Making Parameter-efficient Tuning More Efficient: A Unified Framework for Classification Tasks

Xin Zhou[★*], **Ruotian Ma**[★*], **Yicheng Zou**[★], **Xuanting Chen**[★], **Tao Gui**[♦†], **Qi Zhang**[★♣†], **Xuanjing Huang**[★], **Rui Xie**[♠], **Wei Wu**[♠]
[★]School of Computer Science, Fudan University
[♦]Institute of Modern Languages and Linguistics, Fudan University
[♣]Shanghai Key Laboratory of Intelligent Information Processing
[♠]Meituan Inc., Beijing, China
{xzhou20,rtma19,tgui,qz,xjhuang}@fudan.edu.cn

Abstract

Large pre-trained language models (PLMs) have demonstrated superior performance in industrial applications. Recent studies have explored parameter-efficient PLM tuning, which only updates a small amount of task-specific parameters while achieving both high efficiency and comparable performance against standard fine-tuning. However, all these methods ignore the inefficiency problem caused by the task-specific output layers, which is inflexible for us to re-use PLMs and introduces non-negligible parameters. In this work, we focus on the text classification task and propose *plugin-tuning*, a framework that further improves the efficiency of existing parameter-efficient methods with a unified classifier. Specifically, we re-formulate both token and sentence classification tasks into a unified language modeling task, and map label spaces of different tasks into the same vocabulary space. In this way, we can directly re-use the language modeling heads of PLMs, avoiding introducing extra parameters for different tasks. We conduct experiments on six classification benchmarks. The experimental results show that plugin-tuning can achieve comparable performance against fine-tuned PLMs, while further saving around 50% parameters on top of other parameter-efficient methods.

1 Introduction

In many industrial applications about natural language processing, it becomes a de-facto paradigm that we first pre-train large-scale language models (PLM) (Devlin et al., 2019; Peters et al., 2018; Radford et al., 2019) on external corpora and then fine-tune them on target tasks. However, each fine-tuned PLM is generally applicable to only one task. As the number of applications increases, deploying independent instances of fine-tuned PLMs for different tasks significantly increases the computation and storage costs. For instance, GPT-3 (Brown et al., 2020) contains 175B parameters, which makes it almost impossible to fine-tune and deploy GPT-3 for target tasks. Even if large-scale fine-tuned PLMs are available, they are also not conducive to community distribution and sharing.

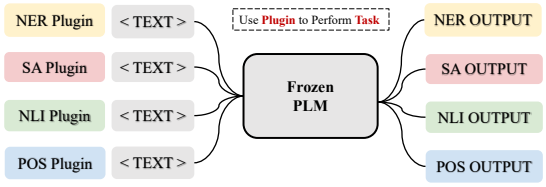


Figure 1: An intuitive overview of plugin-tuning. Plugin tuning only trains and stores a parameter-efficient plugin rather than a full PLM for each task. Different tasks share a unified output layer despite different label spaces.

Recently, a new branch of researches named parameter-efficient tuning (Guo et al., 2021; Zaken et al., 2021; Hu et al., 2022) have received much attention in the NLP community. Compared to standard fine-tuning, these methods fine-tune only a small portion of the model parameters while keeping most of the PLM parameters frozen (Ding et al., 2022). The minimal trainable parameters not only remarkably promote the deployment and storage efficiency when adapting PLMs, but also make it feasible to train large-scale PLMs such as GPT-3. Additionally, recent works validate that the parameter-efficient methods can achieve comparable performance with full-parameters fine-tuning in a wide range of NLP applications (Aghajanyan et al., 2020; Hu et al., 2022; Ding et al., 2022).

However, all the existing parameter-efficient methods ignore the inefficiency problem caused by the specific output layers for different tasks, which results in two problems. First, utilizing different task paradigms and retaining different output layers

[*] Equal Contribution.
[†] Corresponding authors.

is inflexible and hinders the community model distribution and sharing. Second, the parameters introduced by the task-specific output layers are non-negligible especially under the parameter-efficient scenarios. These two essential issues limit parameter-efficient tuning from being further efficient.

