IDPG: An Instance-Dependent Prompt Generation Method

Zhuofeng Wu[1*] **Sinong Wang**[2] **Jiatao Gu**[2] **Rui Hou**[2]
Yuxiao Dong[2,3†] **V.G.Vinod Vydiswaran**[4,1] **Hao Ma**[2]
[1]School of Information, University of Michigan
[2]Meta AI [3]Tsinghua University
[4]Department of Learning Health Sciences, University of Michigan
`{zhuofeng, vgvinodv}@umich.edu`
`{sinongwang, jgu, rayhou, haom}@fb.com`
`yuxiaod@tsinghua.edu.cn`

Abstract

Prompt tuning is a new, efficient NLP transfer learning paradigm that adds a task-specific prompt in each input instance during the model training stage. It freezes the pre-trained language model and only optimizes a few task-specific prompts. In this paper, we propose a conditional prompt generation method to generate prompts for each input instance, referred to as the Instance-Dependent Prompt Generation (IDPG). Unlike traditional prompt tuning methods that use a fixed prompt, IDPG introduces a lightweight and trainable component to generate prompts based on each input sentence. Extensive experiments on ten natural language understanding (NLU) tasks show that the proposed strategy consistently outperforms various prompt tuning baselines and is on par with other efficient transfer learning methods such as Compacter while tuning far fewer model parameters.[1]

1 Introduction

In recent years, pre-training a transformer model on a large corpus with language modeling tasks and fine-tuning it on different downstream tasks has become the primary transfer learning paradigm in natural language processing (Devlin et al., 2019). Notably, this paradigm requires updating and storing all the model parameters for each downstream task. As the model size proliferates (e.g., 330M parameters for BERT (Devlin et al., 2019) and 175B for GPT-3 (Brown et al., 2020)), it becomes computationally expensive and challenging to fine-tune the entire pre-trained language model (LM). Thus, it is natural to ask whether we can transfer the knowledge of a pre-trained LM to downstream tasks by keeping most of the parameters fixed and tuning only a small fraction of them.

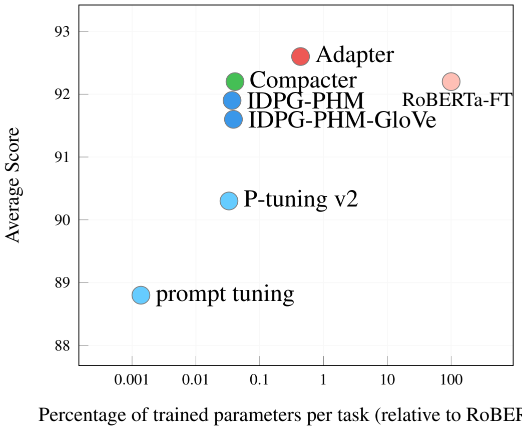


Figure 1: Overall evaluation of competing approaches on ten NLU tasks, with parameters from classification heads excluded. Our method approaches RoBERTa-FT's performance and uses fewer parameters than Adapter-based methods.

Previous studies have attempted to address this question from different perspectives. One line of research (Li and Liang, 2021) suggests augmenting the model with smaller, trainable modules and freezing the original transformer weights. Adapters (Houlsby et al., 2019; Pfeiffer et al., 2021, 2020), for example, insert a small set of additional modules between each transformer layer. Only these additional and task-specific modules are trained during fine-tuning, reducing the number of trainable parameters to $\sim$ 1–3% of the original transformer model per task. Compacter (Mahabadi et al., 2021) optimizes the training parameters further by designing a lightweight module to replace the bottleneck architecture in Adapters.

Another line of work focuses on prompting. The GPT-3 models (Brown et al., 2020; Schick and Schütze, 2021) find that, with proper manual prompts, a pre-trained LM can successfully match the fine-tuning performance of BERT models. LM-BFF (Gao et al., 2021a), EFL (Wang et al., 2021), and AutoPrompt (Shin et al., 2020) extend this direction by inserting prompts in the input embedding

[*] Work partially done while interning at Meta AI.
[†] Work done when at Meta AI.
[1]Our code is publicly available at `https://github.com/CSerxy/IDPG`.

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 5507 - 5521
July 10-15, 2022 ©2022 Association for Computational Linguistics

layer. However, these methods rely on grid-search for a natural language-based prompt from an ample search space, leading to optimization challenges.

To tackle this issue, prompt tuning (Lester et al., 2021), prefix tuning (Li and Liang, 2021), and P-tuning (Liu et al., 2021a,b) approaches propose to prepend trainable prefix tokens to the input layer and train these soft prompts only during the fine-tuning stage. In doing so, the problem of searching discrete prompts is converted to a continuous optimization task, which can be solved by a variety of optimization techniques such as SGD. This significantly reduced the number of trainable parameters to just a few thousand. However, all existing prompt-tuning methods have thus far focused on task-specific prompts, which are inadequate to address the gap between pre-training and fine-tuning objectives. Specifically, it is unlikely to see many different sentences with the same prefix in the pre-training corpus. Thus, a unified prompt may disturb the prediction and lead to a performance drop. In light of these limitations, we instead ask the following question: *Can we generate input-dependent prompts to smooth the domain difference?*

This paper presents the instance-dependent prompt generation (IDPG) strategy for efficiently tuning large-scale LMs. Unlike traditional prompt-tuning methods that rely on a fixed prompt for each task, IDPG instead develops a conditional prompt generation model to generate prompts for each instance. Formally, the IDPG generator can be denoted as $f(x;\mathbf{W})$, where x is the instance representation and $\mathbf{W}$ represents the trainable parameters. Note that by setting $\mathbf{W}$ to a zero matrix and only training the bias, IDPG would degenerate into the traditional prompt tuning process (Lester et al., 2021). To further reduce the number of parameters in the generator $f(x;\mathbf{W})$, we propose to apply a lightweight bottleneck architecture (i.e., a two-layer perceptron) and then decompose it by a parameterized hypercomplex multiplication (PHM) layer (Zhang et al., 2021). To summarize, this work makes the following contributions:

- We introduce an input-dependent prompt generation method—IDPG—that only requires training 134K parameters per task, corresponding to $\sim$0.04% of a pre-trained LM such as RoBERTa-Large (Liu et al., 2019).
- Extensive evaluations on ten natural language understanding (NLU) tasks show that IDPG consistently outperforms task-specific prompt tuning methods by 1.6–3.1 points. Additionally, it offers comparable performance to Adapter-based methods while using fewer parameters.
- We conduct substantial intrinsic studies, revealing how and why each component of the proposed model and the generated prompts could help the downstream tasks.

2 Preliminary

2.1 Manual Prompt

Manual prompt learning (Brown et al., 2020; Schick and Schütze, 2021) inserts a pre-defined label words in each input sentence. For example, it reformulates a sentence sentiment classification task with an input sentence S_1 as

$$x_{in} = \texttt{[CLS]}\,P\,\texttt{[SEP]}\,S_1\,\texttt{[EOS]},$$

where P is the prompt such as “indicating the positive user sentiment”. Using the pre-trained language model $\mathbf{M}$, we can obtain the sentence representation $\mathbf{h}_{\texttt{[CLS]}} = \mathbf{M}(x_{in})$, and train a task-specific head $\texttt{softmax}(\mathbf{W}\mathbf{h}_{\texttt{[CLS]}})$ to maximize the log-probability of the correct label. LM-BFF (Gao et al., 2021a) shows that adding a specifically designed prompt during fine-tuning can benefit the few-shot scenario. EFL (Wang et al., 2021) further suggests that reformulating the task as entailment can further improve the performance in both low-resource and high-resource scenarios.

