Hierarchical Prompt Tuning for Few-Shot Multi-Task Learning

Jingping Liu*
School of Information Science and Engineering, East China University of Science and Technology
Shanghai, China
jingpingliu@ecust.edu.cn

Tao Chen
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China
chentao20@fudan.edu.cn

Zujie Liang
Ant Group
Shanghai, China
liangzujie.lzj@antgroup.com

Haiyun Jiang
Yanghua Xiao*
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
Shanghai, China
jianghy16@fudan.edu.cn
shawyh@fudan.edu.cn

Feng Wei
Yuxi Qian
Ant Group
Shanghai, China
huodeng.wf@antgroup.com
qianyuxi.qyx@antgroup.com

Zhenghong Hao
Bing Han
Ant Group
Shanghai, China
haozhenghong.hzh@antgroup.com
hanbing.hanbing@antgroup.com

ABSTRACT

Prompt tuning has enhanced the performance of Pre-trained Language Models for multi-task learning in few-shot scenarios. However, existing studies fail to consider that the prompts among different layers in Transformer are different due to the diverse information learned at each layer. In general, the bottom layers in the model tend to capture low-level semantic or structural information, while the upper layers primarily acquire task-specific knowledge. Hence, we propose a novel hierarchical prompt tuning model for few-shot multi-task learning to capture this regularity. The designed model mainly consists of three types of prompts: shared prompts, auto-adaptive prompts, and task-specific prompts. Shared prompts facilitate the sharing of general information across all tasks. Auto-adaptive prompts dynamically select and integrate relevant prompt information from all tasks into the current task. Task-specific prompts concentrate on learning task-specific knowledge. To enhance the model's adaptability to diverse inputs, we introduce deep instance-aware language prompts as the foundation for constructing the above prompts. To evaluate the effectiveness of our proposed method, we conduct extensive experiments on multiple widely-used datasets. The experimental results demonstrate that the proposed method achieves state-of-the-art performance for multi-task learning in few-shot settings and outperforms ChatGPT in the full-data setting.

*Corresponding authors

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614913

CCS CONCEPTS

• **Information systems** → **Clustering and classification**.

KEYWORDS

Prompt Tuning, Few-shot Learning, Multi-task Learning

ACM Reference Format:
Jingping Liu, Tao Chen, Zujie Liang, Haiyun Jiang, Yanghua Xiao, Feng Wei, Yuxi Qian, Zhenghong Hao, and Bing Han. 2023. Hierarchical Prompt Tuning for Few-Shot Multi-Task Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614913

1 INTRODUCTION

Pre-trained language models (PLMs), e.g., BERT [11] and RoBERTa [17], have gained significant importance in the field of natural language processing (NLP). To adapt these general-purpose PLMs to specific downstream tasks, a common approach is to fine-tune them by updating all parameters. However, this method tends to yield poor results when dealing with a limited number of training samples, as the models become susceptible to over-fitting.

To address this challenge in low-resource scenarios, a technique called prompt tuning has been introduced to leverage PLMs for downstream tasks. Prompt tuning aims to incorporate prompt tokens into the input sequence and treat the downstream tasks as language modeling problems [22]. These prompt tokens can be discrete or continuous [13]. For instance, in sentiment analysis, we can concatenate the input sentence "I haven't published a paper" with a discrete prompt like "I felt [MASK]". We then ask the pre-trained language model to predict whether the masked token corresponds to option "happy" or "sad", thereby determining the input label. Alternatively, trainable continuous embeddings can be used to replace the discrete prompt, such as $[\boldsymbol{e}_0, ..., \boldsymbol{e}_l, \boldsymbol{e}([MASK])]$, where $\boldsymbol{e}_i$ is the unused token in PLM dictionary. In general, prompt tuning involves keeping the PLM parameters fixed while updating only the prompt representations [14]. Compared to fine-tuning, prompt tuning offers two distinct advantages. First, prompt tuning aligns more closely

with the pre-training objectives, such as masked language modeling, which facilitates better utilization of the knowledge encoded in PLMs and often leads to improved performance [13]. Second, as the number of learning parameters in prompt tuning is typically less than 1% of those in fine-tuning, it offers significant advantages in terms of training cost and time [15]. The successful utilization of prompts in few-shot learning has led to the emergence of various follow-up studies, such as continuous prompt encoding [1] and knowledgeable prompt learning [34]. These advancements further enhance the effectiveness and versatility of prompt tuning in different few-shot scenarios.

Recently, a few works focus on few-shot multi-task prompt tuning. The core idea behind this approach is to learn a specific prompt for each task and utilize these prompts to perform downstream tasks. For instance, ATTEMPT [1] initially trains task-specific prompts on high-resource tasks, and then utilizes an attention module to transfer the acquired prompt knowledge to different low-resource tasks. On the other hand, HyperPrompt [6] introduces a unified HyperNetwork that generates distinct prompts with different parameters for each task. However, these approaches fail to take into account the necessity of employing distinct prompts for different layers in the Transformer architecture, considering the varying information acquired at each layer in the model. In general, the bottom layers of the Transformer tend to capture low-level semantic or structural information that is commonly found in all natural language texts [4]. On the contrary, the upper layers in the model predominantly acquire task-specific knowledge [10]. Consequently, in the multi-task learning process, the prompts at the bottom layers of the model should be shared among different tasks, while the prompts at the upper layers need to be tailored individually for each specific task.

To address the above issues, we propose a novel hierarchical prompt tuning model for few-shot multi-task learning. Our model incorporates three distinct types of language prompts: shared prompts, auto-adaptive prompts, and task-specific prompts. Shared prompts serve to share general information across all tasks by utilizing the same prompt generation matrices. These prompts are positioned at the bottom layers of the pre-trained language model, facilitating the exchange of knowledge among tasks. Auto-adaptive prompts dynamically select and integrate relevant prompt information from all tasks into the current task. These prompts are accomplished through a task-level attention mechanism and gate layer, placed at the middle layers of the language model. By leveraging these prompts, our model can adaptively choose and incorporate useful information from other tasks. Task-specific prompts primarily focus on learning task-specific knowledge. They are implemented by introducing an independent prompt generation matrix for each task and are positioned at the upper layers of the language model. These task-specific prompts enable the model to concentrate on the unique requirements of each task, thereby improving model performance. To enhance the model's adaptability to diverse inputs, we introduce deep instance-aware language prompts as the foundation for constructing the aforementioned prompts. These prompts are derived by transforming the inputs using two down- and up-projection matrices. By incorporating instance-level information, the model gains a deeper understanding of the data, leading to improved performance. Furthermore, we design a progressive training strategy to comprehensively learn the prompts. This approach involves training each type of prompt sequentially, starting from the bottom layer and progressing to the upper layer. When learning a certain type of prompt, we ensure that the parameters of the learned prompts remain fixed to prevent the forgetting of prompt knowledge during the learning process.

Contributions. The contributions are summarised as follows:

- We propose a novel hierarchical prompt tuning framework for few-shot multi-task learning. The most significant characteristic of this framework lies in its ability to facilitate varying degrees of information sharing at different layers across multiple tasks by leveraging distinct types of prompts during the learning process.
- We introduce an adaptive approach to selectively extract task-relevant information from multiple tasks with a task-level attention mechanism and gate layer. In addition, we design a progressive training strategy to achieve efficient prompt tuning.
- We conduct extensive experiments on widely-used benchmark datasets. The experimental results demonstrate that our method achieves a state-of-the-art (SoTA) in the few-shot settings and also outperforms the large language model named ChatGPT in the full-data setting.