In this work, we propose plugin tuning, a framework to further improve all the parameter-efficient methods in classification tasks. We unify all the classification tasks into a same paradigm, thus different tasks can be performed with a unified classifier. Meanwhile, no trainable parameter is required for training the classifier. Specifically, we re-formulate different tasks into a same language modeling task, and directly reuse the language model heads of PLMs for classification. The label spaces of different tasks are mapped to task-specific label words, all belonging to a unified vocabulary space. To select the proper label words, we propose a principled algorithm that is applicable to both token and sentence classification tasks. In this way, the efficiency of all the parameter-efficient methods can be largely promoted. Additionally, the unified task paradigm provides a new way to perform classification tasks with generative PLMs such as GPT-3, which is unexplored in existing parameter-efficient methods. Our codes are publicly available at *Github*[1].

Our contributions can be summarized as follows:

- We propose a unified paradigm for all classification tasks to further improve the efficiency based on all parameter-efficient methods.
- We propose a principled way to select proper label words for both token and sentence classification tasks. Intensive experiments are conducted to analyze the proposed method.
- Plugin-tuning can achieve comparable performance with full-size fine-tuning on six tasks, as well as save up to 50% of the parameters required by the parameter-efficient tuning.

2 How to Boost Efficiency of Fine-tuning?

2.1 Standard Fine-tuning

We start with the introductions of fine-tuning PLM on token and sentence classification tasks, followed by the disadvantages of fine-tuning.

[1]https://github.com/xzhou20/Plugin-tuning

Given a sequence of tokens $\mathbf{X} = [x_1, ...x_n]$, the PLM usually encodes $\mathbf{X}$ with multi-layer bidirectional Transformer and outputs its representation of final hidden state:

$$\mathbf{H} = encoder_{\Phi}(\mathbf{X}), \tag{1}$$

where $\mathbf{H} \in \mathbb{R}^{n \times d_h}$, d_h is the dimension of the hidden state and Φ is the parameters of transformer layers. After encoding, a simple softmax classifier is added to the top of encoder to predict the target label based on the task.

Suppose we are given a sentence classification task S with label space $L_S \in \mathbb{R}^{ls}$. The aim of S is to predict a label $y \in L_S$ of sentence $\mathbf{X}$:

$$p(y|\mathbf{h}_{cls}) = Softmax(\mathbf{W}_S\mathbf{h}_{cls}), \tag{2}$$

while a token classification task T aims to predict a label $y_i \in L_T$ of each x_i in sentence $\mathbf{X}$:

$$p(y_i|\mathbf{h}_i) = Softmax(\mathbf{W}_T\mathbf{h}_i), \tag{3}$$

where $\mathbf{W}_S \in \mathbb{R}^{d_h \times l_S}$ and $\mathbf{W}_T \in \mathbb{R}^{d_h \times l_T}$.

During training phase, the standard fine-tuning directly updates all parameters to minimize the following objective:

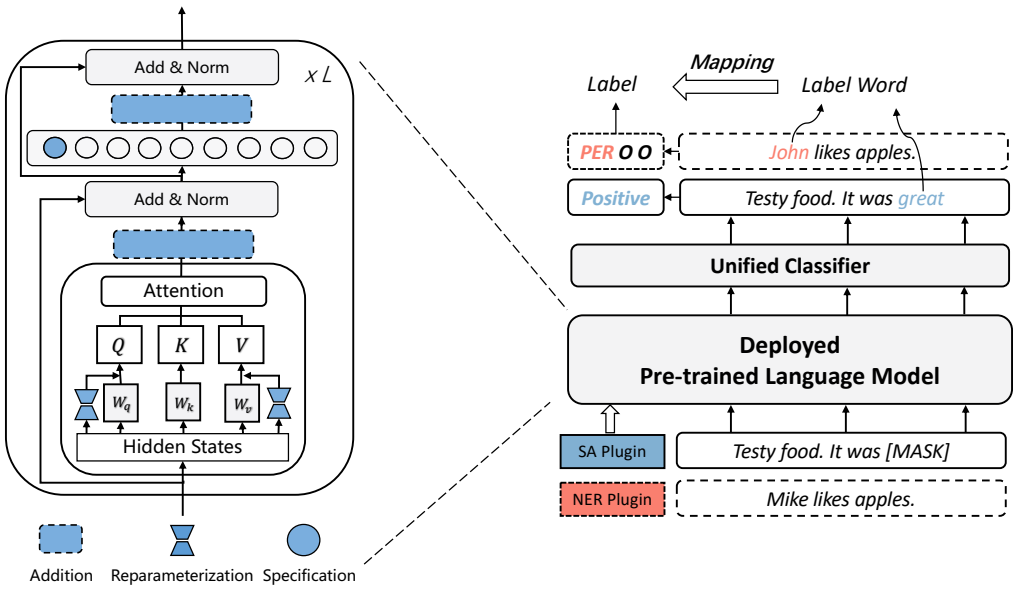
$$Sentence: \min_{\Phi_S, \mathbf{W}_S} -\log(P(y|\mathbf{X})) \qquad y \in L_S, \tag{4}$$