2.2 Prompt Tuning

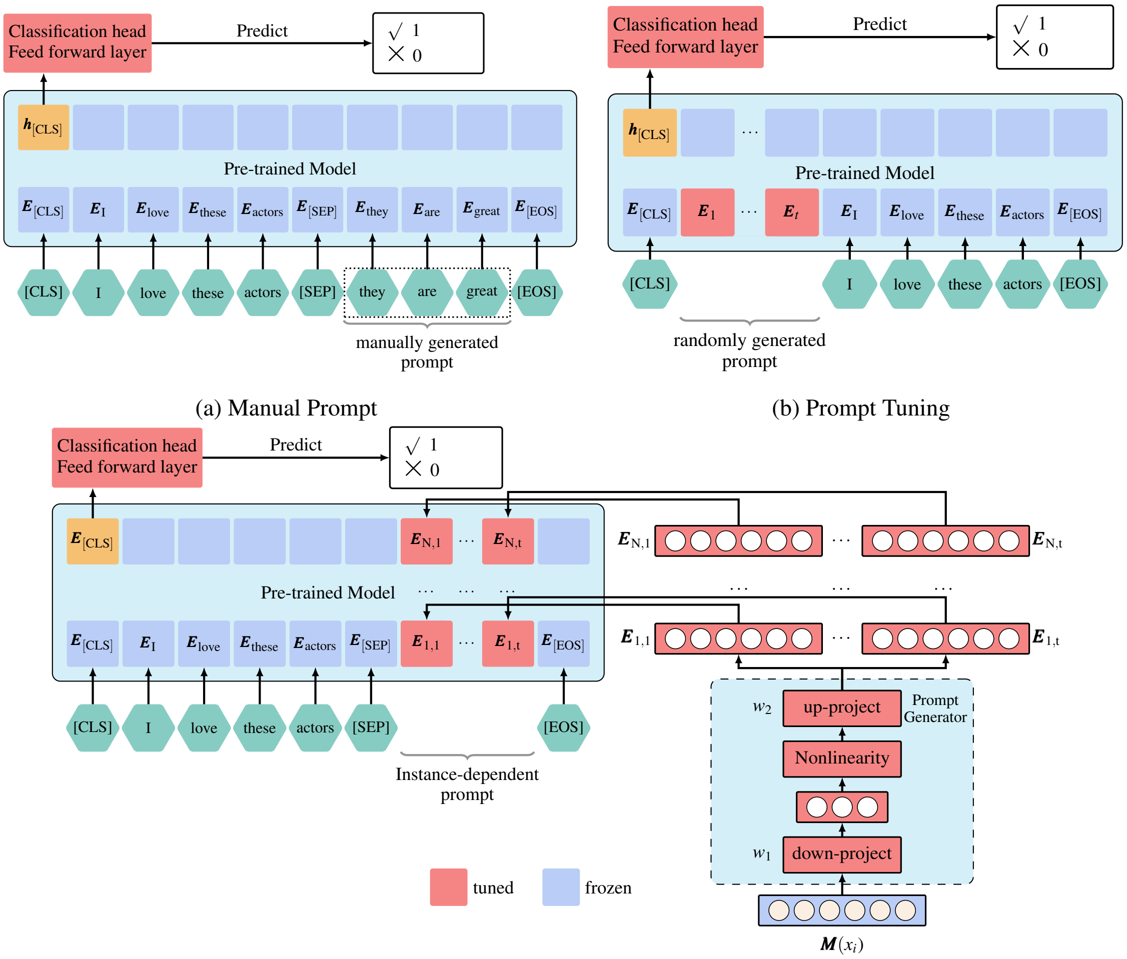
Prompt tuning (Lester et al., 2021), prefix tuning (Li and Liang, 2021), and P-tuning (Liu et al., 2021a,b) methods propose to insert a trainable prefix in front of the input sequence. Specifically, they reformulate the input for single sentence tasks as

$$x_{in} = \texttt{concat}[\mathbf{W}_p, \mathbf{E}(\texttt{[SEP]}\,S_2\,\texttt{[EOS]})]$$

and for sentence pair tasks as

$$x_{in} = \texttt{concat}[\mathbf{W}_p, \mathbf{E}(\texttt{[SEP]}\,S_2\,\texttt{[SEP]}\,S_3\,\texttt{[EOS]})],$$

where $\mathbf{W}_p$ is the embedding table of the inserted prompt, S_2 and S_3 are input sentences, and $\mathbf{E}$ denotes the operation of tokenization and extraction of embeddings. Apart from LM-BFF and EFL, there is no corresponding real text for the prompt as $\mathbf{W}_p$ is a set of random-initialized tensors to represent the soft prompt.


Classification head
Feed forward layer
Predict
√ 1
× 0
$h_{[CLS]}$
Pre-trained Model
$E_{[CLS]}$ E_I E_{love} E_{these} E_{actors} $E_{[SEP]}$ E_{they} E_{are} E_{great} $E_{[EOS]}$
[CLS] I love these actors [SEP] they are great [EOS]
manually generated prompt
(a) Manual Prompt
Classification head
Feed forward layer
Predict
√ 1
× 0
$h_{[CLS]}$
Pre-trained Model
$E_{[CLS]}$ E_1 ⋯ E_t E_I E_{love} E_{these} E_{actors} $E_{[EOS]}$
[CLS] I love these actors [EOS]
randomly generated prompt
(b) Prompt Tuning
Classification head
Feed forward layer
Predict
√ 1
× 0
$E_{[CLS]}$ $E_{N,1}$ ⋯ $E_{N,t}$
Pre-trained Model
$E_{[CLS]}$ E_I E_{love} E_{these} E_{actors} $E_{[SEP]}$ $E_{1,1}$ ⋯ $E_{1,t}$ $E_{[EOS]}$
[CLS] I love these actors [SEP] [EOS]
Instance-dependent prompt
$E_{N,1}$ $E_{N,t}$ $E_{1,1}$ $E_{1,t}$
w_2 up-project
Prompt Generator
Nonlinearity
w_1 down-project
$M(x_i)$
tuned
frozen

(c) Instance-Dependent Prompt Generation

Figure 2: An illustration of (a) manual prompt; (b) prompt-tuning method; (c) our proposed method. The red block refers to the trainable module, while the blue block refers to the frozen module.

3 Instance-Dependent Prompt Generation (IDPG)

We now introduce our proposed method, IDPG, along with various model optimizations. The main procedure is illustrated in Figure 2.

3.1 Instance-Dependent Generation

Let us assume a task T with training data $D_{train} = \{(x_i, y_i)\}_{i=1}^{K}$. Following prompt tuning, we define the input $x_i = \mathbf{E}(\texttt{[SEP]}S_1\texttt{[SEP]}S_2\texttt{[EOS]})$ for sentence-pair task or $x_i = \mathbf{E}(\texttt{[SEP]}S_1\texttt{[EOS]})$ for single-sentence task, where $\mathbf{E}(\cdot)$ is the token embedding for input sentences. Different from all previous works that only define a task-specific prompt $\mathbf{W}_p(T) \in \mathbb{R}^{d \times t}$, where t is the number of tokens in prompt representation and d is the hidden dimension, we propose a instance-dependent prompt generation method. Specifically, we suppose that the generation of prompt should not only depend on the task T, but also be affected by input sequence x_i. If $\mathbf{M}(x_i) \in \mathbb{R}^d$ is a representation of the input sequence x_i from same pre-trained LM $\mathbf{M}$, we design a lightweight model $\mathbf{G}$ to generate the prompt,

$$\mathbf{W}_p(T, x_i) = \mathbf{G}(\mathbf{M}(x_i), T),\ x_i \in D_{train} \tag{1}$$

Then, we insert a prompt $\mathbf{W}_p(T)$ together with input sequence x_i to infer y_i during fine-tuning. In this way, we have a unified template

$$\texttt{softmax}(\mathbf{W}\mathbf{h}_{\texttt{[CLS]}}) \tag{2}$$

$$\mathbf{h}_{\texttt{[CLS]}} = \mathbf{M}(\texttt{concat}[x_i, \mathbf{W}_p(T, x_i)]) \tag{3}$$

where $\mathbf{W}$ is the trainable LM classification head.

To reduce the number of trainable parameters in $\mathbf{G}$, we apply a lightweight bottleneck architecture (i.e., a two-layer perceptron) for generation. As illustrated in Figure 2 (c), the generator $\mathbf{G}$ first projects the original d-dimensional sentence representation $\mathbf{h}_i$ into m dimensions. After passing through a nonlinear function, generator $\mathbf{G}$ projects the hidden representation back to a d dimensions with t timestamps. The total number of parameters for generator $\mathbf{G}$ is $m(d+1)+td(m+1)$ (bias term included). This model can be regarded as the general version of prompt tuning: in the second layer of $\mathbf{G}$, the bias term td is a task-specific prompt, with preceding parts $td \times m$ generating an instance-dependent prompt. The final prompt our method generated is a combination of both. In short, what we discussed here is to generate a t-length prompt for one Transformer layer. An optimization of multi-layer prompt generation will be introduced in Section 3.2.2.

We can control the added number of trainable parameters by setting $m \ll d$, but it is still expensive since hidden dimension d is usually large (1024 in BERT/RoBERTa-Large). In the sequel, we will introduce a parameter squeezing method to further reduce trainable parameters without sacrificing performance.

Note that our proposed method relies on the input sentence representation $\mathbf{M}(x_i)$ to generate prompts. One caveat is that this method will have two forward passes of the pre-trained LM during inference time – first to generate $\mathbf{M}(x_i)$ and then to generate classification results. However, the sentence representation $\mathbf{M}(x_i)$ used in our method is task-agnostic. In practice, we can cache the prediction $\mathbf{M}(x_i)$ and use it in various downstream tasks or rely on a lightweight sentence representation such as GloVe (Pennington et al., 2014) (Cf. Section 4.5.1).

3.2 Optimization

We propose two optimization techniques to further improve our proposed method.

3.2.1 Parameterized Hypercomplex Multiplication (PHM) Layers

Inspired by the recent application of parameterized hypercomplex multiplication (PHM) layers (Zhang et al., 2021) in Compacter (Mahabadi et al., 2021), we leverage PHM layers to optimize our prompt generator, $\mathbf{G}$. Generally, the PHM layer is a fully-connected layer with form $y = \mathbf{W}x + b$, where $x \in \mathbb{R}^d$ is the input feature, $y \in \mathbb{R}^m$ is the output feature, and $\mathbf{W} \in \mathbb{R}^{m\times d}$ and $b \in \mathbb{R}^m$ are the trainable parameters. When m and d are large, the cost of learning $\mathbf{W}$ becomes the main bottleneck. PHM replaces the matrix $\mathbf{W}$ by a sum of Kronecker products of several small matrices. Given a user-defined hyperparameter $n \in \mathbb{Z}^+$ that divides m and d, $\mathbf{W}$ can be calculated as follows:

$$\mathbf{W} = \sum_{i=1}^{n} \mathbf{A}_i \bigotimes \mathbf{B}_i \tag{4}$$

where $\mathbf{A}_i \in \mathbb{R}^{n\times n}$, $\mathbf{B}_i \in \mathbb{R}^{\frac{m}{n}\times\frac{d}{n}}$, and $\bigotimes$ is Kronecker product. In this way, the number of trainable parameters is reduced to $n \times (n \times n + \frac{m}{n} \times \frac{d}{n}) = n^3 + \frac{m\times d}{n}$. As n is usually much smaller than m and d, PHM reduces the amount of parameters by a factor of n.

