General Framework for Self-Supervised Model Priming for Parameter-Efficient Fine-tuning

Anonymous ACL submission

Abstract

Parameter-efficient methods (like Prompt or Adapters) for adapting pre-trained language models to downstream tasks have been popular recently. However, hindrances still prevent these methods from reaching their full potential. For example, two significant challenges are few-shot adaptation and cross-task generalization ability. To tackle these issues, we propose a general framework to enhance the few-shot adaptation and cross-domain generalization ability of parameter-efficient methods. In our framework, we **prime** the self-supervised model for parameter-efficient methods to rapidly adapt to various downstream few-shot tasks. To evaluate the authentic generalization ability of these parameter-efficient methods, we conduct experiments on a few-shot cross-domain benchmark containing 160 diverse NLP tasks. The experiment result reveals that priming by tuning PLM only with extra training tasks leads to the best performance. Also, we perform a comprehensive analysis of various parameter-efficient methods under few-shot cross-domain scenarios.

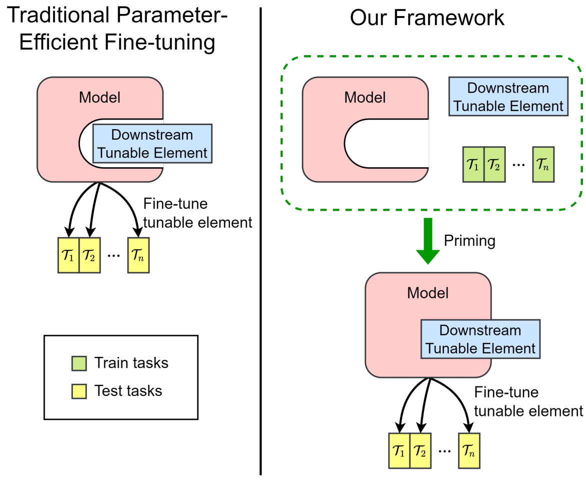


Figure 1: We propose a general framework to improve the performance of parameter-efficient fine-tuning. We prime the self-supervised model with training tasks for parameter-efficient methods.

1 Introduction

In recent years, pre-trained language models (PLMs) in natural language processing (NLP) are blooming everywhere (Devlin et al., 2018; Lewis et al., 2019; Raffel et al., 2019; Brown et al., 2020). However, not only the number of PLMs but also their size is rapidly growing, making it harder to perform full fine-tuning. To address the issue, tons of parameter-efficient fine-tuning methods have bubbled up, such as adapters (Houlsby et al., 2019; Pfeiffer et al., 2020; Zaken et al., 2021; Fu et al., 2022), or prompts (Lester et al., 2021; Li and Liang, 2021).

These innovative methods have made it equitable for researchers with insufficient resources. Also, Gu et al. (2021) demonstrated that prompt tuning is able to compete with fine-tuning when downstream data is sufficient, whereas it fails to compete equally under few-shot scenarios. Gu et al. (2021) pioneers the way of hybrid prompt pre-training, using both hard and soft prompts, which enables the prompts to match the performance of fine-tuning under few-shot settings, whereas other types of pre-training methods remain unexplored. Huang et al. (2022) proposed the method which applies meta-learning to pre-trained soft prompts under few-shot settings. However, they only apply pre-training in the Sentiment Analysis (SA) task, which lacks a comprehensive and general view from a higher level. On the other hand, Vu et al. (2022) indicates that pre-training prompts on source tasks can significantly boost the performance on target tasks.

Houlsby et al. (2019) empirically shows that adapters can achieve comparable performance by fine-tuning the entire model. However, Wang et al. (2022a) showed that there is still a significant performance gap compared to fully fine-tuning when only a handful of data is available. There are also several studies on improving the few-shot performance of adapters. Wang et al. (2022a) uses self-training to leverage large amounts of unlabeled data and successfully boosts the performance on six NLU tasks. Wang et al. (2022b) takes inspiration from the mixture-of-experts models and proposes a new mechanism of stochastic routing to a mixture of adapters. Previous research significantly improves few-shot performance in specific domains, but the ability to generalize to cross-domain remains unexplored.

Since existing self-supervised models are not tailored for cross-domain parameter-efficient fine-tuning, we propose a general framework to tackle the issue. The concept is shown in Fig. 1. We prime the self-supervised model with extra few-shot training tasks for parameter-efficient methods to rapidly adapt to various downstream few-shot tasks. After priming with extra few-shot training tasks, we can bridge the gap between the PLM and parameter-efficient methods like adapter and prompt, enabling them to fit the downstream tasks better.

On top of that, we conduct comprehensive experiments over adapters and prompt tuning, the two well-known parameter-efficient training methods. Our experiments include combinations of multi-task learning and meta-learning on adapters and soft prompts. Specifically, we choose (Ye et al., 2021), an NLP few-shot gym aiming at building few-shot learners who can generalize across diverse NLP tasks. In addition, we analyze the experiment results from different aspects and provide inclusive insight into these parameter-efficient training methods. The experiment result reveals that priming by tuning only PLM with extra training tasks leads to the best performance.

2 Related Work

2.1 Adapter

Adapters are lightweight modules introduced for the transformer architecture. It was first proposed by Houlsby et al. (2019) and soon became popular in NLP with several variants. Instead of fine-tuning the entire model, Adapters add extra trainable parameters and freeze the original PLM. In this work, we mainly adopt AdapterBias (Fu et al., 2022), which obtains comparable performance against Houlsby et al. (2019) while adding much fewer parameters to the model.