$$Token: \min_{\Phi_T, \mathbf{W}_T} -\sum_{i}^{n} \log(P(y_i|\mathbf{X})) \quad y_i \in L_T, \tag{5}$$

where Φ_S, $\mathbf{W}_S$, Φ_B and $\mathbf{W}_T$ are fine-tuned parameters for target tasks. A major drawback of fine-tuning is the need to update all parameters of the PLM. Updating large-scale Φ results in significant computation costs, and fine-tuned parameters Φ_S and Φ_B are also inefficient to store and deploy.

2.2 Not All Parameters are Critical

The question to be asked here is: do we really need to update all the parameters? Although the theoretical questions are not well explored, empirical evidence shows that: not all parameters are critical to be updated (Aghajanyan et al., 2020; Chen et al., 2020). Recent parameter-efficient tuning with distinct tuned parameter selection could achieve comparable performance to standard fine-tuning. In this way, we do not fine-tuning the large-scale Φ, but instead update selected $\Delta\Phi$



(a) Three types of parameter-efficient methods

(b) An example of plugin-tuning

Figure 2: An overview of plugin-tuning. (a) is the three types of parameter-efficient methods, which can be used as our plugin. (b) shows the flow of plugin-tuning, we input the text and task-specific plugin to the deployed PLM, the PLM influenced by plugin outputs the label words in the sentence. The actual labels are obtained by label word mapping. "John" and "great" are label words selected by querying language model, respectively representing label "PER" for task NER and label "Positive" for task SA.

where $|\Delta\Phi| \ll |\Phi|$. Ding et al. (2022) divides existing parameter-efficient methods into three groups: (1) addition-based methods (Houlsby et al., 2019) introduce extra trainable neural modules or parameters that do not exist in the original model or process. (2) specification-based methods (Zaken et al., 2021) only update certain parameters in the original model. (3) reparameterization-based methods (Hu et al., 2022) transform existing parameters to a smaller size. An example of above methods is shown in Figure 2 (a). In this paper we refer to these task-specific parameters $\Delta\Phi$ collectively as **plugin** for convenience, regardless which parameter-efficient method is used.

2.3 Output Layers: An Overlooked Problem

A critical problem overlooked in parameter-efficient methods is the efficiency problem of different output layers. Aghajanyan et al. (2020) shows that only a few thousand parameters are enough to handle various NLP tasks. However, no matter how efficient the method is, the task-specific output layers introduce non-negligible parameters, especially when encountering a large label space. (e.g., ten labels will introduce 10,240 parameters of classifier when using bert-large) . In addition to efficiency, parameter-efficient methods keep the model parameters consistent with pre-trained parameters, but we do not keep the optimization objective consistent with pre-training. The gap between downstream tasks and pre-training tasks may lead to sub-optimal performance (Liu et al., 2021a). This problem can not handle easily because of the different label spaces for different tasks. Therefore, the task-specific output layers hinder the further improvement of effectiveness and efficiency for all parameter-efficient methods, which cannot be overlooked.

3 Our Method

In this work, we propose plugin-tuning, a unified framework to improve parameter-efficient tuning. Our method is based on parameter-efficient tuning but avoids the problems caused by task-specific output layers. An overview of plugin-tuning is shown in Figure 2. The plugin here refer to the trainable parameters required for parameter-efficient method. We inject the plugin to the frozen PLM, and the plugin is applied to a specific position to control the PLM, as shown in Figure (a). The PLM will predict the label word in label-related position of sentence, and the real label can be obtained by label word mapping. Any parameter-efficient method can serve as our plugin. We only optimize the lightweight plugin during training

while keeping other parameters frozen. The details of the unified classifier are shown in the following subsections.

3.1 Unified Classifier

As shown in Section 2.3, the different label spaces make it hard to share a unified output layer. Inspired by prompt-base tuning (Liu et al., 2021b; Ma et al., 2022), we find that a label can be represented by a word. In this manner, different label spaces can be mapped to the same vocabulary space. Thus, we propose a unified classifier to solve this problem by re-using the language model head. To this end, we reformulate token and sentence classification tasks as a unified language modeling task, the PLM is trained to predict a label word as an indication of the real label. Furthermore, we propose an algorithm to search label words automatically by querying the language model.