Suppose that we have a two layer perceptron with down-sample projection $\mathbf{W}_1 \in \mathbb{R}^{m\times d}$ and up-sample projection $\mathbf{W}_2 \in \mathbb{R}^{t\times d\times m}$, where d is the input embedding dimension, m is the hidden layer dimension, and t is the number of tokens we generate. For example, we use RoBERTa-Large with hidden size $d = 1024$, generator hidden size $m = 256$, $n = 16$, prompt length $t = 5$. By substituting the $\mathbf{W}_1$ and $\mathbf{W}_2$ by two PHM layers and letting A_i shared by both layers, we can reduce the number of parameters from 1.5M to 105K.

3.2.2 Multi-layer Prompt Tuning

Prompt tuning (Lester et al., 2021) and P-tuning (Liu et al., 2021b) both insert continuous prompts into the first transformer layer (cf. Figure 2(b)). While proven efficient in some specific settings, single layer prompt tuning has two main limitations: (i) Capturing deep contextual information: the impact of the first-layer prompts on final prediction is low when transformer goes deeper. (ii) Generalizing to long sequence tasks: it is unclear that prompt tuning can perform well in tasks with long input when only a limited number of parameters can be inserted in single layer.

Following Prefix tuning (Li and Liang, 2021) and P-tuning v2 (Liu et al., 2021a), we prepend our generated prompts at each transformer layer to address the above issues. However, simply generalizing our model (IDPG) to a multi-layer version (M-IDPG), will significantly increase the number of training parameters, since each layer requires an independent generator $\mathbf{G}$. Instead, we explore different architectures in Section 4.5.3 to balance the number of tuned parameters against model performance. In short, assuming each layer generator G_i has form $y = \mathbf{W}x + b_i$, we share the weight matrix $\mathbf{W}$ across generators and set the bias term $b_i \in \mathbb{R}^m$ to be layer-specific, where $i = 1, \ldots, N$ is the layer index and N is the number of transformer layers.

Table 1: Main results of different transfer learning method. Each methods are evaluated on full test sets (dev sets for GLUE tasks). We report average results across 5 runs with different initialization. **Bold** marks the best result among all competing methods. Underline marks the best result among all prompt tuning methods. We report the average of accuracy and F1 for both MRPC and QQP, and average of Pearson and Spearman correlation coefficients for STS-B. For all the other tasks, we report accuracy.

Method	**MPQA**	**Subj**	**CR**	**MR**	**SST-2**	**QNLI**	**RTE**	**MRPC**	**STS-B**	**QQP**	Avg
Transformer Fine-tuning											
RoBERTa	90.4$_{\pm 0.2}$	97.1$_{\pm 0.1}$	90.7$_{\pm 0.7}$	91.7$_{\pm 0.2}$	96.4$_{\pm 0.2}$	**94.7**$_{\pm 0.1}$	85.7$_{\pm 0.2}$	91.8$_{\pm 0.4}$	92.2$_{\pm 0.2}$	**91.0**$_{\pm 0.1}$	92.2
EFL	90.3$_{\pm 0.2}$	97.2$_{\pm 0.1}$	93.0$_{\pm 0.7}$	91.7$_{\pm 0.2}$	**96.5**$_{\pm 0.1}$	94.4$_{\pm 0.1}$	85.6$_{\pm 2.4}$	91.2$_{\pm 0.4}$	**92.5**$_{\pm 0.1}$	**91.0**$_{\pm 0.2}$	92.3
Adapter											
Compacter	91.1$_{\pm 0.2}$	97.5$_{\pm 0.1}$	92.7$_{\pm 0.4}$	**92.6**$_{\pm 0.2}$	96.0$_{\pm 0.2}$	94.3$_{\pm 0.2}$	87.1$_{\pm 1.4}$	91.6$_{\pm 0.6}$	91.6$_{\pm 0.1}$	87.1$_{\pm 0.2}$	92.2
Adapter	90.8$_{\pm 0.2}$	97.5$_{\pm 0.1}$	92.8$_{\pm 0.3}$	92.5$_{\pm 0.1}$	96.1$_{\pm 0.1}$	94.8$_{\pm 0.2}$	88.1$_{\pm 0.4}$	91.8$_{\pm 0.6}$	92.1$_{\pm 0.1}$	89.9$_{\pm 0.1}$	**92.6**
Prompting											
Prompt-tuning	90.3$_{\pm 0.2}$	95.5$_{\pm 0.4}$	91.2$_{\pm 1.1}$	91.0$_{\pm 0.2}$	94.2$_{\pm 0.3}$	86.0$_{\pm 0.3}$	87.0$_{\pm 0.4}$	84.3$_{\pm 0.3}$	87.2$_{\pm 0.2}$	81.6$_{\pm 0.1}$	88.8
Prompt-tuning-134	65.7$_{\pm 19}$	95.6$_{\pm 0.2}$	86.7$_{\pm 3.6}$	89.7$_{\pm 0.5}$	92.0$_{\pm 0.5}$	83.0$_{\pm 1.1}$	87.4$_{\pm 0.5}$	84.1$_{\pm 0.5}$	87.6$_{\pm 0.5}$	82.4$_{\pm 0.3}$	85.4
Ptuningv2	90.4$_{\pm 0.3}$	96.5$_{\pm 0.3}$	92.7$_{\pm 0.3}$	91.6$_{\pm 0.1}$	94.4$_{\pm 0.2}$	92.9$_{\pm 0.1}$	78.4$_{\pm 4.3}$	91.4$_{\pm 0.4}$	89.9$_{\pm 0.2}$	84.4$_{\pm 0.4}$	90.3
S-IDPG-PHM	89.6$_{\pm 0.3}$	94.4$_{\pm 0.3}$	90.3$_{\pm 0.2}$	89.3$_{\pm 0.4}$	94.7$_{\pm 0.2}$	90.7$_{\pm 0.3}$	89.2$_{\pm 0.2}$	84.3$_{\pm 0.8}$	84.7$_{\pm 0.9}$	82.5$_{\pm 0.2}$	89.0
S-IDPG-DNN	89.5$_{\pm 0.7}$	94.9$_{\pm 0.4}$	89.9$_{\pm 1.5}$	90.2$_{\pm 0.6}$	95.1$_{\pm 0.2}$	90.5$_{\pm 0.5}$	$\underline{\mathbf{89.4}}_{\pm 0.4}$	83.0$_{\pm 0.5}$	85.3$_{\pm 0.7}$	82.7$_{\pm 0.3}$	89.1
M-IDPG-PHM-GloVe	90.9$_{\pm 0.2}$	97.4$_{\pm 0.1}$	93.3$_{\pm 0.1}$	$\underline{\mathbf{92.6}}_{\pm 0.3}$	95.4$_{\pm 0.2}$	94.4$_{\pm 0.2}$	82.1$_{\pm 0.6}$	92.1$_{\pm 0.4}$	91.0$_{\pm 0.4}$	86.3$_{\pm 0.2}$	91.6
M-IDPG-PHM	$\underline{\mathbf{91.2}}_{\pm 0.2}$	97.5$_{\pm 0.1}$	93.2$_{\pm 0.3}$	$\underline{\mathbf{92.6}}_{\pm 0.3}$	$\underline{96.0}_{\pm 0.3}$	$\underline{94.5}_{\pm 0.1}$	83.5$_{\pm 0.7}$	$\underline{\mathbf{92.3}}_{\pm 0.2}$	91.4$_{\pm 0.4}$	86.2$_{\pm 0.1}$	91.9
M-IDPG-DNN	$\underline{\mathbf{91.2}}_{\pm 0.3}$	$\underline{\mathbf{97.6}}_{\pm 0.2}$	$\underline{\mathbf{93.5}}_{\pm 0.3}$	$\underline{\mathbf{92.6}}_{\pm 0.1}$	95.9$_{\pm 0.1}$	$\underline{94.5}_{\pm 0.2}$	85.5$_{\pm 0.6}$	91.8$_{\pm 0.3}$	$\underline{91.5}_{\pm 0.2}$	$\underline{86.9}_{\pm 0.3}$	$\underline{92.1}$

4 Experiment Results

4.1 Experimental Setup

We evaluate on ten standard natural language understanding (NLU) datasets – MPQA (Wiebe et al., 2005), Subj (Pang and Lee, 2004), CR (Hu and Liu, 2004), MR (Pang and Lee, 2005), and six tasks from GLUE (Wang et al., 2019), viz. SST-2, QNLI, RTE, MRPC, STS-B (Cer et al., 2017) and QQP. We compare our proposed method with a wide range of methods, as follows:

Transformer fine-tuning: We instantiated two versions – a vanilla transformer fine-tuning (Liu et al., 2019) and the entailment-based fine-tuning (Wang et al., 2021).