2.2 Prompt

Prompt-based tuning is an innovative method to use the power of PLMs efficiently. Li and Liang (2021) proposed prepending prefix vectors to the input of the transformer, reducing the computation consumption to a new level, and realizing the parameter efficiency. Han et al. (2021) proposed prompt tuning with rules (PTR). PTR encoded prior human knowledge into prompt tuning by composing sub-prompts into task-specific prompts, reducing the difficulty in designing the template. Pre-trained prompt(PPT) for prompt initialization is proposed by Gu et al. (2021). It shows that without tuning the PLM, it can perform well in downstream tasks when applying pre-trained prompts as downstream initialization. In addition, Gu et al. (2021) further explore their work on large-scale PLM with 11B parameters on few-shot learning. Huang et al. (2022) proposed Meta-learned Prompt Tuning (MetaPT) to further improve PTT (Gu et al., 2021)'s initialization by considering latent structure within the pre-trained data.

2.3 Adapter mix Prompt

The concept of mixing adapters and prompts was proposed by He et al. (2021). They propose *Mix-And-Match adapter* (MAM Adapter), which fuses the scaled parallel adapter with prefix prompt proposed by Li and Liang (2021). In our framework, we also include the concept of mixing adapters and prompts.

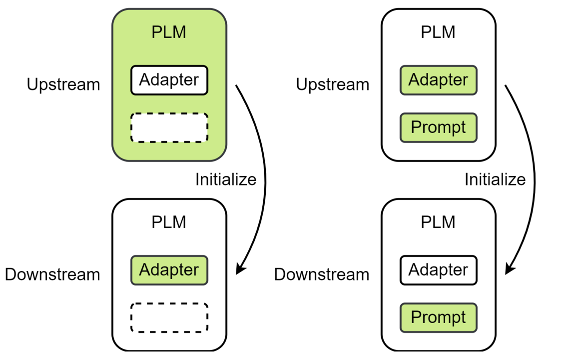
2.4 Meta Learning

Meta-Learning is well-recognized and a systematic pre-training method that enables models to rapidly adapt to different tasks with a small amount of data. Among several Meta Learning algorithms, Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017) has shown its success in many NLP tasks under few-shot settings, which is quite a suitable algorithm to empower our parameter-efficient methods to reach their full potential.

3 Methodology

3.1 Framework

Our work aims to comprehensively discover and analyze the performance of parameter-efficient methods under few-shot scenarios. We propose a general framework to prime the whole model (may include PLMs or other tunable elements) to adapt to various domains under few-shot scenarios. We divide the training pipeline into two parts: **Upstream Learning Stage** and **Downstream Fine-Tuning Stage**. In our work, we adopt MAML(Finn et al., 2017) and Multi-task learning(Caruana, 1997) as the **Learning method** in the upstream learning stage to train our model, which will be discussed in the following sections. In our framework, we tune different parameters in different stages. Specifically, the parameters tuned in the upstream learning stage are called **Upstream tunable elements**, while those tuned in the downstream fine-tuning stage are called **Downstream tunable elements**.



(a) Upstream: PLM Downstream: adapter (b) Upstream: adapter+prompt Downstream: prompt

Figure 2: Different combinations of tunable elements. The elements with dotted lines are unused. The green parts refer to the tunable elements, and the parameters in Downstream are initialized with the Upstream tunable elements.

3.2 Upstream Learning Stage

In **Upstream Learning Stage**, we aim at training the model to a point where downstream tunable elements can swiftly adapt to downstream tasks. Tunable elements include **PLM**, **adapter** and **prompt** in upstream learning stage. Among these elements, in addition to simple combinations like **PLM + adapter** or **prompt**, we also test some unexplored combinations like **adapter + prompt** and enumerate every possible combination within our settings.

Take Fig.2 as example. In the upstream learning stage, we can choose from either tuning only one element like Fig.2a or tuning multiple elements like Fig.2b. However, only one of the adapters and prompts can be tuned in the downstream fine-tuning stage.

3.2.1 Meta Learning

We adopt MAML (Finn et al., 2017) as our learning method. Following the algorithm in Finn et al. (2017), the parameters in the outer and inner loop are trained separately. Instead of tuning the whole model directly, we choose to update the **Upstream tunable elements** and **Downstream tunable elements**, respectively. As shown in Fig. 3 and Alg. 1, we first copy the current model parameters ψ to be the model initialization of the inner loop. Second, we tune the downstream tunable element ψ_d in the inner loop. Lastly, we compute the loss from the tuned model ψ_i' and training tasks $\mathcal{T}_i$ to update ψ_u. The updated ψ_u will be part of the model initialization of the next inner loop.

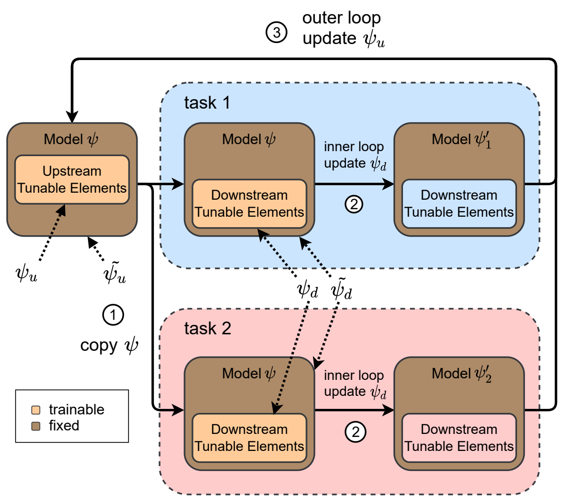


Figure 3: Training details of Parameter-Efficient MAML: (1) Copy ψ to be the initialization of the inner loop. (2) Split ψ into ψ_d (downstream tunable elements) and $\tilde{\psi}_d$. Fine-tune ψ_d for every task in $\mathcal{T}$. 3. Split ψ into ψ_u (upstream tunable elements) and $\tilde{\psi}_u$. Update ψ_u in the outer loop.