Take sentiment classification as an example, we insert the "It was [MASK]." to the end of the sentence, and the PLM is expected to predict a word that indicates the sentiment at the position of [MASK]. As for token classification such as named entity recognition (NER), the PLM are trained to predict a a word that is more common than the original entity word, as shown in Figure 2 (b).

Formally, suppose we are given a classification task with label space L, a vocabulary V, a label word mapping function $M : L \rightarrow V$ and a input $\mathbf{X} = [x_1, ..., x_n]$. The classification task is reformulated to assign a word $y_i \in V$ to the special position:

$$p(y_i|\mathbf{X}) = Softmax(\mathbf{W}_{lm}\mathbf{h}_i), \tag{6}$$

where $\mathbf{h}$ is the final hidden state and i is the label-related position(position of [MASK] for sentence classification, all positions for token classification). In this way, different tasks can re-use the language model head $\mathbf{W}_{lm}$ as the unified classifier, diverse label space can be all adapted to the vocabulary V. Both token and sentence classification tasks have a unified language modeling objective:

$$Unified : \min_{\Delta\Phi} - \sum_{i \in Y_{idx}} \log(P(y_i|\mathbf{X})) \qquad y_i \in V, \tag{7}$$

where $\Delta\Phi$ is the trainable parameters required by parameter-efficient tuning and Y_{idx} are label-related positions, y_i is the label word, which is mapped according to its label. We make the word of label "O" predict itself in token classification. In prediction phase, we take the word with the highest probability in the PLM's prediction and map it to real labes by label word mapping function M.

Algorithm 1 Label Word Searching

Input: Training Set $D = \{X_i, Y_i\}_{i=1}^{N}$.
Parameter: Pre-trained language model LM.
Output: Label Word Map M.

```
Let label word map M = ∅;
for c ∈ L do
    Let freq_c = ∅;
end for
for (X = {x_i}_{i=1}^n, Y = {y_i}_{i=1}^n) ∈ D do
    Let Ŷ = LM(X);
    if i is related to a label c then
        Select the ten words with the highest probability from ŷ_i and update freq_c;
    end if
end for
for c ∈ L do
    The most frequent word w_freq ∈ freq_c is selected as label word of c, M[c] ← w_freq;
end for
return M
```

3.2 Label Word Search

To avoid the cumbersome label word engineering, we propose a method to search the label word automatically. To keep the consistency of pre-training tasks and downstream tasks, we leverage the original PLM for label word searching. Since the classification tasks are reformulated to language modeling task, a appropriate label word can be chosen by the PLM itself. We directly leverage the original pre-trained language model predict the candidate words at their label-related position and select the appropriate labeled words based on the frequency of candidate words under that label. The process is shown in Algorithm 1.

4 Experiment

We verify the effectiveness of the plugin-tuning method on a large number of tasks including both token and sentence classification tasks, and select representative parameter-efficient methods to demonstrate that plugin-tuning is compatible with arbitrary parameter-efficient methods. In addition, we also simulate real scenarios and counted the

deployment time to further verify the efficiency of our method in deployment. Finally, we provide a comprehensive analysis of the unified classifier, which is the main module of the plugin-tuning.

4.1 Dataset

To verify the effectiveness of the our method, we conduct experiments on six representative classification tasks. Next, we describe the datasets selected for token classification and sentence classification respectively. The statistics of the datasets are summarized in Table 1.

We evaluate the proposed method on three token classification tasks, including Named Entity Recognition (NER), Part of Speech tagging (POS) and text chunking (Chunking). For named entity recognition, we select the CoNLL 2003 (Sang and De Meulder, 2003), a newswire domain benchmark. For part of speech tagging, we select the Wall Street Journal (WSJ) data from the Penn TreeBank v3 (Marcus et al., 1993). For text chunking, we select the CoNLL 2000 (Tjong Kim Sang and Buchholz, 2000). BIO2 tagging scheme is used for NER and chunking.