2 RELATED WORK

Related work in this paper can be divided into three groups: Prompt Tuning, Few-shot Learning and Multi-task Learning.

Prompt Tuning. Prompt tuning is a novel paradigm in adapting pre-trained language models to downstream tasks. It involves inserting additional prompt tokens into the input, effectively converting the downstream tasks into pre-training tasks. This technique can be classified into three categories based on the structure of prompt tokens: Discrete Language Prompt, Continuous Language Prompt, and Hybrid Language Prompt. In the case of Discrete Prompts, actual tokens in the PLM vocabulary are concatenated together to form prompts. LAMA [20] asks knowledge engineers to manually construct discrete prompts for different relations in triplets and allow the model to predict missing elements of triplets in a cloze manner. AutoPrompt [22] uses training samples to automatically search for prompt tokens, creating task-adapted language prompts. Continuous Prompt, on the other hand, replaces discrete prompts with trainable embeddings. P-Tuning-V2 [14] inserts trainable embeddings as language prompts into each layer of the language model, achieving comparable performance to fine-tuning. LPT [15] inserts instance-aware prompts in later layers of the language model, reducing the propagation distance between labels and prompt parameters to improve performance. Hybrid Prompt combines discrete and continuous prompts by inserting trainable tokens into actual tokens. P-Tuning [16] inserts several learnable embeddings into the combination of the head entity and its relation, and allows the PLM to predict the tail entity in a cloze way. S-Prompt [28] constructs hybrid prompts using image categories and trainable parameters, and fuses them with the input features to enhance the model performance in image classification tasks.

Few-shot Learning. Few-Shot Learning (FSL) refers to learning from a small number of training samples with supervised information [29]. In general, FSL can be divided into three paradigms: data argument learning, meta-learning, and transfer learning. Data argument learning aims at increasing the sample richness to improve the performance of FSL [23]. Random erasure [38] enhances the scale of existing data by introducing slight modifications to the data, thereby promoting diversity in model inputs. Gao [5] explores the underlying distribution of few-shot data and presents an adversarial covariance augmentation network to address FSL limitations. Meta-learning leverages prior knowledge from data and tasks to extract meta-knowledge that can be applied to future tasks [24]. Meta-GCN [2] is a gradient-based meta-learning framework that leverages higher-order gradients to tackle the task of few-shot link prediction. GFL [33] utilizes the prior knowledge generated from extra graphs to enhance the task accuracy on the target graph. Transfer learning [39] trains on high-resource tasks and then transfers knowledge to low-resource tasks. UPT [27] is to use the learned prompting knowledge from non-target NLP tasks to enhance the model performance on the target task. SUR [3] transfers the general representations acquired from diverse feature extractors trained in multiple domains to other domains during inference.

Multi-task Learning. Multi-Task Learning (MTL) [36] aims to enhance the generalization performance of multiple related tasks by sharing valuable information in each task. Broadly speaking, MTL can be categorized into two main types: Feature-based MTL and Parameter-based MTL. The former focuses on identifying common features across different tasks to facilitate knowledge sharing. By learning these shared features, tasks can benefit from each other's insights and improve their generalization capabilities. [12] proposes a novel adversarial multi-task learning framework, which aims to alleviate the interference among the shared and private latent feature spaces. UniT [9] proposes a Unified Transformer model that utilizes specialized encoders to encode each input modality and subsequently employs a shared decoder to generate predictions, followed by task-specific output heads. The latter is to use model parameters in a task to facilitate the learning process of model parameters in other tasks [36]. MRN [18] is proposed to alleviate the negative transfer phenomenon between different tasks by learning the relationships between tasks on multiple specific layers. HMNET [4] adopts a hierarchical architecture to facilitate information sharing among tasks within the same cluster, which is formed by clustering all pre-defined tasks.

3 OVERVIEW

In this section, we formally define the problem of few-shot multi-task learning and outline our solution for the problem.

3.1 Problem Definition

For a single task, given an input sentence x, the goal is to design a function $f(\cdot)$ to make a prediction on x and output its label y. This can be formally defined as:

$$f(x) \rightarrow y. \tag{1}$$

For multiple few-shot tasks, let $\{D_\tau\}_{\tau=1}^{T}$ be a set of T tasks. $D_\tau = \{(x_\tau^n, y_\tau^n)\}_{n=1}^{N_\tau}$ indicates the dataset of the τ-th task, which contains a limited number of N_τ samples and each sample consists of input text x_τ^n and its golden label y_τ^n. Few-shot multi-task learning aims to use a single backbone model M to learn corresponding function $f_\tau(\cdot)$ for each small dataset D_τ simultaneously, which is formally defined as:

$$f_\tau^M(x_\tau^n) \rightarrow y_\tau^n, \tag{2}$$

3.2 Framework

To enhance the effectiveness of information sharing across multiple tasks, we propose a hierarchical prompt tuning framework for few-shot multi-task learning. Our framework, as depicted in Figure 1, consists of three types of prompts: shared prompt, auto-adaptive prompt, and task-specific prompt. In addition, all three prompt types are developed with the support of an instance-aware prompt to ensure their effectiveness.

Instance-aware Prompt. This is designed at every layer in the Transformer [25] to generate the prompts that are tailored to the input sample. Specifically, we first design down- and up-projection matrices to transform the instance representation vector produced by the previous layer into two prompt vectors. These generated prompts are then inserted into the key and value vectors of the self-attention block at the current layer in Transformer, respectively. This instance-aware prompt would serve as the basis for the implementation of the following three prompts.

Shared Prompt. This prompt is designed to share low-level information across multiple tasks. Given a textual representation of an input sample, we consider the instance-aware prompt as the shared prompt. In this case, we use the same down- and up-projection matrices for multiple tasks to ensure that the generated prompt shares the information among all tasks. In addition, this prompt is inserted into the bottom layers of Transformer, as these layers are more effective at capturing low-level structural or semantic information [10].

Auto-Adaptive Prompt. This prompt is utilized to adaptively select the current prompt-related information from all tasks, ensuring the effective knowledge transfer process. Given the input representation obtained through the previous layers in the model, we utilize the instance-aware prompt to generate an individual prompt for each task with different projection matrices. We then adopt a task-level attention mechanism [19] to determine the importance of task prompts to the prompt of the current task and combine them based on their importance weight. To emphasize the prompt of the current task, we further utilize a gate layer to fuse it with the above-combined prompt, which is then inserted into the intermediate layers of Transformer.

Task-Specific Prompt. This prompt is designed individually for each task. After previous layers generate the textual representation, we consider the instance-aware prompt as the task-specific prompt. Similar to the auto-adaptive prompt, the projection matrices in each task's prompt are also distinct. This type of prompts would be added to the upper layers of Transformer because these layers often tend to learn task-specific knowledge [4].

After generating the final text representation using the prompts described above, we feed it into the mask prediction head of the corresponding task to determine its output label.