Algorithm 1 Parameter-Efficient MAML

$\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$: A set of training tasks
α, β: Outer lr, Inner lr
θ: PLM parameters
$\{\phi_1, \phi_2, \ldots\}$: Tunable elements
$\psi = [\theta; \phi_1; \phi_2; \ldots]$: All parameters of the model

Randomly initialize $\{\phi_1, \phi_2, \ldots\}$

while not done **do**
  for $\mathcal{T}_i \in \mathcal{T}$ **do**
    Split ψ into two parts, ψ_d and $\tilde{\psi}_d$ // ψ_d is tunable in inner loop
    Evaluate $\nabla_{\psi_d} \mathcal{L}_{\mathcal{T}_i}(f_\psi)$ with respect to K samples
    Compute adapted parameters with gradient
    descent: $\psi_{d,i}' = \psi_d - \beta \nabla_{\psi_d} \mathcal{L}_{\mathcal{T}_i}(f_\psi)$
    $\psi_i' = [\psi_{d,i}'; \tilde{\psi}_d]$
  end for
  Split ψ into two parts, ψ_u and $\tilde{\psi}_u$ // ψ_u is tunable in outer loop
  $\psi_u' = \psi_u - \alpha \nabla_{\psi_u} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\psi_i'})$
  $\psi \leftarrow [\psi_u'; \tilde{\psi}_u]$
end while
return ψ

3.2.2 Multi-task Learning

Multi-task Learning (Caruana, 1997) aims to learn multiple different tasks simultaneously while maximizing performance on all of them. The model may be able to learn cross-tasks knowledge beneficial to generalization. In our framework, we tune the upstream tunable elements on training tasks in the upstream learning stage and evaluate the few-shot ability of the model on testing tasks. For the upstream and downstream tunable elements, we can take Fig.2a for example. In the upstream learning stage, we tune the PLM among different training tasks, while we tune adapters on testing tasks respectively in the downstream stage.

3.3 Downstream Fine-Tuning Stage

Since the backbone of our work is to explore the few-shot ability of parameter-efficient methods, **only prompt and adapter are tunable in downstream stage**. In **Downstream Fine-Tuning Stage**, we aim at swiftly adapting the upstream parameters to downstream tasks. In this way, we can evaluate the ability of parameter-efficient methods under few-shot scenarios.

3.4 Specific Methods

The combinations in our experiments include some existing methods, like Meta-Adapters (Bansal et al., 2022). Moreover, we also propose two new approaches, Meta-Prompt and Adapter-mix-Prompt, to further explore the potential of priming the model. In fact, the aforementioned methods can all be considered the specific cases of our unified framework.

3.4.1 Training the Initialization

In Sec. 3.2 we mention that both adapters and prompt are available options of the upstream tunable elements. If we freeze the parameters of PLM and only train the adapters/prompts in both the upstream and downstream stages, we are actually training the initialization of adapters/prompts. Huang et al. (2022); Hou et al. (2022) apply meta-learning to train a better initialization of soft prompts for downstream tasks, which can be regarded as one of the combinations in our framework.

3.4.2 Meta-Adapters

Instead of fine-tuning the whole model, Bansal et al. (2022) proposed *Meta-Adapters* to reduce the number of tunable parameters. They insert meta-adapters in addition to regular adapters in the transformer blocks and keep the PLM frozen to reduce trainable parameters. Since meta-adapters are just extra adapters with different placements, we can view *Meta-Adapters* as a special case of our framework, where two kinds of adapters (meta-adapters and regular adapters) are trained in the upstream learning stage and only regular adapters are tunable in the downstream fine-tuning stage. We also consider the case that only meta-adapters are tunable in the upstream stage.

3.4.3 Meta-Prompt

Inspired by Bansal et al. (2022), we propose *Meta-Prompt*, a newly designed method aiming at improving the performance of regular prompt tuning. Fig. 4 illustrates how Meta-Prompt works. We concatenate the original input text (yellow blocks) with the regular prompt (green blocks) and another meta prompt (pink blocks). In the upstream learning stage, we can choose to tune both prompts (meta-prompt and regular prompt) or tune the meta-prompt only, while the meta-prompt remains fixed in the downstream fine-tuning stage. Similar to Meta-Adapters, Meta-Prompt is also considered one of the combinations in our framework.

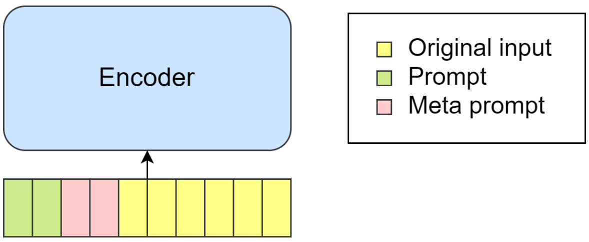


Figure 4: Meta prompt

3.4.4 Adapter mix Prompt

In our framework, adapters and prompts are considered two independent tunable elements. The soft prompt tokens (Lester et al., 2021) are prepended to the original input, and the adapters (Fu et al., 2022) are inserted in the transformer blocks (Vaswani et al., 2017). Therefore, it is viable for us to combine them in a single model. Under this setting, we can either tune adapters or prompt in the downstream stage. Although the concept seems to be similar to He et al. (2021), the implementation varies widely in practice. He et al. (2021) directly fine-tunes the adapters and prompt, while different components are trained in the upstream and downstream stages respectively.