Prompt tuning: We implemented two versions – standard prompt tuning (Lester et al., 2021) and multi-layer prompt tuning (Li and Liang, 2021; Liu et al., 2021a).

Adapter-based fine-tuning: This efficient transfer learning method inserts an adaptation module inside each transformer layer including Compactor (Mahabadi et al., 2021) and Adapter (Houlsby et al., 2019).

We compare these against two versions of single-layer instance-dependent generation methods: S-IDPG-DNN and S-IDPG-PHM. The first version is based on a 2-layer perceptron generator, which contains 1.5M parameters. The second one uses the PHM layer and only contains 105K parameters.

We also explore three versions of multi-layer instance-dependent generation methods: M-IDPG-DNN, M-IDPG-PHM, M-IDPG-PHM-GloVe. Again, the difference between the first two is in the prompt generator, while M-IDPG-PHM-GloVe uses GloVe to encode input sequences.

For a fair comparison, all the pre-trained LMs are 24-layer 16-head RoBERTa-Large models (Liu et al., 2019). Additional training details can be found in Appendix A.1. Notably, Prompt-tuning-134 uses 134 prompt lengths in Table 1, and it is set so to match the training parameters of the proposed method, M-IDPG-PHM.

4.2 Performance in high-resource scenario

Table 1 shows the results of all the methods on full datasets across 10 NLU tasks. We observe that: (i) Our proposed method M-IDPG-PHM consistently outperforms the prompt tuning method and Ptuning v2 by average 3.1pt and 1.6pt, respectively (except on the RTE dataset). (ii) Compared with other efficient transfer learning methods, IDPG performs slightly worse than the Compacter (Mahabadi et al., 2021) and Adapter (Houlsby et al., 2019), across the ten tasks. However, the gap is mostly from RTE and QQP. Note that IDPG uses 15K fewer parameters than the Compacter. M-IDPG-PHM is better than Compacter on four tasks and has the same performance on three tasks. (iii) The improvement of our method is more prominent in the single-sentence classification task. The four best results (MPQA, Subj, CR, MR) among all competing methods in single-sentence classification tasks are made by IDPG models. Specifically, M-IDPG-PHM performs 0.84pt and 0.36pt better than RoBERTa and EFL, respectively. (iv) PHM-based generator performs on par with the DNN-based generator while having a significantly lower number of trainable parameters. (v) GloVe-based sentence encoder also performs similar to LM-based sentence encoder, indicating the advancement of instance-dependent prompt generation does not rely on a robust contextual sentence encoder. (vi) When we fix the training parameters to be the same, the comparison between Prompt-tuning-134 and M-IDPG-PHM illustrates that our approach works better than prompt tuning not just because of using more parameters.

4.3 Efficiency

Table 2 lists the number of trainable parameters for different methods excluding the classification head. The general goal for efficient transfer learning is to train models with fewer parameters while achieving better performance. Traditional prompt-tuning method only requires training a token embedding table with a few thousand parameters. However, its performance is worse than a lightweight adapter model (e.g., Compacter with 149K parameters). Our proposed method, especially the M-IDPG-PHM, falls in the gap between prompt-tuning and adapter model, since it only requires training 134K parameters and performs on par with Compacter.

Method	# Parameters
Transformer Fine-tune (Liu et al., 2019)	355M
Adapter (Houlsby et al., 2019)	1.55M
Compacter (Mahabadi et al., 2021)	149K
Prompt-tuning (Lester et al., 2021)	5K
Prompt-tuning-134 (Lester et al., 2021)	134K
P-Tuningv2 (Liu et al., 2021a)	120K
S-IDPG-PHM	105K
S-IDPG-DNN	1.5M
M-IDPG-PHM-GloVe	141K
M-IDPG-PHM	134K
M-IDPG-DNN	216K

Table 2: Number of trainable parameters of different methods. Note that we did not include the parameters from classification heads.

4.4 Performance in low-resource scenario

We further evaluate our proposed method in the low-resource scenario. Following the existing evaluation protocols in the few-shot setting (He et al., 2021), we sample a subset of the training data for each task with size $K \in \{100, 500, 1000\}$ as our training data and another subset with size 1000 as a development set. We compare our proposed methods with all prompt tuning methods, one fine-tuning model (EFL), and one adapter tuning model (Compacter).

In the extreme low-resource case when K=100, M-IDPG-PHM performs 2.5pt better than the traditional prompt tuning method and 0.5pt better than the multi-layer P-Tuning v2 method. This improvement illustrates that our method has better generalization in few-shot settings. When K becomes larger, IDPG-PHM still maintains good results with 1.9pt and 0.2pt improvement (K=500); and 2.0pt and 0.2pt improvement (K=1000) in accuracy with traditional prompt tuning and P-tuning v2 approaches, respectively. We also observe that sometimes when K is small, our method results have high variance (e.g., 4.6 on MPQA, when $K = 100$). We suspect that this may be due to poor initialization leading the model to non-optimal parameters.

We also note that other state-of-the-art models, such as LM-BFF (Gao et al., 2021a), attempt to address the few-shot learning problem from a different perspective. We want to highlight that we are exploring a solution by training as few parameters as possible while maintaining good performance. Testing the limitation of our model without freezing any parameters would be an interesting investigation, but is not the main focus of this paper.

4.5 Intrinsic Study

We conduct several ablation studies including exploration of different generator architectures and impact of selecting different prompt positions.

4.5.1 Sentence Encoder: GloVe or LMs?

The proposed IDPG method relies on pre-trained LM to extract sentence representation, i.e., `[CLS]` token embedding. Obtaining contextualized transformer sentence embedding is often expensive if it is not pre-computed. One open question is to explore reliability on lightweight sentence representations such as GloVe embedding (Pennington et al., 2014) or token embedding of pre-trained language models.

To answer this question, we apply the pre-trained GloVe word vectors[2] to extract the sentence representation. Specifically, we take the average of word vectors as the sentence embeddings:

$$\mathbf{M}(x_i) = \frac{1}{k}\sum_{j=1}^{k} \texttt{GloVe}(t_j),\ x_i \in D_{train} \quad (5)$$

[2] Obtained from https://nlp.stanford.edu/data/glove.6B.zip version glove.6b.300d.txt

Table 3: Low-resource results are evaluated on full test sets. We report average results across 5 runs with different initialization. **Bold** marks the best result among all competing methods. Underline marks the best result among all prompt tuning methods. We report the average of accuracy and F1 for both MRPC and QQP, and average of Pearson and Spearman correlation coefficients for STS-B. For all other tasks, we report accuracy.