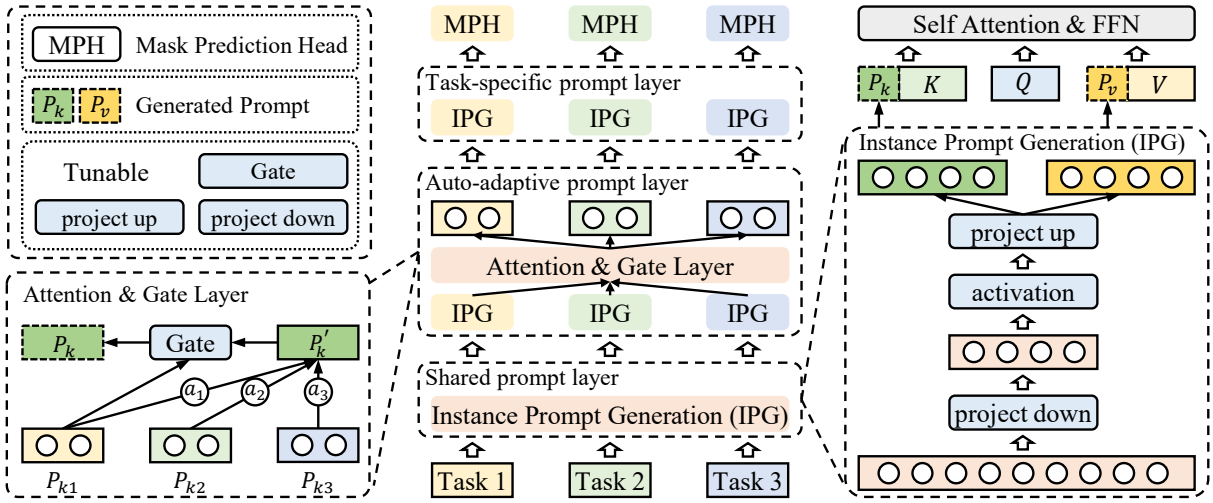

Figure 1: The framework of our solution for few-shot multi-task learning.

4 METHODOLOGY

In this section, we describe our solution for the few-shot multi-task learning in detail from five parts, including instance-aware prompt, shared prompt, auto-adaptive prompt, task-specific prompt, and mask prediction head.

4.1 Instance-aware Prompt

This prompt is adapted for each input sample and varies with different inputs. A straightforward way to implement it is to design a single transformation matrix $\boldsymbol{W} \in \mathbb{R}^{d_h \times d_h}$ to generate a distinct prompt for each input, where d_h is the dimensionality of the input representation. However, this method requires an enormous number of matrix parameters, since the input vector dimension is usually large (e.g., 1024 in BERT/RoBERTa-Large). Hence, to reduce the scale of parameters and training costs, we factorize the single matrix into two matrices for generating prompts.

Specifically, let $\boldsymbol{H}^{m-1} = [\boldsymbol{h}_1^{m-1}, \boldsymbol{h}_2^{m-1}, ..., \boldsymbol{h}_L^{m-1}]$ be the text representation generated by the previous layer, where the first and last are the representations of the special tokens $\langle CLS \rangle$ and $\langle SEP \rangle$ in the language model respectively and L denotes the sequence length. Note that the text representation of the first layer is obtained by initializing the input x through the token representations in the language model. We take the embedding of $\langle CLS \rangle$ as the input representation and transform it through the down-projection matrix, which is formally expressed as:

$$\boldsymbol{h}_{down} = \boldsymbol{W}_{down}\boldsymbol{h}_1^{m-1} + \boldsymbol{b}_{down}, \tag{3}$$

where $\boldsymbol{W}_{down} \in \mathbb{R}^{d_{down} \times d_h}$ and $\boldsymbol{b}_{down} \in \mathbb{R}^{d_{down}}$ are the trainable matrices, $\boldsymbol{h}_{down} \in \mathbb{R}^{d_{down}}$ is the representation after down projection at the m-th layer. Furthermore, we utilize an up-projection matrix to transform $\boldsymbol{h}_{down}$ to generate two prompts $\boldsymbol{P}_k$ and $\boldsymbol{P}_v$. This transformation can be formalized as:

$$\begin{aligned} \boldsymbol{P}_k &= \boldsymbol{W}_{up\_k}\boldsymbol{h}_{down} + \boldsymbol{b}_{up\_k}, \\ \boldsymbol{P}_v &= \boldsymbol{W}_{up\_v}\boldsymbol{h}_{down} + \boldsymbol{b}_{up\_v}, \end{aligned} \tag{4}$$

where $\boldsymbol{P}_k, \boldsymbol{P}_v \in \mathbb{R}^{d_l \times d_h}$. $\boldsymbol{W}_{up\_k}, \boldsymbol{W}_{up\_v} \in \mathbb{R}^{(d_l \times d_h) \times d_{down}}$ and $\boldsymbol{b}_{up\_k}, \boldsymbol{b}_{up\_v} \in \mathbb{R}^{d_l \times d_h}$ are the trainable up-projection matrices. d_l denotes the length of prompts.

After generating the instance-aware prompts $\boldsymbol{P}_k$ and $\boldsymbol{P}_v$, we insert them into the m-th layer in the Transformer to improve the encoding ability. P-Tuning V2 [14] employs deep prompt tuning and inserts deep prompt into the self-attention block of Transformer [25], which achieves similar performance to fine-tuning. Similar to P-Tuning V2, we also insert $\boldsymbol{P}_k$ and $\boldsymbol{P}_v$ into the self-attention block in the m-th layer of Transformer. Specifically, we denote $\boldsymbol{Q} \in \mathbb{R}^{L \times d_h}$, $\boldsymbol{K} \in \mathbb{R}^{L \times d_h}$, and $\boldsymbol{V} \in \mathbb{R}^{L \times d_h}$ as the query, key, and value vectors in self-attention, which are transformed from the input representation. Then, we concatenate $\boldsymbol{P}_k$ and $\boldsymbol{K}$, $\boldsymbol{P}_v$ and $\boldsymbol{V}$, respectively, which is defined as

$$\begin{aligned} \boldsymbol{K}' &= concat(\boldsymbol{P}_k, \boldsymbol{K}), \\ \boldsymbol{V}' &= concat(\boldsymbol{P}_v, \boldsymbol{V}), \end{aligned} \tag{5}$$

where $\boldsymbol{K}', \boldsymbol{V}' \in \mathbb{R}^{(d_l+L) \times d_h}$ are the new key and value vectors. Based on these vectors, the self-attention in Transformer can be refined as:

$$\boldsymbol{X} = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}'^T}{\sqrt{d}})\boldsymbol{V}', \tag{6}$$

where $\boldsymbol{X}$ is the input representation encoded by the self-attention module.

4.2 Shared Prompt

This prompt is designed to share general information among all tasks, such as low-level structural or semantic information. We generate the shared prompt based on the instance-aware prompt. Specifically, given the text representation produced by the previous layer, we first transform it with a down-projection matrix $\boldsymbol{W}_{down}^{sp}$ shared by all tasks, which is formally expressed as:

$$\boldsymbol{h}_{down}^{sp} = \boldsymbol{W}_{down}^{sp}\boldsymbol{h}_1^{m-1} + \boldsymbol{b}_{down}^{sp}. \tag{7}$$

Then, we further transform $\boldsymbol{h}_{down}^{sp}$ by introducing shared up-projection matrices to generate prompts $\boldsymbol{P}_k^{sp}$ and $\boldsymbol{P}_v^{sp}$, as shown below.

$$\begin{aligned}\boldsymbol{P}_k^{sp} &= \boldsymbol{W}_{up\_k}^{sp}\boldsymbol{h}_{down}^{sp} + \boldsymbol{b}_{up\_k}^{sp},\\ \boldsymbol{P}_v^{sp} &= \boldsymbol{W}_{up\_v}^{sp}\boldsymbol{h}_{down}^{sp} + \boldsymbol{b}_{up\_v}^{sp}.\end{aligned} \tag{8}$$

Finally, the prompts are concatenated with the key and value vector according to Eq. (11) and involved in the computation of self-attention. Note that the shared prompts are inserted into the bottom layers of Transformer. In addition, all Transformer layers using the shared prompt share the same down- and up-projection matrices.