4 Experiment

4.1 DataSet

We choose **CrossFit Challenge** (Ye et al., 2021) as our benchmark, which provides 160 different few-shot tasks with unified text-to-text format gathered from existing open-access datasets. For the tasks split, which implies the components of Train, Dev, and Test tasks, we select **random split** in Ye et al. (2021) to be the task split setting in our work. These tasks come from various domains, including Classification, Question Answering, Conditional Generation, and others. More explicit explanations of tasks can be found in Ye et al. (2021). Each few-shot classification or regression task contains 16 examples per class, and other types of tasks contain 32 examples. Briefly speaking, **CrossFit Challenge** is able to evaluate the authentic few-shot generalization ability of models.

In our experiment, we find that performance directly fine-tuning BART(Lewis et al., 2019) is awful in "freebase_qa," whose performance is nearly 0, leading to a lousy evaluation when we calculate relative gain since it will be huge. Because the lousy evaluation strongly influences our following assessment on all tasks, we decide to eliminate the results of freebase_qa when we calculate the model's average performance. However, to maintain the completeness of our experiment, we put all the original data in Table 2 and Table 3 in Appendix.

4.2 Setup

4.2.1 Tunable elements

To maintain the integrity of our experiment, we list all possible combinations, as shown in Table 1 in Appendix, and conduct experiments each by each to explore the impacts of different components. The experiment setup mainly follows Ye et al. (2021). Specifically, the tunable parameters in the upstream stage and downstream stage are different. In the upstream stage, we can tune prompt, adapter, meta-adapter, meta-prompt, and PLM, but we can only tune prompt or adapters in the downstream stage. Figure 2 shows two examples of different combinations.

Adapter

In this work, we mainly adopt AdapterBias (Fu et al., 2022) as our adapter module. AdapterBias adds a token-dependent shift to the hidden output of transformer layers, parameterized by only a vector and a linear layer. Compared with the original adapter design (Houlsby et al., 2019), the trainable parameters are further reduced while obtaining comparable performance.

Prompt

Prompt is one of our tunable elements. In our settings, we applied prompt tuning proposed by Lester et al. (2021), which concatenates tunable tokens before the input sentence and ask the PLM to generate corresponding output text. Following Lester et al. (2021), we set the prompt length to 100 tokens.

Meta-Adapter

Bansal et al. (2022) inserts meta-adapters before and after the regular adapters to make the pre-trained model a better few-shot learner. Meta-adapters have the same architecture as the regular adapters (Houlsby et al., 2019), but only the regular adapters are fine-tuned in the downstream stage.

4.2.2 Hyperparameters

Our hyperparameters settings follow Ye et al. (2021); Lester et al. (2021). The PLM we adopt is BART-base (Lewis et al., 2019) from Huggingface (Wolf et al., 2019). The prompt length in our main experiment is 100. In our implementation of Meta Learning, the optimizer in the outer loop is AdamW with 0.01 weight decay excluding bias and Layernorm terms, while the optimizer in the inner loop is SGD. We set the outer model, prompt, and adapter learning rate to be $8e^{-5}, 8e^{-3}$ and $1e^{-5}$ respectively; the inner learning rate is 0.025 and 0.001 for prompt and adapter, respectively. The epoch is set to 80, the train batch size is 1, and the inner batch size is 4 or 8 depending on GPU memory consumption. On the other hand, in our implementation of Multi-task learning, the optimizer is AdamW with 0.01 weight decay excluding bias and Layernorm terms. The learning rate for PLM, prompt, and adapter is $3e^{-5}$. The epoch is set to 10, and the train batch size is 32.

4.3 Metrics

For the evaluation metric, we also follow Ye et al. (2021), adopting *Average Relative Gain* (**ARG**) as the index of performance on each task. To depict our model's capability of generalization more precisely, we take *Relative Gain Standard Deviation* (**RGSTD**) into account, which is the standard deviation of relative gains among different tasks. Comparably, **RGSTD** can better represent the cross-task generalization ability. In a nutshell, **ARG** and **RGSTD** are both considered while evaluating the authentic few-shot ability of different baselines.

4.4 Main Result

The complete result can be found at Table 2 and Table 3 in Appendix. To better visualize the experiment result, we provide a scatter graph version to help understanding at Fig 5. From the scatter graph, we can simply compare the few-shot ability between different combinations.

Generally speaking, the combinations located at the bottom right corner are those with extraordinary few-shot ability since they have higher **ARG** and lower **RGSTD** simultaneously. On the other hand, the combinations located at the upper left corner show less generalization ability and robustness under few-shot scenarios.

Next, we use abbreviations to substitute the complete name of each combination for simplicity. For learning methods, we use **Meta** to represent Meta learning and **Multi** to represent Multi-task learning. For tunable elements, we use **M** to represent PLM, **A** to represent adapter, **P** to represent prompt, **MA** to represent meta-adapter and **MP** to represent meta-prompt.

To be more concise, we differentiate learning methods and downstream tunable elements by marker types. All combinations using MAML as their learning methods are squares, whereas those using Multi-task learning are circles. Also, all combinations with prompt as their downstream tunable elements are hollow, whereas those with adapters are solid. The parameter-efficient FT baselines of directly fine-tuning are in the shape of a star.

Lastly, we can simply summarize the result here and leave the detailed analysis to the next sections. Most points are on the right of the parameter-efficient FT baseline (blue stars) while some points even surpass the BART-FT baseline (ARG=0.0). Thus, it is obvious that our framework significantly enhances the performance of traditional parameter-efficient fine-tuning.