Method	**MPQA**	**Subj**	**CR**	**MR**	**SST-2**	**QNLI**	**RTE**	**MRPC**	**STS-B**	**QQP**	Avg
K = 100											
Fine-tuning (EFL)	**86.2**$_{\pm0.4}$	88.4$_{\pm0.8}$	83.7$_{\pm2.4}$	81.4$_{\pm1.0}$	86.2$_{\pm1.3}$	77.7$_{\pm1.5}$	84.2$_{\pm1.2}$	72.6$_{\pm3.7}$	**84.1**$_{\pm1.6}$	**78.1**$_{\pm0.4}$	**82.2**
Adapter-tuning (Compacter)	81.0$_{\pm2.9}$	88.7$_{\pm0.8}$	**84.7**$_{\pm2.1}$	**83.7**$_{\pm0.7}$	85.7$_{\pm0.9}$	75.6$_{\pm0.8}$	84.7$_{\pm0.6}$	80.0$_{\pm0.9}$	78.1$_{\pm1.4}$	77.1$_{\pm0.6}$	81.9
prompt tuning	75.9$_{\pm1.6}$	86.8$_{\pm0.8}$	72.9$_{\pm1.4}$	74.1$_{\pm1.4}$	82.9$_{\pm2.0}$	**82.7**$_{\pm0.2}$	86.5$_{\pm0.6}$	80.0$_{\pm1.3}$	70.2$_{\pm3.1}$	76.5$_{\pm0.4}$	78.9
P-Tuningv2	74.3$_{\pm2.9}$	89.7$_{\pm0.8}$	80.1$_{\pm1.0}$	82.5$_{\pm1.1}$	85.1$_{\pm1.6}$	78.2$_{\pm0.5}$	83.6$_{\pm0.7}$	**80.1**$_{\pm0.6}$	78.8$_{\pm3.0}$	76.8$_{\pm0.5}$	80.9
S-IDPG-PHM	79.0$_{\pm3.7}$	87.6$_{\pm1.1}$	75.0$_{\pm1.6}$	76.2$_{\pm1.3}$	87.6$_{\pm1.3}$	80.4$_{\pm1.2}$	86.3$_{\pm0.5}$	79.3$_{\pm0.4}$	70.9$_{\pm2.5}$	76.1$_{\pm0.6}$	79.8
S-IDPG-DNN	78.0$_{\pm2.1}$	84.2$_{\pm1.6}$	76.3$_{\pm4.5}$	77.4$_{\pm0.5}$	**89.6**$_{\pm1.2}$	81.1$_{\pm0.8}$	**87.4**$_{\pm0.8}$	78.8$_{\pm1.3}$	70.6$_{\pm2.8}$	74.1$_{\pm0.9}$	79.8
M-IDPG-PHM-GloVe	76.6$_{\pm2.0}$	**90.7**$_{\pm0.4}$	80.6$_{\pm2.6}$	83.0$_{\pm1.5}$	85.6$_{\pm0.8}$	77.9$_{\pm1.3}$	84.4$_{\pm0.9}$	79.6$_{\pm0.9}$	77.8$_{\pm1.6}$	76.1$_{\pm0.7}$	81.2
M-IDPG-PHM	75.5$_{\pm4.6}$	90.5$_{\pm0.6}$	80.2$_{\pm1.5}$	82.5$_{\pm1.1}$	85.9$_{\pm1.2}$	78.8$_{\pm1.6}$	84.0$_{\pm0.4}$	79.9$_{\pm0.8}$	79.3$_{\pm0.4}$	77.1$_{\pm0.2}$	81.4
K = 500											
Fine-tuning (EFL)	85.1$_{\pm1.7}$	94.1$_{\pm0.4}$	**90.9**$_{\pm0.6}$	87.6$_{\pm0.5}$	**92.5**$_{\pm0.6}$	85.7$_{\pm0.6}$	57.5$_{\pm1.0}$	82.3$_{\pm0.6}$	**88.8**$_{\pm0.5}$	**79.0**$_{\pm0.3}$	84.3
Adapter-tuning (Compacter)	**86.0**$_{\pm0.8}$	94.9$_{\pm0.2}$	89.5$_{\pm1.0}$	88.5$_{\pm0.2}$	91.9$_{\pm0.9}$	82.2$_{\pm0.6}$	83.9$_{\pm0.8}$	82.7$_{\pm0.5}$	86.6$_{\pm0.5}$	78.9$_{\pm0.3}$	**86.5**
prompt tuning	82.4$_{\pm1.3}$	91.2$_{\pm0.1}$	86.8$_{\pm0.4}$	84.6$_{\pm0.8}$	88.6$_{\pm1.0}$	**86.3**$_{\pm0.4}$	86.5$_{\pm0.4}$	80.0$_{\pm0.4}$	77.4$_{\pm1.9}$	77.8$_{\pm0.3}$	84.2
P-Tuningv2	84.0$_{\pm1.3}$	94.6$_{\pm0.3}$	89.0$_{\pm1.8}$	88.1$_{\pm0.5}$	91.3$_{\pm0.7}$	84.6$_{\pm0.8}$	84.2$_{\pm1.5}$	**83.2**$_{\pm0.7}$	83.8$_{\pm0.5}$	78.6$_{\pm0.3}$	86.1
S-IDPG-PHM	81.6$_{\pm2.7}$	91.4$_{\pm0.7}$	85.8$_{\pm2.0}$	85.8$_{\pm0.5}$	88.5$_{\pm1.3}$	85.0$_{\pm0.4}$	86.3$_{\pm1.3}$	81.9$_{\pm0.8}$	78.3$_{\pm1.5}$	78.1$_{\pm0.3}$	84.3
S-IDPG-DNN	84.8$_{\pm0.7}$	90.8$_{\pm0.6}$	89.7$_{\pm1.0}$	86.1$_{\pm2.8}$	90.4$_{\pm1.6}$	84.8$_{\pm0.3}$	**87.7**$_{\pm0.7}$	82.0$_{\pm1.4}$	79.1$_{\pm2.3}$	77.1$_{\pm0.4}$	85.3
M-IDPG-PHM-GloVe	84.0$_{\pm1.7}$	**95.0**$_{\pm0.2}$	89.0$_{\pm1.1}$	88.1$_{\pm0.5}$	90.4$_{\pm1.3}$	85.1$_{\pm0.1}$	84.0$_{\pm1.0}$	82.3$_{\pm0.5}$	84.1$_{\pm0.8}$	78.2$_{\pm0.8}$	86.0
M-IDPG-PHM	85.2$_{\pm1.1}$	94.6$_{\pm0.0}$	89.1$_{\pm1.6}$	**88.8**$_{\pm0.4}$	91.6$_{\pm1.1}$	84.9$_{\pm0.9}$	83.9$_{\pm0.7}$	82.5$_{\pm0.5}$	84.2$_{\pm0.5}$	78.6$_{\pm0.3}$	86.3
K = 1000											
Fine-tuning (EFL)	87.7$_{\pm0.7}$	95.1$_{\pm0.2}$	89.8$_{\pm1.2}$	89.2$_{\pm0.5}$	93.6$_{\pm0.4}$	**88.0**$_{\pm0.7}$	87.3$_{\pm1.3}$	**87.9**$_{\pm0.9}$	**90.8**$_{\pm0.2}$	79.8$_{\pm0.3}$	**88.9**
Adapter-tuning (Compacter)	**88.2**$_{\pm0.6}$	95.6$_{\pm0.3}$	**89.9**$_{\pm1.4}$	**90.0**$_{\pm0.3}$	92.9$_{\pm0.2}$	85.2$_{\pm0.7}$	86.8$_{\pm0.7}$	86.1$_{\pm0.6}$	89.6$_{\pm0.5}$	**79.9**$_{\pm0.3}$	88.4
prompt tuning	83.9$_{\pm2.0}$	92.6$_{\pm0.4}$	87.2$_{\pm1.4}$	86.7$_{\pm0.3}$	89.9$_{\pm1.0}$	86.9$_{\pm0.1}$	86.4$_{\pm0.7}$	82.5$_{\pm0.3}$	82.9$_{\pm1.3}$	78.6$_{\pm0.3}$	85.8
P-Tuningv2	87.0$_{\pm0.9}$	**95.9**$_{\pm0.4}$	88.3$_{\pm1.5}$	89.5$_{\pm0.3}$	93.2$_{\pm0.5}$	87.4$_{\pm0.4}$	85.1$_{\pm1.1}$	82.6$_{\pm1.1}$	87.8$_{\pm0.3}$	79.3$_{\pm0.4}$	87.6
S-IDPG-PHM	83.4$_{\pm1.7}$	93.4$_{\pm0.9}$	89.2$_{\pm0.8}$	88.0$_{\pm0.9}$	90.2$_{\pm1.0}$	85.5$_{\pm0.6}$	86.9$_{\pm0.6}$	83.1$_{\pm0.4}$	83.9$_{\pm0.8}$	78.9$_{\pm0.4}$	86.3
S-IDPG-DNN	85.9$_{\pm0.8}$	93.3$_{\pm1.2}$	**89.9**$_{\pm0.8}$	89.6$_{\pm1.1}$	92.2$_{\pm0.8}$	85.2$_{\pm1.3}$	**87.7**$_{\pm0.8}$	82.5$_{\pm0.9}$	84.7$_{\pm0.9}$	78.0$_{\pm0.8}$	86.9
M-IDPG-PHM-GloVe	86.5$_{\pm0.7}$	95.5$_{\pm0.3}$	87.7$_{\pm1.3}$	89.3$_{\pm0.4}$	93.4$_{\pm0.3}$	87.5$_{\pm0.3}$	84.9$_{\pm0.9}$	82.7$_{\pm0.7}$	87.6$_{\pm0.3}$	79.1$_{\pm0.7}$	87.4
M-IDPG-PHM	87.7$_{\pm0.5}$	95.6$_{\pm0.2}$	89.2$_{\pm1.2}$	89.8$_{\pm0.4}$	**93.7**$_{\pm0.6}$	87.2$_{\pm0.5}$	85.6$_{\pm0.6}$	82.5$_{\pm0.9}$	87.8$_{\pm0.8}$	79.1$_{\pm0.4}$	87.8

where x_i is the input sequence with k tokens $t_1, \ldots, t_k$. According to Table 1, using GloVe as sentence encoder to generate prompts doesn't sacrifice much performance over the ten tasks and outperforms prompt tuning and P-tuning v2. It indicates that our model does not benefit a lot from a strong contextual pre-trained LM. Instead, a light sentence encoder such as GloVe can also help the tasks. Also, instance-dependent prompt tuning shows promising improvement over non-instance-dependent prompt tuning models. One of the drawbacks of our method is that it is twice as expensive to run compared to Compacter, even though it uses slightly fewer parameters. Adopting GloVe as sentence encoder would avoid going through the LM twice, thus effectively reducing IDPG's run-time complexity by half.

4.5.2 Prompt Generator: PHM or DNN?

To reduce the tuning parameters, we substitute the DNN layers with PHM layers. An open question we seek to answer is what is the best generation model for prompt regardless of training parameters. Hence, we compare the PHM-based prompt generator with the DNN-based prompt generator, as shown in Table 1. We observe that including DNN as a generator doesn't improve performance significantly, with +0.1pt gain on average, while adding 87K parameters (with hidden size m=16). On the other hand, this ablation study further verifies PHM layers' efficiency in the generation model.