As for sentence classification tasks, we select three common tasks, Sentiment analysis (SA), Natural Language Inference (NLI) and Question Classification (QC). We select the dataset based on the amount of data from GLUE (Wang et al., 2018). For SA, we chose the commonly used SST2. For NLI, we chose MNLI, a dataset from GLUE with a large amount of data to challenge our method. For QA, we choose TREC50 with only thousands of data to explore whether plugin-tuning can perform well in this scenario. The test sets for these three datasets are not publicly available, so we use the original validation set directly as the test set.

Task	Dataset	#Train	#Test	Labels
NER	CoNLL2003	204,567	46,666	9
CHK	CoNLL2000	211,727	47,377	23
POS	WSJ	912,344	129,654	46
SA	SST2	67,350	873	2
NLI	MNLI	392,702	9815	3
QC	TREC50	5452	500	47

Table 1: Statistics of the datasets. # means the number of sentences in SA, NLI and RC, and means the number of tokens in the NER, Chunking and POS.

4.2 Baseline

We select representative methods from the three types of delta tuning to verify whether plugin tuning is compatible with different delta tuning methods. Besides that, the standard fine-tuning is used to show the proper performance of PLMs.

Fine-Tuning (Liu et al., 2019) optimizes all parameters of the PLM for each task, which is the main approach for transferring the PLM to downstream tasks.

LoRA (Hu et al., 2022) is a reparameterization-based baseline. LoRA injects trainable low-rank matrices into transformer layers to approximate the weight updates, which can achieve good performance with extremely small parameters.

BitFit (Zaken et al., 2021) is a specification-based baseline. It only trains the bias term and classifier in the PLM.

Adapter (Houlsby et al., 2019) is a popular addition-based method of parameter-efficient tuning, which adds adapter layers to every transformer blocks. Only the adapter layers and the classifier are trainable during the training phase.

Plugin-tuning is our proposed method, which can be applied to any parameter-efficient methods to boost its efficiency and performance.

4.3 Implementation Details

In this work, we implement the parameter-efficient methods with roberta-base (Liu et al., 2019). Each parameter-efficient method follows the official code. AdamW optimizer and linear decaying schedule are used for all baselines. We search the learning rate from 1e-3 to 1e-5, epochs from {5, 10, 30} with a batch size of 16. We report the best results on the test set for each task. The model and hyperparameters are selected based on the validation set. The label word searching is also based on roberta-base. The selected label word is shown in the appendix.

4.4 Main Results

In this section, we verify the impact of using plugin-tuning on the performance and parameters of parameter-efficient tuning on six tasks. The results are shown in Table 2. A detailed analysis of the experimental results is presented below.

Parameter Efficiency Benefiting from the unified classifier, plugin-tuning can further reduce task-specific parameters required by parameter-efficient tuning on all tasks, and its effect is

Task/Method		LoRA Ori → Plugin-tuning	BitFit Ori → Plugin-tuning	Adapter Ori → Plugin-tuning
NER	F1	91.22 → 91.22 (↑0.00)	91.17 → 91.05 (↓ 0.12)	91.13 → 91.34((↑0.21)
	Param	44.3k → 36.9k (↓15%)	161k → 153k (↓4%)	127k → 120k (↓5%)
CHK	F1	95.58 → 96.34 (↑1.95)	93.25→94.48(↑1.23)	96.10 → 96.89 (↑0.79)
	Param	61.4k → 36.9k (↓29%)	172k → 153k (↓10%)	138k → 120k (↓13%)
POS	Acc.	96.84 → 97.42 (↑0.58)	97.03 → 97.57 (↑0.54)	97.51 → 97.54 (↑0.03)
	Param	72.2k → 36.9k (↓49%)	189k → 153k (↓18%)	156k → 120k (↓23%)
SA	Acc.	93.89 → 94.15 (↑0.16)	94.04 → 94.03 (↓0.01)	94.95 → 94.56 (↓0.39)
	Param	38.4k → 36.9k (↓4%)	156k → 153k (↓1%)	121k → 120k (↓1%)
NLI	Acc.	85.45 → 85.46 (↑0.01)	84.61 → 84.53 (↓0.08)	86.26 → 86.36 ((↑0.10)
	Param	39.2k → 36.9k (↓6%)	156k → 153k (↓1%)	122K → 119K (↓2%)
QC	Acc.	92.20 → 92.80 (↑0.60)	91.80 → 92.20 (↑0.40)	92.80 → 93.20 (↑0.40)
	Param	73.0k → 36.9k (↓49%)	190k → 153k (↓19%)	156k → 120k (↓23%)