5 Analysis

5.1 Learning methods

In our work, MAML and Multi-task learning are two available learning methods. In Fig. 5, the colors stand for different combinations of upstream

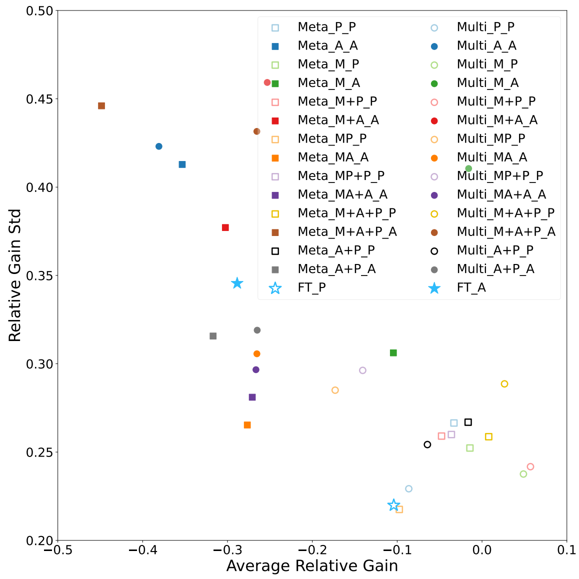


Figure 5: Experiment Result: The picture illustrates the performance of each combination. Generally speaking, points located at the bottom right side perform the best. All combinations using MAML as their learning methods are squares, while those using Multi-task learning are circles. All combinations with prompt as their downstream tunable elements are hollow, while those with adapters are solid.

and downstream tunable elements. If we observe the points having the same color, we can find that circles mostly locate on the right side of squares, which implies that Multi-task learning surpasses MAML in most cases. However, if we analyze the results from other aspects, these two learning methods exhibit opposite trends. In the case prompt serving as the downstream element, from Fig. 6(a)(c), we can easily tell MAML produces a more stable result among different upstream tunable elements. On the contrary, in the case adapter serving as the downstream element, two learning methods produce similar results. In a nutshell, when it comes to stability, MAML takes the lead by a small margin.

5.2 Upstream tunable elements

To better formulate the impact of different factors, we divide all combinations into four groups by their learning method and downstream tunable elements. In each group, the learning method and downstream tunable elements are set to be consistent. After taking a careful look at Fig. 6 and Fig. 5, we can reach the following conclusions:

5.2.1 What benefit parameter-efficient tuning the most

To compare the combinations of tuning only PLM with traditional parameter-efficient fine-tuning, we can take a look at *gray points* (FT_P & FT_A) and *orange points* (Meta_M_P & Meta_M_A & Multi_M_P & Multi_M_A) in Fig. 6, and in every group, *orange points* take a great lead to *gray points*. Also, we can tell that the *orange points* perform well in different groups, for almost all *orange points* are comparable to BART-FT baseline (ARG=0.0). **Tuning only PLM** in the upstream stage does help the tunable elements to reach the best performance in downstream tasks. The improvement is evident regardless of learning methods and downstream tunable elements, which is beyond our expectations, since tuning different elements at different stages is not always the most common approach. From our perspective, tuning PLM in the upstream stage manages to alleviate the issue that PLM is not optimized for parameter-efficient methods. However, unlike tuning only PLM in the upstream stage, tuning PLM with other tunable elements simultaneously fails to maintain exceptional in the adapter's case (Fig. 6 (b)(d)). The actual reason may need further research to fully unveil.

For the implementation detail, it is worth mentioning that for the case tuning only PLM in the upstream stage, parameter-efficient elements(adapters or prompts) are initialized but remain frozen until entering the downstream stage. In other words, the PLM is actually fitting a random initialized parameter-efficient element in the Multi-task learning case and fitting a few-shot tuned parameter-efficient element in MAML case.

5.2.2 Tuning prompt in the upstream stage helps

To compare the combinations of directly fine-tuning and tuning parameter-efficient elements in both upstream and downstream stages, we can inspect the dynamic between *blue points* (Meta_P_P & Meta_A_A & Multi_P_P & Multi_A_A) and *gray points* (FT_P & FT_A) in Fig 6. The conclusion is that tuning prompts in the upstream stage does help prompts to fit the downstream tasks better while in adapters case the impact is reversed. It seems that prompts can transfer across different tasks better than adapters.

5.2.3 Tuning elements with their meta counterparts doesn't fit better

To evaluate whether tuning adapter or prompt with their meta counterparts in the upstream learning stage help fitting downstream tasks, we can put attention to *red points* (Meta_MP_P & Meta_MA_A & Multi_MP_P & Multi_MA_A) and *purple points* (Meta_MP+P_P & Meta_MA+A_A & Multi_MP+P_P & Multi_MA+A_A) in Fig 6. We can see almost all *red points* located near *purple points*, which means tuning adapters or prompts with their meta counterpart doesn't bring much improvement.

5.2.4 Meta-adapters do help adapter to fit better

To evaluate whether meta-adapter or meta-prompt help adapter or prompt to fit downstream tasks better, We can take a look at the dynamic between *red* (Meta_MP_P & Meta_MA_A & Multi_MP_P & Multi_MA_A) & *purple points* (Meta_MP+P_P & Meta_MA+A_A & Multi_MP+P_P & Multi_MA+A_A) and *blue points*(Meta_P_P & Meta_A_A & Multi_P_P & Multi_A_A). We can easily get to the conclusion that Meta-adapters do help adapters (Fig. 6 (b)(d)) to fit the downstream tasks better while meta-prompt fails to bring the same level improvement (Fig. 6 (a)(c)). The design of meta-prompt can be further explored by future research.