4.3 Auto-Adaptive Prompt

This is utilized to dynamically select task prompt information that is helpful for the input sample of the current task. Under few-shot multi-task setting, it is an important strategy to transfer information from related tasks to the current task for enhancing model performance. Furthermore, we believe that information from different tasks contributes inconsistently to the current task. Hence, we adopt a task-level attention mechanism to measure the importance of the prompts of different tasks to the input and combine the task prompts based on their importance weights.

First, let $\boldsymbol{h}_1^{m-1}$ be the text representation of the current task, which is generated by the previous layer. We introduce individual down- and up-projection matrices for each task and obtain prompts based on these matrices and $\boldsymbol{h}_1^{m-1}$. The process can be formalized as:

$$\boldsymbol{h}_{down}^{ap\_i} = \boldsymbol{W}_{down}^{ap\_i}\boldsymbol{h}_1^{m-1} + \boldsymbol{b}_{down}^{ap\_i}, \tag{9}$$

$$\begin{aligned}\boldsymbol{P}_k^{ap\_i} &= \boldsymbol{W}_{up\_k}^{ap\_i}\boldsymbol{h}_{down}^{ap\_i} + \boldsymbol{b}_{up\_k}^{ap\_i},\\ \boldsymbol{P}_v^{ap\_i} &= \boldsymbol{W}_{up\_v}^{ap\_i}\boldsymbol{h}_{down}^{ap\_i} + \boldsymbol{b}_{up\_v}^{ap\_i},\end{aligned} \tag{10}$$

where $\boldsymbol{W}_{down}^{ap\_i}$, $\boldsymbol{W}_{up\_k}^{ap\_i}$, $\boldsymbol{W}_{up\_v}^{ap\_i}$, $\boldsymbol{b}_{down}^{ap\_i}$, $\boldsymbol{b}_{up\_k}^{ap\_i}$, and $\boldsymbol{b}_{up\_v}^{ap\_i}$ are the down- and up- projection matrices of the i-th task. $\boldsymbol{P}_k^{ap\_i}$ and $\boldsymbol{P}_v^{ap\_i}$ are the prompts obtained by passing the text representation through the matrices of the i-th task.

Second, we take $\boldsymbol{P}_k^{ap}$ as an example to detail the task-level attention mechanism. Given $\boldsymbol{h}_1^{m-1}$ and the prompt set $\mathcal{P}_k^{ap} = \{\boldsymbol{P}_k^{ap\_i}\}_{i=1}^T$, the task-level attention mechanism aims to compute the alignment score between $\boldsymbol{h}_1^{m-1}$ and $\boldsymbol{P}_k^{ap\_i}$ with the function $g(\boldsymbol{P}_k^{ap\_i}, \boldsymbol{h}_1^{m-1}) = \frac{\boldsymbol{P}_k^{ap\_i}\cdot(\boldsymbol{h}_1^{m-1})^T}{\sqrt{d_h}}$, which shows the attention of $\boldsymbol{P}_k^{ap\_i}$ on $\boldsymbol{h}_1^{m-1}$. A softmax function is then unitized to convert the alignment score into a probability distribution, i.e., $a(\boldsymbol{P}_k^{ap\_i}|\mathcal{P}_k^{ap}, \boldsymbol{h}_1^{m-1})$. A small $a(\boldsymbol{P}_k^{ap\_i}|\mathcal{P}_k^{ap}, \boldsymbol{h}_1^{m-1})$ indicate a weak correlation between the i-th task and the input sample of the current task. This process can be defined as:

$$\begin{aligned}&a(\boldsymbol{P}_k^{ap\_i}|\mathcal{P}_k^{ap}, \boldsymbol{h}_1^{m-1}) = Softmax(h),\\ &h = [g(\boldsymbol{P}_k^{ap\_i}, \boldsymbol{h}_1^{m-1})]_{i=1}^T.\end{aligned} \tag{11}$$

Hence, by combining Eq. (11) and the prompt set $\{\boldsymbol{P}_k^{ap\_i}\}_{i=1}^T$, we have

$$\boldsymbol{P}_k^{\hat{ap}\_\tau} = \sum_{i=1}^{T} a(\boldsymbol{P}_k^{ap\_i}|\mathcal{P}_k^{ap}, \boldsymbol{h}_1^{m-1})\boldsymbol{P}_k^{ap\_i}, \tag{12}$$

where $\boldsymbol{P}_k^{\hat{ap}\_\tau}$ is the attention prompt of the input sample in the τ-th task.

In order to highlight the importance of the prompt of the current task, we further combine the attention prompt $\boldsymbol{P}_k^{\hat{ap}\_\tau}$ with the target task prompt $\boldsymbol{P}_k^{ap\_\tau}$ obtained through Eq. (10). A common practice is to utilize a hyperparameter to combine the above two prompts, as shown below.

$$\boldsymbol{P'}_k^{ap\_\tau} = \alpha\boldsymbol{P}_k^{ap\_\tau} + (1-\alpha)\boldsymbol{P}_k^{\hat{ap}\_\tau}, \tag{13}$$

where $0 < \alpha < 1$. However, this approach has two problems. One is that it is difficult to determine the optimal hyperparameter in advance. Another is that the hyperparameter corresponding to different inputs in the same task may be inconsistent. To this end, we introduce a gate layer [32] for prompt combination. The gate layer is used to automatically adjust the importance of the two types of prompts, which is formulated as:

$$\begin{aligned}\boldsymbol{P}_g &= concat(\boldsymbol{P}_k^{ap\_\tau}, \boldsymbol{P}_k^{\hat{ap}\_\tau}),\\ \boldsymbol{g} &= \sigma(\boldsymbol{W}_1(\boldsymbol{W}_2\boldsymbol{P}_g + \boldsymbol{b}_2) + \boldsymbol{b}_1),\\ \boldsymbol{P'}_k^{ap\_\tau} &= \boldsymbol{g}\odot\boldsymbol{P}_k^{ap\_\tau} + (1-\mathbf{g})\odot\boldsymbol{P}_k^{\hat{ap}\_\tau},\end{aligned} \tag{14}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d_h\times d_{down}}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_h}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_{down}\times d_h}$, $\boldsymbol{b}_2 \in \mathbb{R}^{d_{down}}$ are trainable parameters. σ is the Sigmoid activate function used to limit the weight between 0 and 1. $\odot$ refers to the element-wise product. Similarly, we obtain $\boldsymbol{P'}_v^{ap\_\tau}$ through the above process. Finally, we incorporate $\boldsymbol{P'}_k^{ap\_\tau}$ and $\boldsymbol{P'}_v^{ap\_\tau}$ into the self-attention block of Transformer through Eq. (11). Notably, the auto-adaptive prompt is inserted into the middle layers of the Transformer.