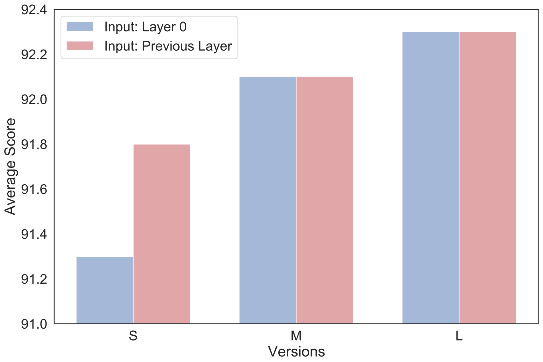


Figure 3: Comparison between three different multi-layer generator models (S, M, L versions), and comparison between taking layer 0's output or previous layer's output as input.

4.5.3 Multi-layer Architecture Exploration

When applying the instance-dependent generation model **G** into a multi-layer case, the first challenge we face is the considerable increase in training parameters. If each transformer layer requires an independent generator $\mathbf{G}_i$, the number of training parameters increases N times, where N is the number of transformer layers (24 in RoBERTa-Large). Assuming $\mathbf{G}$ has the form $y = \mathbf{W}x + b$, there are three alternatives: (i) Smallest version (S version): sharing both W and b; (ii) Middle version (M version): sharing W and making b layer-specific; and (iii) Largest version (L version): making both W and b layer-specific.

Another way to reduce the training parameters is by adjusting the hidden size m of the generator. We compare two models with $m = 16$ and $m = 256$. Surprisingly, we find that generator with a hidden size 16 is not far from the large model (92.0 vs. 92.1, respectively, in M version). We hypothesize that the smaller hidden size of 16 is already enough to store useful instance information, and setting m too large may be less efficient.

Besides, in single-layer prompt generation model, the input to $\mathbf{G}$ is $M(x_i)$ - the representation of input sequence x_i. In a multi-layer case, the input to each layer generator has another option, i.e., the previous layer's output. However, as shown in Figure 3, the experiment results suggest no significant difference between the two input ways. As for the generator selection, the three models perform as expected (S version < M version < L version). In Table 1, M-IDPG-PHM uses the previous layer's output as input, M version as the generator, and 16 as the generator hidden size. Detailed information for all models' performance on each task can be found in Appendix A.3.

4.5.4 Prompt Insertion: Single-layer or Multi-layer?

P-tuning v2 (Liu et al., 2021a) conducted substantial ablation studies on the influence of inserting prompt into different transformer layers. To boost single-layer IDPG performance, we add supplementary training (cf. Appendix A.4) and conduct ablation studies in Appendix A.5. We come to a similar conclusion that multi-layer instance-dependent prompt tuning model (M-IDPG) is significantly better than the single-layer method (S-IDPG) in both evaluation settings. An interesting finding is that the impact of supplementary training on S-IDPG is high while it is limited for M-IDPG.

4.5.5 How Prompts Help?

Given two sentences, we encode each of them by one of the comparison models and compute the cosine similarity. We sort all sentence pairs in STS-B dev set in descending order by the cosine similarity scores and get a distribution for number of pairs in each group that is included in Top-k ranking. We compare a vanilla model without any prompts with M-IDPG-PHM. Both models are fine-tuned on STS-B training set. As shown in Figure 4, prompts bring the similar sentences closer while pushing the dissimilar ones apart.

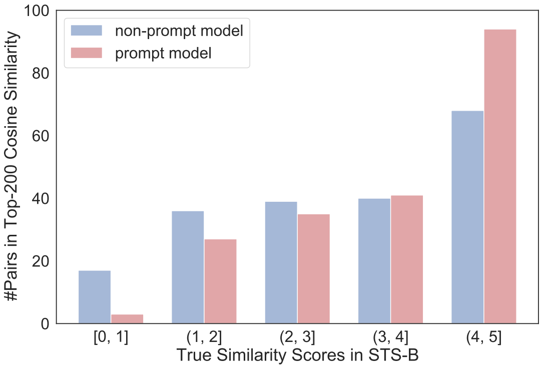


Figure 4: The number of pairs of each group in Top-200 cosine similarity ranking. More results can be found in Appendix A.6.

4.5.6 IDPG Scalability

We study our proposed model's scalability in this section. In general, the performance of IDPG in downstream tasks improves gradually when using a larger prompt length (Cf. Appendix A.7).

5 Related Work

Supplementary Training: Existing works (Phang et al., 2018; Liu et al., 2019) have observed that starting from the fine-tuned MNLI model results in a better performance than directly from the vanilla pre-trained models for RTE, STS, and MRPC tasks. A series of work (SentenceBERT (Reimers and Gurevych, 2019), BERT-flow (Li et al., 2020), SimCSE (Gao et al., 2021b)) explored intermediate training to improve STS tasks. All of them applied pre-fine tuning on NLI datasets. More recently, EFL (Wang et al., 2021) proposed a task transformation paradigm, improving single sentence tasks with less labels using rich sentence-pair datasets.

Adapter Tuning: Adapter tuning has emerged as a novel parameter-efficient transfer learning paradigm (Houlsby et al., 2019; Pfeiffer et al., 2020), in which adapter layers – small bottleneck layers – are inserted and trained between frozen pre-trained transformer layers. On the GLUE benchmark, adapters attain within 0.4% of the performance of full fine-tuning by only training 3.6% parameters per task. Compactor (Mahabadi et al., 2021) substitutes the down-projector and up-projector matrices by a sum of Kronecker products, reducing the parameters by a large margin while maintaining the overall performance.

Prompting: Hand-crafted prompts were shown to be helpful to adapt generation in GPT-3 (Brown et al., 2020). Existing works including LM-BFF (Gao et al., 2021a; Wang et al., 2021) explored the prompt searching in a few-shot setting.

Recently, several researchers have proposed continuous prompts training to overcome the challenges in discrete prompt searching. Prefix tuning (Li and Liang, 2021) and P-tuningv2 (Liu et al., 2021a) prepend a sequence of trainable embeddings at each transformer layer and optimizes them. Two contemporaneous works – prompt tuning (Lester et al., 2021) and P-tuning (Liu et al., 2021b), interleave the training parameters in the input embedding layer instead of each transformer layer. All these methods focus on task-specific prompt optimization. Our proposed method, IDPG, is the first prompt generator that is not only task-specific but also instance-specific.

6 Conclusion and Discussion

We have introduced IDPG, an instance-dependent prompt generation model that generalizes better than the existing prompt tuning methods. Our method first factors in an instance-dependent prompt, which is robust to data variance. Parameterized Hypercomplex Multiplication (PHM) is applied to shrink the training parameters in our prompt generator, which helps us build an extreme lightweight generation model. Despite adding fewer parameters than prompt tuning, IDPG shows consistent improvement. It is also on par with the lightweight adapter tuning methods such as Compacter while using a similar amount of trainable parameters. This work provided a new research angle for prompt-tuning of a pre-trained language model.

Acknowledgment

We thank Brian Lester, Jonas Pfeiffer, Tianyu Gao, Qinyuan Ye, Zhenhao Zhang, Fei Sun, members of the University of Michigan's NLP4Health research group, and all anonymous reviewers for helpful discussion and valuable feedback.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

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*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021a. 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 (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021b. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jiawei Low, Lidong Bing, and

Luo Si. 2021. On the effectiveness of adapter-based tuning for pretrained language model adaptation. 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 2208–2222, Online. Association for Computational Linguistics.

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.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177.

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*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9119–9130, Online. 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 (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

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

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

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

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

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 (ACL-04)*, pages 271–278, Barcelona, Spain.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

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.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

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.

Sinong Wang, Han Fang, Madian Khabsa, Hanzi Mao, and Hao Ma. 2021. Entailment as few-shot learner. *arXiv preprint arXiv:2104.14690*.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2):165–210.

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. *arXiv preprint arXiv:2102.08597*.

A Appendix

A.1 Experimental Settings

A.1.1 Training hyperparameters

We use RoBERTa-Large (Liu et al., 2019) model implemented by Fairseq (Ott et al., 2019) as our basic model. The detailed model hyperparameters are listed in Table 4.

Note that for both transformer fine-tuning methods, including RoBERTa (Liu et al., 2019) and EFL (Wang et al., 2021), we follow their official training instructions, i.e., using a polynomial learning rate scheduler with 6% steps to warm up and tuning for ten epochs. For all adapter-based and prompt-based methods, we train them more sufficiently (with fifty epochs) on small datasets (i.e., MPQA, Subj, CR, MR, RTE, MRPC, STS-B).

A.1.2 Model hyperparameters

We report the detailed model hyperparameters for each method in Table 1 and illustrate how numbers in Table 2 are computed.

Compacter: hidden size $d = 1024$, adapter hidden size $m = 16$, user defined $n = 4$, each transformer layer inserts 2 compacters. Down-project s_i matrix takes $1024/4 \times 4 \times 24 \times 2 = 48K$, down-project t_i matrix takes $16/4 \times 4 \times 24 \times 2 = 0.75K$, hidden bias takes $16 \times 24 \times 2 = 0.75K$, up-project s_i and t_i matrix takes the same number of parameters as down-projector, the output bias takes $1024 \times 24 \times 2 = 48K$, the shared matrix A_i takes $4^3 \times 24 \times 2 = 3K$. **Total parameters**: $48 + 0.75 + 0.75 + 48 + 0.75 + 48 + 3 = 149.25K$.