5.2.5 Tuning adapter & prompt at the same time

To evaluate the performance of tuning adapter & prompt at the same time in the upstream learning stage, we can focus on *brown points* (Meta_M+A+P_P & Meta_M+A+P_A & Multi_M+A+P_P & Multi_M+A+P_A) and *pink points* (Meta_A+P_P & Meta_A+P_A & Multi_A+P_P & Multi_A+P_A) in Fig. 6. When comparing these points with *gray points* (FT_P & FT_A), we can tell the improvement is significant when the downstream tunable element is prompt (Fig. 6 (a)(c)). Nevertheless, it doesn't help adapters (Fig. 6 (b)(d)) to fit the downstream tasks better as prompts. The latent mechanism can be further studied by future research.

5.3 Downstream tunable elements

In our setting, prompts and adapters are two available downstream tunable elements. From Fig 5, we can tell prompt beats adapters generally since

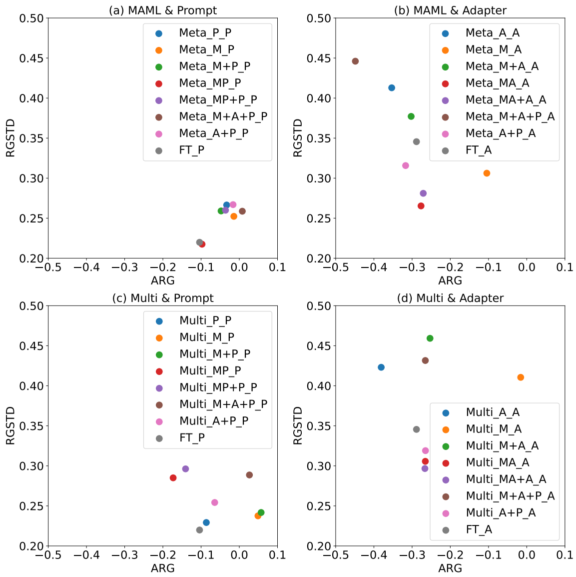


Figure 6: Upstream tunable elements result: In this figure, we divide all combinations into four groups by their learning methods and downstream tunable elements. In each group, the learning methods and downstream tunable elements are set to be consistent.

almost all *hollow points* locate relatively closer to the bottom right side (Hollow points are prompt and solid points are adapters). However, it's more rigorous to eliminate the impact of learning methods and upstream tunable elements. Therefore, we only compare those with the same learning method and upstream tunable elements. To be more specific, we ignore those with their meta counterpart in their upstream tunable elements (MP, MA, MP+P, MA+A), which are the *red & purple points* in Fig 6.

If we do a cross-comparison between the (a)(c) and (b)(d) in Fig 6, we can tell that prompt takes the lead by a great margin, even in the fine-tuning case. The conclusion implies that prompts can fit downstream tasks better than adapters in terms of generality and stability.

5.4 Tasks

In Fig.7, the bar of FT is corresponding to the blue solid and hollow stars in Fig. 5, respectively. In this section, we regard it as the baseline for prompts and adapters, respectively. However, in Fig. 5 and Fig. 6, the evaluation metrics are ARG, which calculates the average relative gain of each task, lacking information on individual tasks. To prevent the result severely influenced by a single task, we construct Fig.7 to visualize the relative improvement of each task.

The result not only eliminates our potential worry but also backs up the validity of our best results. From Fig. 7, we can see in most tasks, our best results – "Only tune PLM" in the upstream learning stage significantly improve the performance of traditional parameter-efficient fine-tuning regardless of the learning method and parameter-efficient methods.

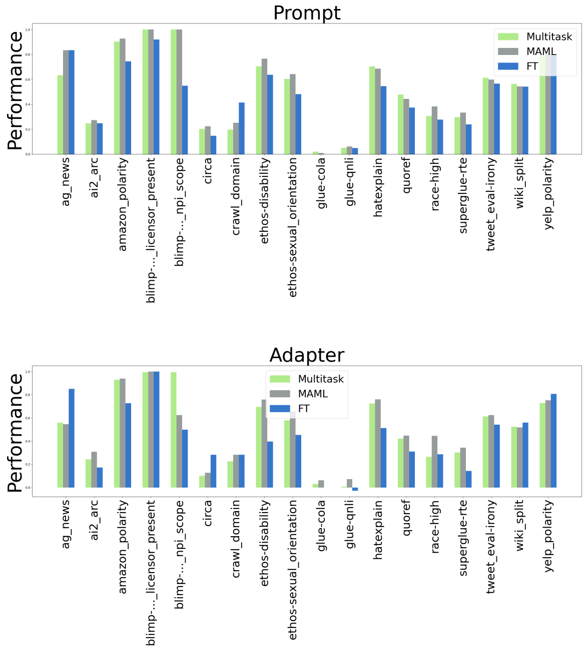


Figure 7: The performance of "Only Tune PLM" in prompt over all tasks. The horizontal axis is the name of the few-shot datasets, and the vertical axis is the performance.

6 Conclusion

In this paper, we propose a general framework to prime the self-supervised model for parameter-efficient methods to rapidly adapt to various downstream few-shot tasks. Among several combinations of learning methods and tunable elements, our experiment result shows that tuning only PLM in the upstream stage does enhance the performance of parameter-efficient methods adapting to few-shot downstream tasks by a great margin. Apart from this, the experiment reveals that prompts generally fit various downstream few-shot tasks better than adapters. Lastly, we find out that applying MAML as the learning method produces a more stable result while Multi-task Learning produces extreme value more easily.

References

Trapit Bansal, Salaheddin Alzubi, Tong Wang, Jay-Yoon Lee, and Andrew McCallum. 2022. Meta-adapters: Parameter efficient few-shot fine-tuning through meta-learning. In *First Conference on Automated Machine Learning (Main Track)*.

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.