4.4 Task-Specific Prompt

This prompt is specially designed for each task and aims to focus on learning the unique information of each task. Specifically, for the τ-th task, we design two down- and up-project matrices that are not shared with other tasks. Based on the matrices, we obtain the prompt of the input sample in the task, which is defined as

$$\boldsymbol{h}_{down}^{tp} = \boldsymbol{W}_{down}^{tp}\boldsymbol{h}_1^{m-1} + \boldsymbol{b}_{down}^{tp}, \tag{15}$$

$$\begin{aligned}\boldsymbol{P}_k^{tp} &= \boldsymbol{W}_{up\_k}^{tp}\boldsymbol{h}_{down}^{tp} + \boldsymbol{b}_{up\_k}^{tp},\\ \boldsymbol{P}_v^{tp} &= \boldsymbol{W}_{up\_v}^{tp}\boldsymbol{h}_{down}^{tp} + \boldsymbol{b}_{up\_v}^{tp},\end{aligned} \tag{16}$$

where $\boldsymbol{W}_{down}^{tp}$, $\boldsymbol{W}_{up\_k}^{tp}$, $\boldsymbol{W}_{up\_v}^{tp}$, $\boldsymbol{b}_{down}^{tp}$, $\boldsymbol{b}_{up\_k}^{tp}$, and $\boldsymbol{b}_{up\_v}^{tp}$ are the trainable matrices of the τ-th task. Note that this type of prompt is inserted into the self-attention block of top layers in Transformer since these layers tend to learn task-specific knowledge.

4.5 Mask Prediction Head

In this section, we design a Mask Prediction Head for each task to predict the labels of the input samples. To fully leverage the capabilities of the language model, we transform the label prediction problem into a masked prediction task [16]. This is achieved by designing a template for each task type and inserting the input x into it. The language model is then asked to predict the true label of $[MASK]$. For instance, Table 1 provides the templates for natural language inference (NLI), sentiment classification (SC), similarity and paraphrase (SP) tasks in GLUE [26], along with their original

Table 1: Template and label mappings for each task. S_1 **and** S_2 **refer to the input sentences in the tasks.**

Task	Template	Label Mapping
NLI	$<S_1>$? [MASK] $<S_2>$	Entailment: Yes, Neutral: Maybe, Contradiction: No
SC	$<S_1>$. It was [MASK]	Positive: Great, Negative: Terrible
SP	$<S_1>$. [MASK] $<S_2>$	Equivalent: Yes, Not equivalent: No

labels and the mapping tokens. These tokens would be filled in the [MASK] position.

After substituting the input into the template, we use the language model to obtain its hidden state $\boldsymbol{H} = [\boldsymbol{h}_1, ..., \boldsymbol{h}_{[MASK]}, ..., \boldsymbol{h}_L]$, where $\boldsymbol{h}_{[MASK]} \in \mathbb{R}^{d_h}$ used to predict the final result. Hence, we have

$$\Pr = softmax(\boldsymbol{W}_{mask}\boldsymbol{h}_{[MASK]} + \boldsymbol{b}_{mask}), \tag{17}$$

where $\boldsymbol{W}_{mask} \in \mathbb{R}^{d_v \times d_h}$ and $\boldsymbol{b}_{mask} \in \mathbb{R}^{d_v}$ are the frozen weight in the pre-trained language model, d_v is the vocabulary size of language model, and Pr is the probability of prediction result. Finally, we adopt cross-entropy as the training loss of the proposed model.

5 TRAINING DETAILS

A straightforward way to train hierarchical prompts is to feed data to the model and update all parameters in the prompts at once. However, this approach is unreasonable as it assumes that all parameters have the same utility. In fact, the parameters of different layers have varying effects. For instance, parameters at the bottom layers are used for sharing across tasks, while those at the upper layers are used for learning specific tasks. Hence, in this paper, we design a progressive training strategy for the proposed method, as illustrated in Algorithm 1.

Our training strategy consists of three stages. First, we input all task data into the model and train shared prompts through multi-task learning. After training, we freeze their parameters to ensure that the learned general information is not forgotten. Second, we feed the data from all tasks into the model again to train auto-adaptive prompts. Once trained, we also fix their parameters, preserving the ability of different tasks to adaptively select information from different prompts. Finally, for each task, we input its sample data into the model to train the corresponding task-specific prompt. This task-by-task training allows the model to focus on a single task, thereby fully improving its performance. During this process, all other task-specific prompts will not be updated. Additionally, in the training process, we freeze the parameters of the language model and only update the parameters in the prompts.

During the training phase for various prompts, different training data is utilized. In the case of the shared and auto-adaptive prompts, each dataset D_τ is initially partitioned into batches of data $B_\tau = \{b_1, b_2, ..., b_M\}$. Subsequently, the batch sets for all tasks are combined to form $B = \{B_\tau\}_{\tau=1}^{T}$, which is then randomly shuffled in batches. For task-specific language prompts, the current task's dataset $D\tau$ is divided into batches of data $B_\tau = \{b_1, b_2, ..., b_M\}$ and randomly shuffled in batches.

Algorithm 1: Progressive Training Process

Input : Datasets $\{D_\tau\}_{\tau=1}^{T}$, Model M
Output: The trained model M

Initial $\boldsymbol{W}^{sp}$, $\{\boldsymbol{W}_\tau^{ap}\}_{\tau=1}^{T}$, $\{\boldsymbol{W}_\tau^{tp}\}_{\tau=1}^{T}$,
// *Stage* 1 : *Training shared prompt*
Split $\{D_\tau\}_{\tau=1}^{T}$ into batch set $\mathcal{B}$,
for $b \in \mathcal{B}$ **do**
  Input b into M,
  Compute loss and update $\boldsymbol{W}^{sp}$, $\{\boldsymbol{W}_\tau^{ap}\}_{\tau=1}^{T}$, and $\{\boldsymbol{W}_\tau^{tp}\}_{\tau=1}^{T}$,
end
Freeze $\boldsymbol{W}^{sp}$,
// *Stage* 2 : *Training auto-adaptive prompts*
Split $\{D_\tau\}_{\tau=1}^{T}$ into batch set $\mathcal{B}$,
for $b \in \mathcal{B}$ **do**
  Input b into M,
  Compute loss and update $\{\boldsymbol{W}_\tau^{ap}\}_{\tau=1}^{T}$ and $\{\boldsymbol{W}_\tau^{tp}\}_{\tau=1}^{T}$,
end
Freeze $\{\boldsymbol{W}_\tau^{ap}\}_{\tau=1}^{T}$,
// *Stage* 3 : *Training task-specific prompts*
for $D_i \in \{D_\tau\}_{\tau=1}^{T}$ **do**
  Split D_i into $\mathcal{B}_i$,
  for $b \in \mathcal{B}_i$ **do**
    Input b into M,
    Compute loss and update $\boldsymbol{W}_i^{tp}$,
  end
end

6 EXPERIMENTS

In this section, we first conduct extensive experiments to evaluate the effectiveness of our proposed methods. Then, we perform ablation experiments to analyze the effect of each key component in our method. Finally, we provide a detailed analysis of our method.

6.1 Experimental Setup

Datasets. Following LPT [15], we evaluate our method on six text classification datasets from GLUE [26]. Since the golden labels are not available in the test set, we make the same adjustments as in LPT: 1) The development set is used as the test set. 2) 1,000 samples are selected from the original training set as the development set according to the label proportion, and the remaining data is used as the training set. In the few-shot scenario, we randomly select 100 and 500 samples in the training set to train our model, respectively. The statistics of these datasets are listed in Table 2.