Adapter: hidden size $d = 1024$, adapter hidden size $m = 16$. **Total parameters**: $(1024 \times 16 + 16 + 16 \times 1024 + 1024) \times 24 \times 2 = 1.55M$.

Prompt-tuning: prompt length $t = 5$. **Total parameters**: $5 \times 1024 = 5K$.

Prompt-tuning-134: prompt length $t = 134$. **Total parameters**: $134 \times 1024 = 134K$.

P-tuning v2: prompt length $t = 5$, inserted layers 24. **Total parameters**: $5 \times 24 \times 1024 = 120K$.

S-IDPG-PHM: hidden size $d = 1024$, generator hidden size $m = 256$, prompt length $t = 5$, user defined $n = 16$ (Cf. Equation 4). First PHM layer W_1 takes $1024/16 \times 256/16 \times 16 + 256 = 16.25K$ parameters, second PHM layer W_2 takes $256/16 \times 5 \times 1024/16 \times 16 + 5 \times 1024 = 85K$ parameters, the shared matrix A_i takes $16^3 = 4K$ (Note we use one shared matrix in single version IDPG). **Total parameters**: $105K$.

S-IDPG-DNN: hidden size $d = 1024$, generator hidden size $m = 256$, prompt length $t = 5$. **Total parameters**: $1024 \times 256 + 256 + 256 \times 5 \times 1024 + 5 \times 1024 = 1.5M$.

M-IDPG-PHM-GloVe: input vector size 300, generator hidden size $m = 16$, prompt length $t = 5$, user defined $n = 4$ (Cf. Equation 4). First PHM layer W_1 takes $300/4 \times 16/4 \times 4 + 16 = 1216$ parameters, second PHM layer W_2 takes $16/4 \times 5 \times 1024/4 \times 4 + 5 \times 1024 \times 24 = 140K$ parameters, the shared matrix A_i takes $4^3 \times 2 = 128$. **Total parameters**: $141K$.

M-IDPG-PHM: hidden size $d = 1024$, generator hidden size $m = 16$, prompt length $t = 5$, user defined $n = 16$ (Cf. Equation 4). First PHM layer W_1 takes $1024/16 \times 16/16 \times 16 + 16 = 1K$ parameters, second PHM layer W_2 takes $16/16 \times 5 \times 1024/16 \times 16 + 5 \times 1024 \times 24 = 125K$ parameters, the shared matrix A_i takes $16^3 16 \times 2 = 8K$. **Total parameters**: $134K$.

M-IDPG-DNN: hidden size $d = 1024$, generator hidden size $m = 16$, prompt length $t = 5$. **Total parameters**: $1024 \times 16 + 16 + 16 \times 5 \times 1024 + 5 \times 1024 \times 24 = 216K$.

A.2 Datasets

We provide a detailed information in Table 5 for 10 NLU datasets we used.

A.3 Detailed results for Multi-layer Architecture Exploration

We provide a detailed result table for all compared methods in Section 4.5.3. Note that the M version model with $m = 16$ and previous layer as input one is slightly higher than the results shown in Table 1(Cf. M-IDPG-PHM), this is because we tune the learning rate more carefully in Table 6 ($lr \in \{1e^{-2}, 7e^{-3}, 5e^{-3}, 3e^{-3}, 1e^{-3}, 7e^{-4}, 5e^{-4}, 3e^{-4}, 1e^{-4}\}$) to seek the best performance each model can reach. While in Table 1, we tune the learning rate from $\{5e^{-3}, 1e^{-3}, 5e^{-4}, 1e^{-4}\}$ to make the fair comparison with other models.

A.4 Supplementary Training for Single-layer IDPG

According to previous works (Phang et al., 2018; Wang et al., 2021), supplementing pre-trained LMs with rich data helps tasks with limited labels and stabilizes downstream fine-tuning. Following this idea, we also conduct intermediate training for single-layer IDPG.

Hyperparam	Supplmentary	Finetune	few-shot
#Layers	24	24	24
Hidden size	1024	1024	1024
FFN inner hidden size	4096	4096	4096
Attention heads	16	16	16
Attention head size	64	64	64
dropout	0.1	0.1	0.1
Learning Rate	linearly decayed	fixed	fixed
Peak Learning Rate	$1e^{-5}$	$\{5e^{-3}, 1e^{-3}, 5e^{-4}, 1e^{-4}\}$	$\{5e^{-4}, 1e^{-4}, 5e^{-5}, 1e^{-5}\}$
Batch Size	32	{16, 32}	16
Weight Decay	0.1	0.1	0.1
Training Epoch	10	10 or 50	10 or 50
Adam ϵ	$1e^{-6}$	$1e^{-6}$	$1e^{-6}$
Adam β_1	0.9	0.9	0.9
Adam β_2	0.98	0.98	0.98

Table 4: Hyperparameters for supplmentary training, fine-tuning, few-shot fine-tuning.

Corpus	|Train|	|Valiadation|	Task	Evaluation Metrics
Single Sentence Tasks				
CR	1,775	2,000	sentiment	accuracy
MR	8,662	2,000	sentiment	accuracy
SUBJ	8,000	2,000	sentiment	accuracy
MPQA	8,606	2,000	opinion polarity	accuracy
SST-2	67,349	1,821	sentiment analysis	accuracy
Sentence Pair Tasks				
QNLI	104,743	5,463	NLI	accuracy
RTE	2,491	278	NLI	accuracy
MRPC	3,668	409	paraphrase	accuracy/F1
QQP	363,846	40,430	paraphrase	accuracy/F1
STS-B	5,749	1,500	sentence similarity	Pearson/Spearman corr.

Table 5: The datasets evaluated in this work.

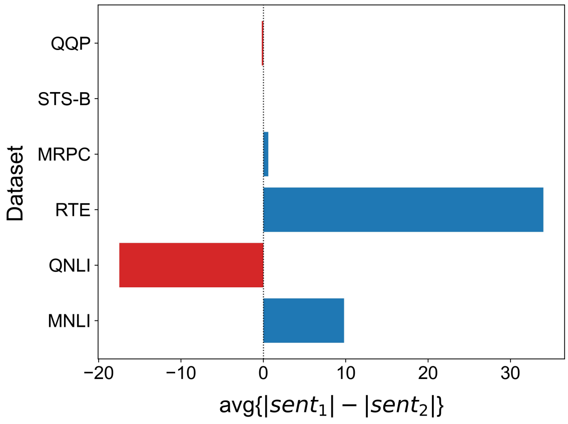


Figure 5: Length difference of GLUE sentence pair datasets.

However, a drawback of supplementary training is that if the data distribution of the downstream tasks is quite different from the supplementary training task, i.e., MRPC vs. MNLI (Wang et al., 2019), it may harm the downstream performance. Figure 5 provides a comprehensive statistic among all sentence pair tasks in GLUE benchmark. For example, the length of the first sentence in MNLI is 9.8 longer than the second sentence on average, while this length difference in MRPC is only 0.6. One natural solution to smooth the length distribution difference between tasks is to insert prompt in both supplementary training and downstream fine-tuning stage. For example, assuming that we are adding a prompt with a length $t = 5$ after the second sentence in the supplementary training stage on MNLI. Then, when fine-tuning downstream tasks

Table 6: Main results of different transfer learning method. Each methods are evaluated on full test sets (dev sets for GLUE tasks). We report average results across 5 runs with different initialization. We report the average of accuracy and F1 for both MRPC and QQP, and average of Pearson and Spearman correlation coefficients for STS-B. For all the other tasks, we report accuracy.