Rich Caruana. 1997. Multitask learning: A knowledge-based source of inductive bias.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

Chin-Lun Fu, Zih-Ching Chen, Yun-Ru Lee, and Hung-yi Lee. 2022. Adapterbias: Parameter-efficient token-dependent representation shift for adapters in nlp tasks. *arXiv preprint arXiv:2205.00305*.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. Ppt: Pre-trained prompt tuning for few-shot learning.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. Ptr: Prompt tuning with rules for text classification.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning.

Yutai Hou, Hongyuan Dong, Xinghao Wang, Bohan Li, and Wanxiang Che. 2022. MetaPrompting: Learning to learn better prompts. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3251–3262, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

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

Yukun Huang, Kun Qian, and Zhou Yu. 2022. Learning a better initialization for soft prompts via meta-learning.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer.

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.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou', and Daniel Cer. 2022. SPoT: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059, Dublin, Ireland. Association for Computational Linguistics.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Awadallah, and Jianfeng Gao. 2022a. List: Lite prompted self-training makes parameter-efficient few-shot learners. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2262–2281.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022b. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. 2021. CrossFit: A few-shot learning challenge for cross-task generalization in NLP. pages 7163–7189.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

7 Appendix

7.1 Combination

Learning Methods	Upstream Tunable Elements	Downstream Tunable Elements	Abbreviation
MAML	prompt	prompt	Meta_P_P
	adapter	adapter	Meta_A_A
	model	prompt	Meta_M_P
	model	adapter	Meta_M_A
	model+prompt	prompt	Meta_M+P_P
	model+adapter	adapter	Meta_M+A_A
	meta-prompt	prompt	Meta_MP_P
	meta-adapter	adapter	Meta_MA_A
	meta-prompt+prompt	prompt	Meta_MP+P_P
	meta-adapter+adapter	adapter	Meta_MA+A_A
	model+adapter+prompt	prompt	Meta_M+A+P_P
	model+adapter+prompt	adapter	Meta_M+A+P_A
	adapter+prompt	prompt	Meta_A+P_P
	adapter+prompt	adapter	Meta_A+P_A
Multitask	prompt	prompt	Multi_P_P
	adapter	adapter	Multi_A_A
	model	prompt	Multi_M_P
	model	adapter	Multi_M_A
	model+prompt	prompt	Multi_M+P_P
	model+adapter	adapter	Multi_M+A_A
	meta-prompt	prompt	Multi_MP_P
	meta-adapter	adapter	Multi_MA_A
	meta-prompt+prompt	prompt	Multi_MP+P_P
	meta-adapter+adapter	adapter	Multi_MA+A_A
	model+adapter+prompt	prompt	Multi_M+A+P_P
	model+adapter+prompt	adapter	Multi_M+A+P_A
	adapter+prompt	prompt	Multi_A+P+P
	adapter+prompt	adapter	Multi_A+P_A

Table 1: Experiment Combinations

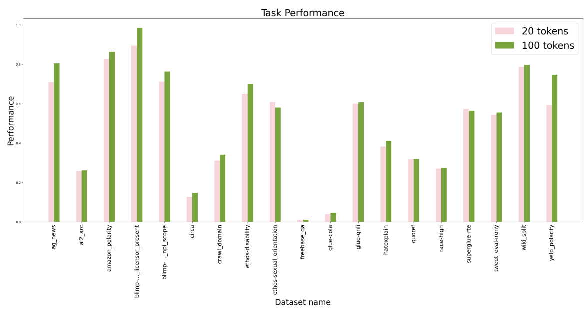


Figure 8: Average performance of prompt tokens length (20/100) in all tasks

7.1.1 Prompt

From Fig. 8, we notice that the performance of 100 tokens slightly outperforms that of 20 tokens, and this verifies the suggestion in (Lester et al., 2021), which shows that 100-tokens is more fitful for model size around 10^8 than 20-tokens. However, in Fig. 8, we can also find in some tasks, the average performance of 20 tokens is better or close to 100 tokens, like ai2_arc, ethos-sexual_orientation, glue-qnli, glue-rte, quoref, race-high, and superglue-rte. We can find that these tasks are in one of the following categories: MQA(Multiple-choice Question Answering), NLI(Natural Language Inference), or simple classification tasks. Thus, as Liu et al. (2021) reveals, shorter prompt token length can perform well in these simple tasks, and longer prompt token length is more fitful for those challenging tasks.