Baselines. We compare our models with the following baselines:

- **Adapter-based methods. Adapter** [7] add an adapter module in each layer of the Transformer and only train the parameters of the adapter module to solve classification tasks. **AdapterDrop** [21] proposes the method of dynamically selecting and removing adapter modules during training and inference to improve efficiency while maintaining overall task performance.

Table 2: The statistics of text classification datasets.

Dataset	Task Type	Train	Dev	Test
MNLI	NLI	392,702	19,647	19,643
QNLI	NLI	104,743	5,463	5,463
RTE	NLI	2,490	277	3,000
SST-2	SC	67,349	872	1,821
QQP	SP	363,846	40,430	390,965
MRPC	SP	3,668	408	1,725

Table 3: Results on the few-shot scenario (100 samples).

Models	MNLI	QNLI	RTE	SST-2	QQP	MRPC	AVG
Model-tuning	51.5	73.9	48.6	89.6	71.6	78.3	68.9
Adapter	42.9	61.5	52.0	90.8	67.3	77.9	65.4
AdapterDrop	41.0	60.4	50.4	87.8	64.7	76.6	63.4
BitFit	42.0	60.3	50.8	91.8	64.9	77.0	64.4
LoRA	48.1	65.9	51.0	91.0	69.7	78.5	67.3
P-Tuning	47.3	55.8	59.6	90.0	52.7	74.2	63.2
P-Tuning-V2	37.3	54.2	54.9	89.4	60.7	75.0	61.9
S-IDPG-PHM	47.8	56.9	59.8	90.5	54.5	75.2	64.1
LPT-NPG	64.3	75.7	68.1	92.7	**74.6**	**80.6**	76.0
Ours	**71.9**	**81.2**	**73.6**	**93.2**	73.5	78.4	**78.6**

- **Prompt-based methods. P-Tuning** [16] and **P-Tuning-V2** [14] add trainable embedding as the continuous prompt for each task. **S-IDPG-PHM** [31] and **LPT-NPG** [15] are proposed to generate an instance-aware prompt for each input sample.
- **Others. Model-tuning** refers to adapting downstream tasks by updating all parameters of the PLM backbone. **BitFit** [35] freezes most of the model's parameters and only adjusts the bias terms and task-specific classification head parameters. **LoRA** [8] add low-rank decomposition matrices between full connection layers to reduce the number of learning parameters and improve model performance in tasks.

Metrics. Referring to the settings in LPT [15], we report *accuracy* for NLI and SC tasks, *acc_and_F1* for SP tasks, respectively. Note that *acc_and_F1* means the average of accuracy and F1.

Parameter Settings. For the pre-trained language model, we used RoBERTa-large [17] in our approach. For the dimension setting, we set $d_{down} = 64$, $d_h = 1,024$, $d_l = 5$, $d_v = 50,264$, where d_v is the vocabulary size in RoBERTa. During training, we use AdamW as the optimizer. We search the learning rate from $5e$-4 to $1e$-2 with 10% warming up steps. After warming up steps, we linearly decayed the learning rate to zero. For the few-shot setting, we train shared and auto-adaptive prompts for 10 epochs respectively, and 100 epochs for task-specific prompts. For the full-data setting, we train shared and auto-adaptive prompts for 3 epochs respectively, and 10 epochs for task-specific prompts. The batch size is set as 16. The max length of input sentences is set as 452. We save the best model's weights that achieve the best score in the development set. For the reported results, we run 3 times with different random seeds and take their average as the final results. All implement code of our models is based on the HuggingFace's Transformers [30] library and the model is trained on GeForce RTX 3090 GPU with 24GB of memory.

Table 4: Results on the few-shot scenario (500 samples).

Models	MNLI	QNLI	RTE	SST-2	QQP	MRPC	AVG
Model-Tuning	75.3	85.2	67.0	91.4	77.3	85.1	80.2
Adapter	70.5	78.0	67.5	92.0	72.1	83.6	77.2
Adapter-Drop	66.1	78.9	62.0	91.2	73.4	82.5	75.6
BitFit	73.0	80.4	59.0	92.2	75.6	83.5	77.2
LoRA	74.5	82.5	62.8	92.1	76.4	84.1	78.7
P-Tuning	45.5	58.0	61.2	91.1	52.6	74.6	63.8
P-Tuning-V2	61.6	73.7	56.0	91.3	71.7	76.6	71.8
S-IDPG-PHM	46.3	59.4	64.7	91.3	56.4	75.1	65.5
LPT-NPG	76.0	83.2	74.7	92.6	**77.9**	**81.4**	80.9
Ours	**78.7**	**85.0**	**78.7**	**93.8**	76.7	80.1	**82.1**

6.2 Main results

In order to verify the effectiveness of our method on the few-shot scenario, we compare the method with all baselines using 100 and 500 training samples. The results are reported in Tables 3 and 4.

According to the tables, we observe that: 1) Our method outperforms all baseline models on most datasets, demonstrating its effectiveness in the few-shot scenarios. On average, our method achieves a 2.6% and 1.2% improvement over the best baseline method (i.e., LPT-NPG) on the settings of 100 and 500 training samples, respectively. It's worth noting that our model performs slightly worse than LPT-NPG on QQP and MRPC, but achieves better overall results. 2) Our method has a significant improvement over Model-tuning, particularly when trained with fewer samples. For instance, our method outperforms it by an impressive 9.7% on average results with only 100 training samples. This is because Model-tuning updates the entire language model parameters and is prone to over-fitting when training samples are insufficient. 3) In comparing prompt-based methods, the models based on the instance-aware prompt (S-IDPG-PHM, LPT-NPG, and our method) perform better overall than the methods that rely on task-based prompts (P-Tuning and P-Tuning-V2). This indicates that in the few-shot scenario, prompts generated based on input samples are more adaptable to downstream tasks, thereby improving the model performance. 4) Among the models that use instance-aware prompts, our method achieves the best performance, which highlights the effectiveness of the hierarchical prompt structure in transferring knowledge from other tasks to the current task in multi-task learning.

6.3 Ablation Study

To further analyze the impact of each component in our method, we perform an ablation study with 100 training samples. Specifically, we sequentially eliminate one component and evaluate the effectiveness of the remaining component combinations. Notably, when

Table 5: Results of ablation study.

Models	MNLI	QNLI	RTE	SST-2	QQP	MRPC	AVG
w/o IP	57.6	61.7	64.9	92.4	67.6	69.7	68.9
w/o SP	68.2	59.1	66.7	91.6	63.2	73.1	70.3
w/o AP	66.3	69.3	68.5	92.3	72.3	72.1	73.4
w/o Attention	68.2	79.1	68.9	91.9	74.5	74.4	76.1
w/o Gate layer	70.7	77.4	70.7	92.7	73.2	75.0	76.6
Ours	**71.9**	**81.2**	**73.6**	**93.2**	**73.5**	**78.4**	**78.6**

the instance-aware prompt is removed, we assign the same prompt to all input samples across each task. In the absence of shared prompts, we replace them with auto-adaptive prompts. Similarly, if auto-adaptive prompts are removed, we consider shared prompts as a replacement. In addition, it is necessary to retain task-specific prompts during the training phase in order to save model weights for each task. Hence, in this section, we do not remove task-specific prompts. These results are reported in Table 5.