Table 2: Main Results for six classification tasks. We report F1 score for Chunking and NER, accuracy for POS, SA, NLI and QC. Higher is better for these metrics. Param means the number of task-specific parameters; k stands for thousand. Lower is more efficient for the Param. We compare the performance and parameters on the Original baseline and the baseline with Plugin-tuning (Ori → Plugin-tuning). The change in performance and the percentage of reduced parameters are calculated. Items marked in red indicate that plugin-tuning boosts its performance or efficiency.

particularly significant on tasks with many labels. For tasks with more than 40 labels such as POS, plugin-tuning can reduce the number of parameters by up to 49%. For the theoretical worst case, on the sentiment classification task with only two labels and the NLI task with three labels, the advantage of our method is not significant, the percentage of parameter reduction is at least 1%. At current stage, plugin-tuning is more suitable for scenarios where the number of labels is large. As parameter-efficient tuning develops, the amount of task-specific parameters required will be further reduced, and the advantages of plugin-tuning will gradually become more pronounced.

Overall Performance In addition to the efficiency of the parameters, plugin-tuning also achieves performance improvements on most tasks, and this improvement is mainly seen on the token-level classification task. The most significant improvement is in the chunking task, with an average of 1 point improvement on the three datasets. We speculate that this is because the unified classifier reuses the parameters of the pre-trained model, and the gap between its training target and the pre-trained task is small, so it can benefit from the rich knowledge hidden in the pre-trained model. Token-level tasks with many categories and a high degree of detail benefit more significantly from them. For the sentence-level classification task, there are both gains and losses in performance after applying plugin-tuning, but the overall impact on performance is not significant. We found that neither performance gains nor performance drops were significant on tasks with large amounts of data. We conjecture that this is because neither the task-specific classifier nor the unified classifier is a dominant factor for performance when the data is large.

Comparison with Fine-tuning We selected the best performance among all plugin-tuning results to compare with standard fine-tuning. The results are shown in Table 3. Although the overall performance is inferior to fine-tuning, the difference between plugin-tuning and fine-tuning has been negligible. Since number of parameters of plugin-tuning is only 0.01% of fine-tuning, plugin-tuning can be used as an efficient and effective alternative to fine-tuning.

Model	POS	NER	CHK	SA	NLI	QC
Fine-tuning	97.69	91.45	97.03	94.72	87.60	93.20
Plugin-tuning	97.42	91.34	96.89	94.56	86.36	93.20

Table 3: Comparison of standard fine-tuning and plugin-tuning.

Analysis of Different Tasks and Models Our method is most useful for Adapter and LoRA, and is not stable on BitFit. However, the differences in

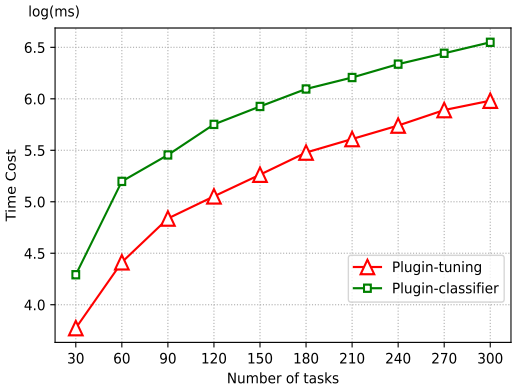


Figure 3: Time cost of redeployment. We take the log value to better show the results.

parameter-efficient methods are not as significant as the impact of the differences between tasks. From a task perspective, BitFit is unstable and performs poorly on larger datasets. Adapter and LoRA achieve similar and better performance on most tasks, but LoRA requires a smaller number of parameters and is therefore more efficient.

4.5 Efficiency Advantage

In this section, we study the difference in the efficiency of different parameter-efficient methods on deployment. We focus on redeployment, i.e., when a new task arrives, we want to release the resources of the old model and deploy a model that can execute the new task. Redeployment is suitable for scenarios where computational resources are tight. In online environments, the tasks from users arrive in the stream, and these tasks are diverse and hardly the same. The wide variety of tasks makes models need to be redeployed frequently, and in time-sensitive systems like search engines, the time of redeployment is part of the consideration.

To simulate the online setting, we construct the task streams by randomly sampling 30 to 300 samples from six tasks and show the time cost for switching tasks, the redeployment strategy is used at the arrival of each task.