Method	**m**	**MPQA**	**Subj**	**CR**	**MR**	**SST-2**	**QNLI**	**RTE**	**MRPC**	**STS-B**	**QQP**	Avg
Input: Layer 0												
S version	256	$91.2_{\pm0.2}$	$97.6_{\pm0.1}$	$93.8_{\pm0.3}$	$92.6_{\pm0.2}$	$95.9_{\pm0.1}$	$93.8_{\pm0.1}$	$79.9_{\pm8.0}$	$90.8_{\pm0.5}$	$90.9_{\pm0.4}$	$85.9_{\pm0.4}$	91.2
M version	256	$91.2_{\pm0.3}$	$97.5_{\pm0.1}$	$93.6_{\pm0.3}$	$92.7_{\pm0.2}$	$95.7_{\pm0.2}$	$94.3_{\pm0.1}$	$85.5_{\pm1.0}$	$91.8_{\pm0.3}$	$91.4_{\pm0.2}$	$87.0_{\pm0.4}$	92.1
L version	256	$91.3_{\pm0.1}$	$97.6_{\pm0.2}$	$93.8_{\pm0.3}$	$92.6_{\pm0.1}$	$95.5_{\pm0.2}$	$94.5_{\pm0.2}$	$86.5_{\pm0.5}$	$92.5_{\pm0.8}$	$91.6_{\pm0.1}$	$87.3_{\pm0.3}$	92.3
Input: Previous Layer												
S version	256	$91.2_{\pm0.2}$	$97.5_{\pm0.1}$	$93.5_{\pm0.3}$	$92.6_{\pm0.1}$	$95.8_{\pm0.3}$	$94.0_{\pm0.1}$	$83.4_{\pm1.5}$	$91.9_{\pm0.3}$	$91.1_{\pm0.3}$	$86.9_{\pm0.2}$	91.8
M version	256	$91.0_{\pm0.2}$	$97.5_{\pm0.1}$	$93.4_{\pm0.4}$	$92.6_{\pm0.2}$	$96.0_{\pm0.1}$	$94.4_{\pm0.2}$	$86.6_{\pm1.2}$	$91.5_{\pm0.4}$	$91.4_{\pm0.2}$	$86.3_{\pm0.1}$	92.1
L version	256	$91.3_{\pm0.2}$	$97.4_{\pm0.0}$	$93.3_{\pm0.3}$	$92.5_{\pm0.2}$	$95.8_{\pm0.1}$	$94.5_{\pm0.3}$	$86.9_{\pm0.8}$	$92.1_{\pm0.4}$	$91.7_{\pm0.2}$	$87.1_{\pm0.2}$	92.3
Input: Previous Layer												
S version	16	$91.4_{\pm0.2}$	$97.5_{\pm0.1}$	$93.6_{\pm0.2}$	$92.5_{\pm0.2}$	$95.7_{\pm0.2}$	$93.9_{\pm0.0}$	$83.6_{\pm0.8}$	$91.9_{\pm0.4}$	$90.9_{\pm0.3}$	$85.5_{\pm0.4}$	91.6
M version	16	$91.2_{\pm0.2}$	$97.5_{\pm0.1}$	$93.4_{\pm0.3}$	$92.6_{\pm0.3}$	$96.0_{\pm0.3}$	$94.5_{\pm0.1}$	$83.5_{\pm0.7}$	$92.3_{\pm0.2}$	$91.4_{\pm0.4}$	$87.1_{\pm0.1}$	92.0

such as MRPC, we concatenate the prompt after the first sentence. In this way, the length difference in MNLI and MRPC becomes more balanced: 4.8 vs. $0.6 + 5 = 5.6$. As shown in Figure 6, we test five different insertion positions (Pos 0–4) for sentence pair tasks and three different positions (Pos 0, 1, 4) for single sentence tasks. We further reduce the distribution difference by reconstructing the supplementary training data. We double the MNLI dataset by reordering the two sentences on one shard, and use the doubled dataset during intermediate training.

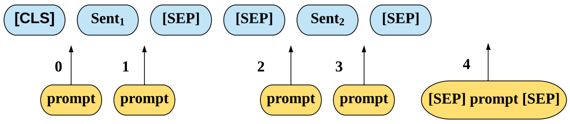


Figure 6: Insertion positions for sentence-pair tasks.

Architecture	**Avg**	**Voting**
PHM	86.1	86.9
+residual	85.9	86.7
+LayerNorm	86.1	87.1
+residual+LayerNorm	77.8	81.2

Table 7: Ablation study on generator architecture. We report average results and voting results across 5 runs.

A.5 Ablation study for single-layer IDPG

A.5.1 Generator Architecture Exploration

We explore three different architectures for the proposed PHM-based generator: (i) Residual: a residual structure (He et al., 2016) is applied to add the sentence representation to each generated tokens; (ii) LayerNorm: layer normalization (Ba et al., 2016) is also added to normalize the generated token embedding; (iii) residual + layerNorm: a mixed model that uses both the residual component and LayerNorm. Note that, to balance the token embedding and sentence embedding, we apply LayerNorm to each embedding first, then after the add-up, use LayerNorm again to control the generated tokens. We observe that adding LayerNorm slightly improves the voting results, while residual performs slightly worse. One surprising result is that the mixed model of Residual and LayerNorm has significantly poorer performance.

A.5.2 Prompt Position

As we discussed in Section A.4, the prompt position has a direct impact on the prediction results. We conduct a comprehensive study of the prompt position for our proposed method in both supplementary training and downstream fine-tuning phases.

Looking at the prompt position in downstream tasks first, Figure 7(a) shows that for both standard prompt tuning and our proposed method, the best position is 0 for single-sentence tasks and 1 for sentence-pair tasks. This result is intuitive for single-sentence tasks since prompt in position 0 can be regarded as the premise and original input sentence as the hypothesis. For sentence-pair tasks, we hypothesize that inserting prompt into position 1 can better align the two input sentences. Figure 7(b) illustrates the effect of prompt position on the supplementary training phase. It is interesting that IDPG achieves best results in position 0 while the standard prompt-tuning achieves the best results in position 4 for both single-sentence and sentence-pair tasks.

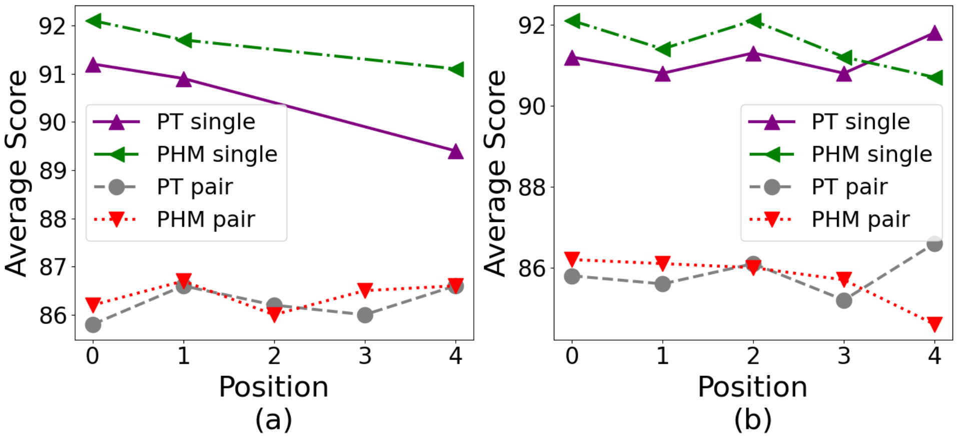
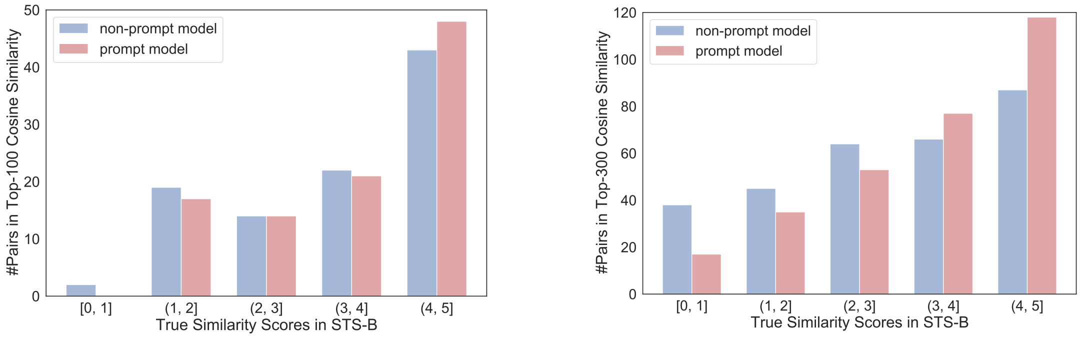


Figure 7: Impact of prompt position on (a) downstream tasks; (b) supplementary training phase.



(a) The number of pairs of each group in Top-100 cosine similarity ranking.

(b) The number of pairs of each group in Top-300 cosine similarity ranking.

Figure 8: The number of pairs of each group in Top-k cosine similarity ranking.

A.6 Cosine Similarity Distributions in STS-B

We present the cosine similarity distributions when $k = 100$ and $k = 300$ in Figure 8a and in Figure 8b, respectively.

A.7 Ablation Study on Prompt Length

We present the impact of prompt length among several prompt tuning methods in Figure 9. IDPG shows its stability when scaling to larger models with longer prompts.

A.8 Potential Risks

Our proposed model IDPG is a novel efficient transfer learning method. It tunes small portion parameters while directly employs backbone model parameters without any changing. However, if the backbone model stored online is attacked, whether IDPG could still work well remains unknown. One should be careful to apply our proposed model and all other prompt tuning methods in high-stakes areas without a comprehensive test.

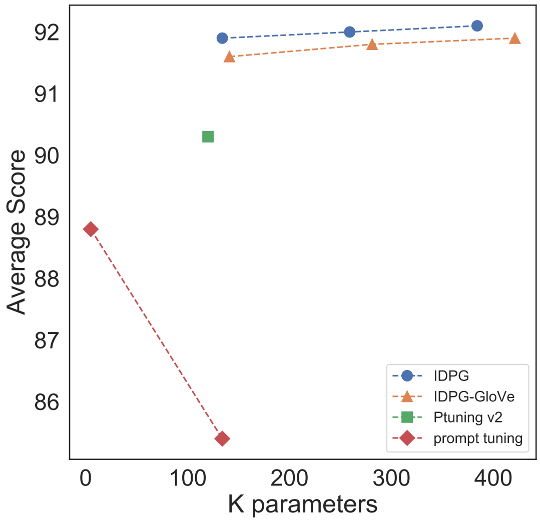


Figure 9: Impact of prompt length.