From the results, we have the following conclusions. 1) Removing any component led to a decrease in performance, indicating that each component contributes positively to our model. 2) The instance-aware prompt has the most significant impact on our model's performance, as removing it causes the greatest performance drop. This is because the other three prompt types rely on this instance-aware prompt as their foundation. 3) Removing shared prompts led to a greater performance drop than removing auto-adaptive prompts. This suggests that shared prompts are more beneficial in enhancing the model's performance by providing shared information across different tasks. 4) Removing the attention mechanism or gate layer results in less significant performance drops compared to other modules. This is due to these sub-modules having relatively low parameter weights in auto-adaptive prompts. 5) The SST-2 task's performance remains stable across all scenarios. The possible reason is that it's a low-difficulty single-sentence classification task that the model can solve well, as evidenced by the results presented in Section 6.2, even in a few-shot scenario or with some modules removed.

6.4 Detail Analysis

In this section, we study the impact of prompt length, different layer combinations, and attention layer on the setting of 100 training samples. Besides, we further analyze the efficiency of our model and all baselines, respectively. Furthermore, we compare the proposed method with a large language model (LLM) named ChatGPT.

Impact of Prompt Length. To study the impact of prompt length on model performance, we perform our proposed method with prompt lengths ranging from 2 to 9 (i.e., $d_l \in [2, 9]$). The results are presented in Figure 2, which displays the average performance and number of trainable parameters with different prompt lengths. As illustrated in the figure, increasing the length of the prompt results in a corresponding increase in the number of trainable parameters. When the prompt length is less than 5, the model's performance improves with longer prompts. This is because longer prompts enable the model to learn richer information through more adjustable parameters, which ultimately enhances performance.

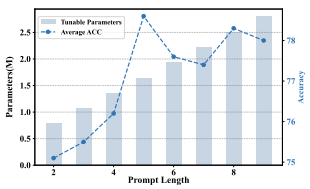

Figure 2: Results with different prompt lengths.

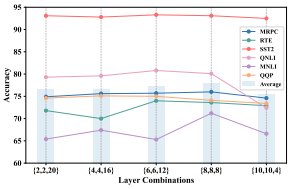

Figure 3: Results with different layer combinations.

However, when the prompt length exceeds 5, the model's performance becomes erratic, hovering around 78.0% due to the limited availability of training resources in the few-shot scenario. With an increase in the number of trainable parameters, the training samples become insufficient to effectively adjust the parameters, resulting in no significant improvement in performance. Therefore, in order to maintain a balance between model performance and the number of parameters, a prompt length of 5 is chosen for our proposed method.

Impact of Different Layer Combinations. In RoBERTa-large, there are a total of 24 transformer layers. In order to analyze the impact of three different prompts occupying different layers, we conduct experiments with varying layer combinations, including $\{\{2, 2, 20\}, \{4, 4, 16\}, \{6, 6, 12\}, \{8, 8, 8\}, \{10, 10, 4\}\}$, where A, B, and C in $\{A, B, C\}$ represent the number of transformer layers occupied by the shared prompts, auto-adaptive prompts, and task-specific prompts, respectively. The results are depicted in Figure 3, which shows the results of all datasets with different layers combinations. From the results, we have the following conclusions: 1) The results of SST-2, MRPC, and QQP datasets show minimal variations in results across different layer combinations, while the MNLI, RTE, and QNLI datasets display significant fluctuations. This indicates that different datasets require different optimal layer combinations. 2) The overall trend for all datasets reveals an initial increase followed by a decrease in average results. Notably, the best average performance is achieved with the layer combination of $\{8, 8, 8\}$. This suggests that the best overall result is obtained when the number of transformer layers for the three prompts is evenly distributed. Hence, we determine $\{8, 8, 8\}$ as the final combination in our experiments.

Impact of Attention Layer. In Auto-adaptive prompts, we introduce an attention mechanism to automatically select the information that is helpful to the input of the current task from the prompts of other tasks. In order to further analyze the interdependence of different tasks, we visualize the attention weights. Figure 4 illustrates the average attention weights of other datasets to all samples of the current dataset. From the results, we observe that the RTE, QQP, and QNLI datasets exhibit higher attention weights (darker colors), indicating that they have a greater impact on other datasets. Conversely, the SST-2 dataset shows the least influence on other datasets, as evidenced by its lighter color. This may be because this dataset focuses on the sentiment classification of individual sentences, which presents relatively low prediction difficulty and provides less beneficial information for other task datasets.

Efficiency Evaluation. To assess the efficiency of our method, we compared it with other baselines using 100 training samples.

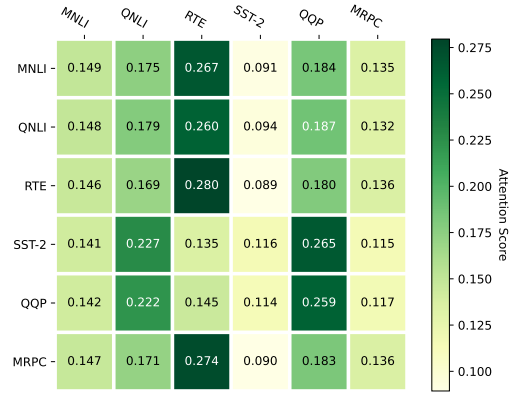

Figure 4: Attention weights in auto-adaptive prompts.

Table 6: Training efficiency and memory cost on RTE.

Models	Accuracy	Training Speed tokens/ms (↑)	Memory Cost GB (↓)
Model Tuning	52.0	11.6	23.5
Adapter	50.3	15.5 (1.3x)	16.5 (29.8%)
AdapterDrop	49.4	21.6 (1.9x)	**9.5 (59.6%)**
BitFit	50.2	16.5 (1.4x)	15.7 (33.2%)
LoRA	20.1	16.4 (1.4x)	16.2 (31.1%)
P-Tuning	58.2	16.9 (1.5x)	17.8 (24.3%)
P-Tuning-V2	53.2	19.2 (1.7x)	16.8 (28.8%)
S-IDPG-PHM	58.8	12.0 (1.0x)	16.8 (28.5%)
LPT-NPG	69.5	**23.2 (2.0x)**	10.1 (56.6%)
Ours	**73.8**	17.8 (1.5x)	14.6 (37.8%)

The experiments are conducted on a GeForce RTX 3090 GPU with 24GB of memory, and the results are summarized in Table 6. The table shows the accuracy, training speed (measured in tokens per millisecond), and memory cost. We also provide the improvement speed ratio and memory saving ratio of other models in comparison to the Model-Tuning baseline. From the table, we observe that: 1) Although the proposed method may not be the fastest or most cost-effective in terms of training speed and memory usage, it still outperforms the majority of other methods and achieves the best performance, indicating a good balance is kept between efficiency and effectiveness. 2) Our method is slightly less efficient than P-tuning-V2, which also adds prompts to key and value. This is because the prompts of each layer in our method are generated automatically based on the input, which is more time-consuming. However, we believe that this trade-off is worth it because our method produces better results. 3) Our method's memory savings are second only to LPT-NPG and AdapterDrop. We achieve this by using shorter prompt lengths and shared prompt generative matrices, which reduce memory costs to some extent.