To show the disadvantage of the task-specific classifier, we use plugin-tuning based on LoRA, which replaces the task-specific classifier with a unified one. For a fair comparison, we set the task-specific parameters of each baseline to 36864 (note that LoRA need additional parameters for the classifier).

The trend of time cost is shown in Figure 3. We take the log value to better show the results.

Although the parameters are the same scale, we find plugin-Classifier takes an additional 50% of time to reload the task-specific classifier (It will be longer on tasks with more labels), it demonstrates the efficiency of the unified classifier. These results can show the efficiency advantages of plugin-tuning over other parameter-efficient methods.

4.6 Analysis of Label Word Searching

Although the efficiency and performance of plugin-tuning have been verified by previous experiments, another part of plugin-tuning, the label word, still needs to be further analyzed. In this section, we show the necessity of label word searching and discuss the sample efficiency of our proposed algorithm 1.

4.6.1 Effectiveness Study

There are many ways to select a label word, it is still a question whether our method can select the right label words. In our view, a suitable label word needs to have at least no negative impact on the model and be easy to obtain. Therefore, in this section, we compare three different label word construction methods. (1) Each label word is a **handcraft**,designed by Human. the human-created label word sets are shown in Appendix. (2) Each label word is a **VirtualToken** in the vocabulary, the embedding of special token is randomly initialized. (3) Each label word is selected by our proposed **AutoSearch** algorithm.

We apply plugin-tuning to LoRA and test the performance of the three label word construction methods on different tasks based on the same parameter settings. The experimental results are shown in Table 4. When using VirtualToken, the performance drops significantly for tasks with many labels (CoNLL-NER), indicating that the random vector is hard for a model to fit. The results of Human shows that human intuition may be the same as the pre-trained model, which explains the good performance SST-2. However, on the token classification task, the human-created label word may not be consistent with the task goal, leading to bad results on CoNLL-NER. Our proposed AutoSearch can effectively leverage the knowledge of pre-trained models for downstream tasks, thus it outperforms other methods on all tasks.

4.6.2 Sample Efficiency

Our proposed label word search algorithm requires some labeled data to find the suitable label word. A concern here is how much data it needs to find such

Method	SST-2	CoNLL-NER
AutoSearch	**94.15**	**91.22**
VirtualToken	93.80	90.07
Handcraft	94.03	90.09

Table 4: Comparison of label word searching methods.

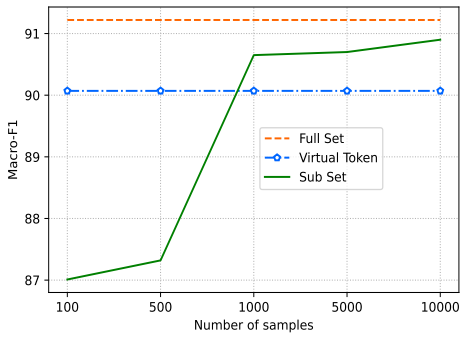


Figure 4: Performance variation of the NER task on the CoNLL03 when using different numbers of samples for searching label word.

label words. The sample efficiency is important since traversing the entire dataset can be time-consuming. Therefore, we select [100, 500, 1000, 5000, 10000] data uniformly from the train set, and search the label words in these subsets to verity the sample efficiency. The results in Figure 4 show that the 1000 samples are enough for achieving comparable performance. This indicates that our algorithm is data efficient.

5 Related Work

5.1 Pre-trained Language Model

Large-scale pre-trained language models, which pre-trained on a huge amount of data with self-supervised objectives, have greatly improved the performance of various downstream tasks (Qiu et al., 2020). Fine-tuning the pre-trained language model such as BERT (Devlin et al., 2019), roberta (Liu et al., 2019), and T5 (Liu et al., 2019) achieves state-of-art performance on various tasks. Increasing the number of PLM parameters is one of the intuitive ways to enhance its performance, thus leading to the creation of many giant models (Brown et al., 2020). But it is inconvenient to fine-tuning these models and apply fine-tuned models because of the large-scale parameters. They are also not conducive to community distribution and sharing. In this work, we explore the parameter-efficient tuning, which makes the deployment of PLMs feasible in many industrial scenarios.

