPUs.

We follow Gao et al. (2021) and show the manual templates and label words in our experiments in Table 7 and Table 8. Note that, since the vocabulary of the GPT2 model doesn't have the `[MASK]` token, we justly use it to represent the positions that are needed to predict.

D Results in Few-shot Scenario

We also evaluate all methods in the few-shot scenario of 500 training samples. The results are shown in Table 9, our method LPT w/ NPG also outperforms all baselines which is consistent with the scenario of 100 training samples. And we observe that the gap between both LPT w/ PPG and LPT w/ NPG narrows when training data increases.

E Efficiency Evaluation on Different Prompt Layers.

We select prompt layer in the range of $\{7, 13, 19\}$ to explore the influence from different prompt layers for the trade-off in terms of training budget and performance. The experiment settings are consistent with those described in Section 6.5. Table 10 shows the performance, the number of tunable parameters, training speed and memory cost for LPT with three different prompt layers. When prompt layer is the 13-th layer, both performance and training efficiency are better than when it is the 7-th layer. When the prompt layer is the 19-th layer, the efficiency is improved while the performance also degrades a lot.

[2] https://www.quora.com/q/quoradata/

Category	Datasets	|Train|	|Dev|	|Test|	$\|\mathcal{Y}\|$	Type	Labels
Single-sentence	SST-2	67349	872	1821	2	sentiment	positive, negative
	MPQA	7606	1000	2000	2	opinion polarity	positive, negative
	MR	7662	1000	2000	2	sentiment	positive, negative
	Subj	7000	1000	2000	2	subjectivity	subjective, objective
	Trec	4952	500	500	6	question cls.	abbr., entity, description, human, loc., num.
Sentence-pair	MNLI	392702	19647	19643	3	NLI	entailment, neutral, contradiction
	MRPC	3668	408	1725	2	paraphrase	equivalent, not equivalent
	QNLI	104743	5463	5463	2	NLI	entailment, not entailment
	QQP	363846	40430	390965	2	paraphrase	equivalent, not equivalent
	RTE	2490	277	3000	2	NLI	entailment, not entailment

Table 5: The statistics of datasets evaluated in this work. For MNLI task, the number of samples in development and test sets is summed by matched and mismatched samples. $|\mathcal{Y}|$ is the number for classes.

Hyperparameter	RoBERTa		DeBERTa		GPT2	
	Full-data	Few-shot	Full-data	Few-shot	Full-data	Few-shot
#Layers	24	24	24	24	36	36
Hidden size	1024	1024	1024	1024	1280	1280
Dropout rate	0.1	0.1	0.1	0.1	0.1	0.1
Peak learning rate	5e-4−1e-2	5e-4−1e-2	5e-4−1e-2	5e-4−1e-2	5e-4−1e-2	5e-4−1e-2
Warmup type	linearly decayed	linearly decayed	linearly decayed	linearly decayed	linearly decayed	linearly decayed
Warmup rate	{0, 0.06}	{0, 0.06}	{0, 0.06}	{0, 0.06}	{0, 0.06}	{0, 0.06}
Batch size	{16, 32}	{8, 16, 32}	{16, 32}	{8, 16, 32}	{8, 16}	{4, 8, 16}
Weight decay	0.1	0.1	0.1	0.1	0.1	0.1
Training step	–	1000	–	1000	–	1000
Training epoch	10	–	10	–	10	–
AdamW β_1	0.9	0.9	0.9	0.9	0.9	0.9
AdamW β_2	0.999	0.999	0.999	0.999	0.999	0.999
AdamW ϵ	1e-8	1e-8	1e-8	1e-8	1e-8	1e-8

Table 6: Search space for each hyperparameter considered in our method.

Task	Template	Label words
SST-2	$\langle S_1 \rangle$ It was `[MASK]` .	positive: great, negative: terrible
MPQA	$\langle S_1 \rangle$ It was `[MASK]` .	positive: great, negative: terrible
MR	$\langle S_1 \rangle$ It was `[MASK]` .	positive: great, negative: terrible
Subj	$\langle S_1 \rangle$ It was `[MASK]` .	subjective: subjective, objective: objective
TREC	`[MASK]` : $\langle S_1 \rangle$	abbreviation: Expression, entity: Entity, description: Description human: Human, location: Location, numeric: Number
MNLI	$\langle S_1 \rangle$? `[MASK]` , $\langle S_2 \rangle$	entailment: Yes, netural: Maybe, contradiction: No
MRPC	$\langle S_1 \rangle$ `[MASK]` , $\langle S_2 \rangle$	equivalent: Yes, not equivalent: No
QNLI	$\langle S_1 \rangle$? `[MASK]` , $\langle S_2 \rangle$	entailment: Yes, not entailment: No
QQP	$\langle S_1 \rangle$ `[MASK]` , $\langle S_2 \rangle$	equivalent: Yes, not equivalent: No
RTE	$\langle S_1 \rangle$? `[MASK]` , $\langle S_2 \rangle$	entailment: Yes, not entailment: No

Table 7: Manual templates and label words used on RoBERTa and DeBERTa models.

Task	Template	Label words
Subj	$\langle S_1 \rangle$ It was `[MASK]` .	subjective: subjective, objective: objective
TREC	`[MASK]` : $\langle S_1 \rangle$	abbreviation: Expression, entity: Entity, description: Description human: Human, location: Location, numeric: Number
MRPC	$\langle S_1 \rangle$ $\langle S_2 \rangle$ They are `[MASK]` .	equivalent: Yes, not equivalent: No
RTE	$\langle S_1 \rangle$ $\langle S_2 \rangle$ They are `[MASK]` .	entailment: Yes, not entailment: No

Table 8: Manual templates and label words used on GPT2 model.