Comparison with ChatGPT. Recently, LLMs such as ChatGPT, have demonstrated impressive performance in various NLP tasks. In order to verify the effectiveness of the approach proposed in this paper, we compare it with ChatGPT[1], referring to the experimental results of ChatGPT derived from [37]. From the table, we make the following observations: 1) ChatGPT achieves relatively good results on text classification datasets, slightly surpassing our model trained on few-shot settings. This could be attributed to the larger parameter scale and pre-training corpus size of ChatGPT compared to our usage of RoBERTa. Additionally, ChatGPT's training corpus, being non-public, may potentially include our classification task's test data. 2) Our model, trained on the full data, outperforms ChatGPT. Specifically, our model demonstrates an average improvement of 6.4% over ChatGPT. It is important to note two problems when applying ChatGPT in practical scenarios. First, the source code for ChatGPT is not publicly available, and it can only be accessed through the API interface. This limitation raises concerns about the potential leakage of sensitive corporate information. Second, the stability of ChatGPT's API interface calls is not entirely reliable, and there is a cost associated with using it. Consequently, when confronted with large-scale test data, both time and monetary expenses can become substantial. Considering these factors, the open-source few-shot multi-task learning model we propose becomes essential and highly beneficial.

Table 7: Comparison results with ChatGPT.

Models	MNLI	QNLI	RTE	SST-2	QQP	MRPC	AVG
ChatGPT	81.3	84.0	**88.0**	92.0	79.3	72.1	82.7
100 Samples	71.9	81.2	73.6	93.2	73.5	78.4	78.6
500 Samples	78.7	85.0	78.7	93.8	76.7	80.1	82.1
Full data	**88.6**	**93.4**	80.8	**95.4**	**88.3**	**88.5**	**89.1**

7 CONCLUSION

In this paper, we propose a novel hierarchical prompt tuning framework for few-shot multi-task learning. Our framework comprises three key components: shared prompts, auto-adaptive prompts, and task-specific prompts. The shared prompts facilitate the sharing of general information by utilizing shared generation matrices. The auto-adaptive prompts leverage attention mechanisms and gate layers to dynamically adjust and incorporate relevant information from various tasks into the current task. The task-specific prompts prioritize task-specific information through distinct prompt generation matrices. Furthermore, we introduce deep instance-aware prompts to enhance the adaptability of our prompts to various inputs. We evaluate our method on multiple publicly available datasets, showing superior performance over baseline models for few-shot multi-task learning.

ACKNOWLEDGMENTS

This work was supported by Ant Group Research Fund, Shanghai Sailing Program (No. 23YF1409400), Science and Technology Commission of Shanghai Municipality Grant (No. 22511105902), and Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0103).

[1]https://chat.openai.com/

REFERENCES

[1] Akari Asai, Mohammadreza Salehi, Matthew E Peters, and Hannaneh Hajishirzi. 2022. Attentional Mixtures of Soft Prompt Tuning for Parameter-efficient Multi-task Knowledge Sharing. *ArXiv* (2022), arXiv–2205.11961.

[2] Avishek Joey Bose, Ankit Jain, Piero Molino, and William L Hamilton. 2019. Meta-graph: Few shot link prediction via meta learning. *arXiv preprint arXiv:1912.09867* (2019).

[3] Nikita Dvornik, Cordelia Schmid, and Julien Mairal. 2020. Selecting relevant features from a multi-domain representation for few-shot classification. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*. Springer, 769–786.

[4] Zhaoye Fei, Yu Tian, Yongkang Wu, Xinyu Zhang, Yutao Zhu, Zheng Liu, Jiawen Wu, Dejiang Kong, Ruofei Lai, Zhao Cao, et al. 2022. Coarse-to-Fine: Hierarchical Multi-task Learning for Natural Language Understanding. In *Proceedings of the 29th International Conference on Computational Linguistics*. 4952–4964.

[5] Hang Gao, Zheng Shou, Alireza Zareian, Hanwang Zhang, and Shih-Fu Chang. 2018. Low-shot Learning via Covariance-Preserving Adversarial Augmentation Networks. In *Neural Information Processing Systems*.

[6] Yun He, Steven Zheng, Yi Tay, Jai Gupta, Yu Du, Vamsi Aribandi, Zhe Zhao, YaGuang Li, Zhao Chen, Donald Metzler, et al. 2022. Hyperprompt: Prompt-based task-conditioning of transformers. In *International Conference on Machine Learning*. PMLR, 8678–8690.

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

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

[9] Ronghang Hu and Amanpreet Singh. 2021. Unit: Multimodal multitask learning with a unified transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1439–1449.

[10] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language?. In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

[11] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[12] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742* (2017).

[13] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[14] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In *Annual Meeting of the Association for Computational Linguistics*.

[15] Xiangyang Liu, Tianxiang Sun, Xuanjing Huang, and Xipeng Qiu. 2022. Late Prompt Tuning: A Late Prompt Could Be Better Than Many Prompts. *ArXiv* (2022), arXiv–2210.11292.

[16] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT Understands, Too. *arXiv e-prints* (2021), arXiv–2103.10385.

[17] 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* (2019), arXiv–1907.11692.

[18] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S Yu. 2017. Learning multiple tasks with multilinear relationship networks. *Advances in neural information processing systems* 30 (2017).

[19] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. 2021. A review on the attention mechanism of deep learning. *Neurocomputing* 452 (2021), 48–62.

[20] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language Models as Knowledge Bases?. 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)*. 2463–2473.

[21] 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*. 7930–7946.

[22] 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)*. 4222–4235.

[23] Yisheng Song, Ting Wang, Subrota K Mondal, and Jyoti Prakash Sahoo. 2022. A comprehensive survey of few-shot learning: Evolution, applications, challenges, and opportunities. *arXiv preprint arXiv:2205.06743* (2022).

[24] Joaquin Vanschoren. 2018. Meta-Learning: A Survey. *ArXiv* abs/1810.03548 (2018).

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[26] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.

[27] Jianing Wang, Chengyu Wang, Fuli Luo, Chuanqi Tan, Minghui Qiu, Fei Yang, Qiuhui Shi, Songfang Huang, and Ming Gao. 2022. Towards Unified Prompt Tuning for Few-shot Text Classification. *ArXiv* (2022), arXiv–2205.05313.

[28] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. 2022. S-Prompts Learning with Pre-trained Transformers: An Occam's Razor for Domain Incremental Learning. *ArXiv* (2022), arXiv–2207.12819.

[29] Yaqing Wang, Quanming Yao, James Tin-Yau Kwok, and Lionel Ming shuan Ni. 2019. Generalizing from a Few Examples: A Survey on Few-Shot Learning. *arXiv: Learning* (2019).

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

[31] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, VG Vinod Vydiswaran, and Hao Ma. 2022. IDPG: An Instance-Dependent Prompt Generation Method. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 5507–5521.

[32] Liqiang Xiao, Honglun Zhang, and Wenqing Chen. 2018. Gated multi-task network for text classification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 726–731.

[33] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. 2020. Graph few-shot learning via knowledge transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6656–6663.

[34] Hongbin Ye, Ningyu Zhang, Shumin Deng, Xiang Chen, Hui Chen, Feiyu Xiong, Xi Chen, and Huajun Chen. 2022. Ontology-enhanced Prompt-tuning for Few-shot Learning. In *Proceedings of the ACM Web Conference 2022*. 778–787.

[35] 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)*. 1–9.

[36] Yu Zhang and Qiang Yang. 2022. A Survey on Multi-Task Learning. *IEEE Trans. Knowl. Data Eng.* 34, 12 (2022), 5586–5609. https://doi.org/10.1109/TKDE.2021.3070203

[37] Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2023. Can chatgpt understand too? a comparative study on chatgpt and fine-tuned bert. *arXiv preprint arXiv:2302.10198* (2023).

[38] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13001–13008.

[39] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2019. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* 109 (2019), 43–76.