5.2 Parameter-efficient Learning

Parameter-efficient learning, is proposed to solve the problem caused by PLMs' large-scale parameters. Unlike traditional methods such as distillation (Sanh et al., 2019), pruning (Michel et al., 2019) and quantization (Shen et al., 2020) that directly reduce the parameters of the model itself, parameter-efficient methods only fine-tune a small portion of the model parameters while keeping the rest untouched or learn external modules for new task. The rationale behind these methods can be related to the intrinsic dimension (Li et al., 2018; Aghajanyan et al., 2020), which states that PLM are often overparameterized and only need to learn a good solution in a small parameter space. Or prompt-tuning (Liu et al., 2021a), which originated from GPT-3, by entering special text to let the frozen PLM perform the target task. Recently, Ding et al. (2022) proposed to call these methods delta-tuning and analyzed it from a control theory perspective. Many attempts have been made to find the which part of parameters is efficient to learn, such as adapter (Houlsby et al., 2019), prefix-tuning(Li and Liang, 2021), and LoRA(Hu et al., 2022). In this work, we boost the efficiency of parameter-efficient tuning by avoiding the problem caused by task-specific classifier.

6 Conclusion

In this work, we propose plugin-tuning, a unified framework to improve parameter-efficient tuning for both token and sentence classification tasks. We use a unified classifier to handle different classification tasks by re-formulating them to a unified language modeling task, where no trainable parameter is required for training the classifier. To select the proper label words, we also propose a principled algorithm that is applicable to both token and sentence classification tasks. In this way, the efficiency of all the parameter-efficient methods can be largely promoted. The experiments show that our method achieves comparable performance against fine-tuned PLMs while further saving up to 49% parameters on top of other parameter-efficient methods. Future directions might be applying the unifying strategy into other tasks like text matching that require specific label spaces.

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded National Natural Science Foundation of China (No. 62076069, 61976056), Shanghai Municipal Science and Technology Major Project (No.2021SHZDZX0103). This research was supported by Meituan.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

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.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.

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.

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

Demi Guo, Alexander M. Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning.

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.

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

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes.

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.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021b. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

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.

Ruotian Ma, Xin Zhou, Tao Gui, Yiding Tan, Linyang Li, Qi Zhang, and Xuanjing Huang. 2022. Template-free prompt tuning for few-shot NER. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5721–5732, Seattle, United States. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Erik F Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8815–8821.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models.

A Label Word and Template

In this section, we show the template and label word in Table 5, Table 6 and Table 7. These tables are shown in the next page.

Dataset	Template	Label Word Set
SST-2	[X] It was $\langle mask \rangle$.	{positive: great, negative: terrible}
MNLI	[PREMISE] ? $\langle mask \rangle$ [HYPOTHESIS]	{entailment: Also, contradiction: But, nautral: Yeah}

Table 5: Templates and label words of plugin-tuning for sentence classification

Dataset	Label Word Set
CoNLL03	{B-ORG: United, B-MISC: American, B-PER: Paul, I-PER: Smith, B-LOC: France, I-ORG: Inc, I-MISC: Cup, I-LOC: York}
CoNLL2000	{B-NP: the, B-PP: of, I-NP: and, B-VP: is, I-VP: be, B-SBAR: that, B-ADJP: more, B-ADVP: also, I-ADVP: much, I-ADJP: lower, I-SBAR: if, I-PP: as, B-PRT: up, B-LST: 7, B-INTJ: yes, I-INTJ: was, B-CONJP: As, I-CONJP: well, I-PRT: or, B-UCP: wines, I-UCP:}
WSJ	{NNP: Mr, VBZ: is, JJ: first, NN: year, TO: to, VB: be, .: ., CD: million, DT: the, VBD: said, IN: in, PRP: he, NNS: people, VBP: have, MD: will, VBN: been, POS: "", JJR: more, ": ", RB: also, ,: „, FW: v, CC: and, WDT: which, (: (,):), :: -, PRP$: his, RBR: less, VBG: going, EX: There, WP: who, WRB: when, $: $, RP: up, NNPS: Yankees, SYM: /, RBS: most, UH: O, PDT: all, "": "", LS: 3, JJS: best, WP$: whose, NN|SYM: TV }

Table 6: Label words of plugin-tuning on token classification tasks

Dataset	Template	LabelWordSet
SST-2	[X] It was $\langle mask \rangle$.	{positive: good, negative: bad}
CoNLL03	-No Template-	{B-LOC:location, I-LOC:place, B-PER:person, I-PER:human, B-MISC: entity, I-MISC:other, B-ORG:organization, I-ORG:party},

Table 7: Human-created label words for ablation study