7.2 Exeperiment Result

7.2.1 MAML

Learning Method	Baseline	MAML													
Downstream		prompt							adapter						
Upstream		prompt	model	model + prompt	meta_prompt	meta_prompt + prompt	model + adapter + prompt	adapter + prompt	adapter	model	model + adapter	meta_adapter	meta_adapter + adapter	model + adapter + prompt	adapter + prompt
ag_news	0.86	0.86	0.63	0.56	0.85	0.85	0.83	0.84	0.69	0.53	0.61	0.85	0.78	0.02	0.07
ai2_arc	0.23	0.29	0.25	0.30	0.24	0.26	0.25	0.27	0.21	0.24	0.25	0.20	0.23	0.24	0.22
amazon_polarity	0.91	0.89	0.90	0.92	0.82	0.88	0.91	0.90	0.85	0.57	0.75	0.77	0.89	0.61	0.84
blimp-sentential_negation_npi_licensor_present	1.00	1.00	1.00	0.99	0.95	1.00	1.00	1.00	0.60	0.56	0.53	0.52	0.56	0.93	0.51
blimp-sentential_negation_npi_scope	0.93	0.57	1.00	0.66	0.59	0.74	1.00	0.76	0.51	0.51	0.52	0.51	0.55	0.52	0.53
circa	0.45	0.17	0.20	0.21	0.14	0.06	0.20	0.07	0.02	0.10	0.04	0.13	0.09	0.00	0.13
crawl_domain	0.33	0.43	0.20	0.18	0.41	0.40	0.20	0.44	0.23	0.27	0.26	0.19	0.23	0.11	0.29
ethos-disability	0.72	0.81	0.71	0.70	0.59	0.74	0.79	0.72	0.51	0.25	0.31	0.69	0.67	0.32	0.51
ethos-sexual_orientation	0.64	0.47	0.60	0.63	0.46	0.57	0.72	0.68	0.50	0.31	0.29	0.48	0.49	0.51	0.40
freebase_qa	0.00	0.01	0.02	0.03	0.00	0.03	0.04	0.01	0.01	0.06	0.06	0.00	0.01	0.03	0.03
glue-cola	0.09	0.04	0.05	0.06	0.05	0.06	0.04	0.05	-0.06	0.00	0.00	0.04	0.02	0.00	0.01
glue-qnli	0.61	0.53	0.71	0.67	0.56	0.53	0.71	0.57	0.50	0.62	0.62	0.52	0.55	0.69	0.50
hatexplain	0.42	0.38	0.48	0.44	0.39	0.39	0.45	0.40	0.20	0.30	0.13	0.38	0.29	0.03	0.25
quoref	0.29	0.33	0.31	0.25	0.29	0.36	0.31	0.38	0.29	0.41	0.39	0.27	0.28	0.28	0.34
race-high	0.24	0.27	0.30	0.30	0.24	0.26	0.31	0.26	0.19	0.31	0.31	0.16	0.23	0.31	0.25
superglue-rte	0.50	0.54	0.61	0.56	0.56	0.57	0.61	0.54	0.54	0.55	0.55	0.54	0.53	0.58	0.53
tweet_eval-irony	0.57	0.57	0.56	0.55	0.53	0.56	0.55	0.57	0.53	0.42	0.35	0.54	0.30	0.14	0.34
wiki_split	0.80	0.80	0.80	0.77	0.78	0.80	0.79	0.80	0.80	0.72	0.75	0.77	0.78	0.00	0.77
yelp_polarity	0.62	0.88	0.92	0.92	0.73	0.82	0.92	0.79	0.07	0.17	0.35	0.83	0.13	0.16	0.20

Table 2: MAML

7.2.2 Multitask

Learning Method	Baseline	Multi-task													
Downstream		prompt							adapter						
Upstream		prompt	model	model + prompt	meta_prompt	meta_prompt + prompt	model + adapter + prompt	adapter + prompt	adapter	model	model + adapter	meta_adapter	meta_adapter + adapter	model + adapter + prompt	adapter + prompt
ag_news	0.86	0.81	0.84	0.84	0.84	0.84	0.83	0.82	0.21	0.25	0.38	0.84	0.64	0.11	0.32
ai2_arc	0.23	0.25	0.27	0.29	0.23	0.25	0.27	0.26	0.24	0.26	0.26	0.23	0.22	0.26	0.23
amazon_polarity	0.91	0.88	0.93	0.93	0.75	0.80	0.93	0.87	0.63	0.56	0.44	0.86	0.88	0.62	0.92
blimp-sentential_negation_npi_licensor_present	1.00	0.96	1.00	1.00	0.99	0.99	1.00	0.96	0.57	0.98	0.99	0.55	0.74	1.00	0.57
blimp-sentential_negation_npi_scope	0.93	0.52	1.00	1.00	0.53	0.72	1.00	0.63	0.52	0.58	0.59	0.52	0.57	0.55	0.53
circa	0.45	0.08	0.22	0.19	0.15	0.11	0.14	0.07	0.02	0.03	0.02	0.08	0.12	0.00	0.10
crawl_domain	0.33	0.41	0.25	0.26	0.41	0.43	0.29	0.39	0.27	0.25	0.20	0.18	0.27	0.17	0.30
ethos-disability	0.72	0.65	0.77	0.77	0.57	0.65	0.73	0.67	0.60	0.64	0.64	0.59	0.60	0.46	0.49
ethos-sexual_orientation	0.64	0.54	0.64	0.65	0.47	0.61	0.63	0.54	0.50	0.50	0.51	0.47	0.46	0.49	0.55
freebase_qa	0.00	0.04	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.05	0.06	0.00	0.02	0.06	0.01
glue-cola	0.09	0.08	0.06	0.07	0.00	0.02	0.03	0.06	-0.05	-0.05	-0.05	-0.01	0.01	0.00	0.03
glue-qnli	0.61	0.55	0.69	0.69	0.54	0.56	0.72	0.55	0.51	0.66	0.67	0.53	0.52	0.71	0.53
hatexplain	0.42	0.39	0.44	0.42	0.40	0.41	0.46	0.39	0.14	0.35	0.36	0.37	0.28	0.12	0.25
quoref	0.29	0.34	0.38	0.39	0.28	0.28	0.36	0.32	0.30	0.41	0.41	0.31	0.28	0.39	0.32
race-high	0.24	0.24	0.33	0.33	0.24	0.24	0.33	0.24	0.21	0.33	0.33	0.23	0.25	0.34	0.23
superglue-rte	0.50	0.54	0.60	0.60	0.54	0.53	0.58	0.54	0.52	0.60	0.59	0.53	0.52	0.64	0.59
tweet_eval-irony	0.57	0.57	0.54	0.55	0.53	0.56	0.59	0.57	0.57	0.48	0.40	0.54	0.49	0.49	0.54
wiki_split	0.80	0.80	0.81	0.80	0.81	0.80	0.80	0.80	0.74	0.67	0.67	0.79	0.75	0.64	0.78
yelp_polarity	0.62	0.51	0.93	0.94	0.38	0.15	0.94	0.86	0.01	0.48	0.30	0.86	0.02	0.38	0.04

Table 3: Multitask