Method	Tunable Parameters	SST-2 (acc)	MPQA (acc)	MR (acc)	Subj (acc)	TREC (acc)	MNLI (acc)	MRPC (acc and F1)	QNLI (acc)	QQP (acc and F1)	RTE (acc)	Avg
Model Tuning	355M	91.4 (0.8)	87.2 (1.1)	<u>89.4</u> (0.6)	**95.1** (0.4)	95.4 (0.5)	<u>75.3</u> (2.1)	**85.1** (1.8)	**85.2** (0.9)	<u>77.3</u> (1.2)	67.0 (7.7)	<u>84.8</u>
Adapter	1.6M	92.0 (1.0)	86.5 (1.5)	88.4 (1.0)	<u>95.1</u> (0.4)	95.0 (0.4)	70.5 (4.8)	83.6 (1.1)	78.0 (1.8)	72.1 (6.7)	67.5 (6.7)	82.9
AdapterDrop	811K	91.2 (1.0)	84.4 (1.2)	88.4 (0.8)	95.1 (0.4)	**95.7** (0.4)	66.1 (4.5)	82.5 (1.6)	78.9 (1.0)	73.4 (0.6)	62.0 (3.2)	81.6
BitFiT	273K	<u>92.2</u> (1.0)	<u>87.6</u> (0.9)	89.0 (0.8)	94.7 (0.2)	95.0 (0.6)	73.0 (2.8)	83.5 (0.6)	80.4 (1.4)	75.6 (1.4)	59.0 (1.8)	82.7
LoRA	788K	92.1 (1.1)	87.5 (0.7)	88.6 (1.4)	95.1 (0.2)	<u>95.5</u> (0.9)	74.5 (2.9)	<u>84.1</u> (0.6)	82.5 (1.5)	76.4 (1.1)	62.8 (3.2)	83.9
Prompt Tuning	21K	91.1 (1.5)	74.7 (5.1)	88.3 (0.6)	86.4 (0.4)	81.7 (2.4)	45.5 (1.5)	74.6 (0.3)	58.1 (1.6)	52.6 (5.8)	61.2 (1.7)	71.4
P-tuning v2	985K	91.3 (0.3)	85.1 (1.6)	88.0 (1.5)	94.5 (0.4)	94.6 (0.8)	61.6 (2.7)	76.6 (1.8)	73.7 (2.4)	71.7 (1.8)	56.0 (1.1)	79.3
S-IDPG-PHM	114K	91.3 (0.5)	75.9 (3.8)	88.7 (0.4)	87.2 (0.6)	84.7 (2.1)	46.3 (1.1)	75.1 (0.8)	59.4 (0.7)	56.4 (3.0)	64.7 (1.7)	73.0
LPT												
LPT w/o PG	21K	91.9 (0.4)	83.6 (1.0)	88.7 (0.6)	92.5 (0.7)	84.2 (0.8)	54.5 (5.8)	80.0 (0.8)	75.3 (2.2)	73.1 (1.9)	64.8 (3.1)	78.9
LPT w/ NPG	792K	**92.6** (0.4)	**87.8** (0.5)	**90.0** (0.4)	94.9 (0.2)	93.5 (0.4)	**76.0** (1.0)	81.4 (0.9)	<u>83.2</u> (1.3)	**77.9** (0.8)	**74.7** (2.7)	**85.2**
LPT w/ MPPG	263K	91.0 (0.8)	86.3 (1.0)	89.3 (0.3)	94.6 (0.3)	93.2 (0.9)	70.9 (3.5)	82.5 (0.6)	78.1 (2.0)	75.1 (1.1)	69.0 (3.3)	83.0
LPT w/ APPG	263K	91.9 (0.3)	86.2 (0.9)	89.0 (0.3)	94.3 (0.2)	92.5 (1.2)	69.2 (3.5)	82.2 (1.3)	79.4 (1.9)	74.8 (1.3)	<u>70.4</u> (1.4)	83.0

Table 9: Results in the few-shot scenario of 500 training samples. We report mean and standard deviation of performance over 4 different data splits for all the methods. **Bold** and <u>Underline</u> indicate the best and the second best results. All the results are obtained using RoBERTa$_{\text{LARGE}}$.

Method	Accuracy	Tunable Parameters	Training Speed tokens/ms (↑)	Memory Cost GB (↓)
Model Tuning	52.0 (1.9)	355M	11.6	23.5
Prompt Tuning	58.2 (1.7)	21K	16.9 (1.5×)	17.8 (24.3%)
LPT w/ NPG				
PL = 7	<u>63.2</u> (3.3)	792K	18.5 (1.6×)	13.4 (43.0%)
PL = 13	**69.5** (3.1)	792K	23.2 (2.0×)	10.1 (56.6%)
PL = 19	62.6 (3.3)	792K	28.5 (2.5×)	6.7 (71.5%)
LPT w/ MPPG				
PL = 7	59.9 (4.4)	263K	19.8 (1.7×)	14.3 (39.1%)
PL = 13	62.4 (3.1)	263K	23.4 (2.0×)	10.6 (54.9%)
PL = 19	58.8 (1.5)	263K	28.8 (2.5×)	7.0 (70.2%)
LPT w/ APPG				
PL = 7	58.6 (2.3)	263K	19.8 (1.7×)	14.3 (39.1%)
PL = 13	63.0 (2.2)	263K	23.4 (2.0×)	10.6 (54.9%)
PL = 19	60.1 (2.2)	263K	28.8 (2.5×)	7.0 (70.2%)

Table 10: Trade-off between performance and training efficiency. 'PL' denotes the prompt layer. **Bold** and <u>Underline</u> marks the best and the second best results, respectively. All methods are evaluated on RTE dataset using RoBERTa$_{\text{LARGE}}$ model.