# ADL 2022fall Final Project Report
# CrossFit Classification Tasks by Parameter-Efficient Fine-Tuning

石旻翰
National Taiwan University,
Electrical Engineering

顏伯傑
National Taiwan University,
Electrical Engineering

陳光銘
National Taiwan University,
Mechanical Engineering

陳志臻
National Taiwan University,
Mechanical Engineering

## 1 Abstract

In recent year, Parameter-Efficient Fine-Tuning, like prompt and adapter, becomes more popular for adapting Pretrained Language Model to rapidly fine-tuning on downstream tasks. Since it takes the advantage of few parameters tuning, some scientists with less computational resources encourage to use the power of large-sized Pretrained Language Model, like T5. However, some hindrances prevent prompt/adapter from reaching their true potential. For example, prompt is sensitive for its initialization. To tackle these issues, we take CrossFit Challenge[17], and conduct some experiments, like pretraining prompt/adapter in upstream, to find a better initialization of them for downstream tasks. Also, we try to train the large-sized Pretrained Language Model, but the experiment results suggest that it does not a better way to further improve the performance than using prompt/adapter. Furthermore, our work shows a great power of parameter efficiency and has a satisfying performance among downstream tasks that is tuned with Multitask Learning, which outperforms those tuned with MAML. We provide some discussions on these results.

## 2 Introduction of Task Definition

In this work, we want to discover the few-shot learning ability and efficiency of some Parameter-Efficient Fine-Tuning methods, like prompt[12], and adapter[10]. Thus, we adopt CrossFit challenge[17], which contains a lot of few-shot Natural Language Processing(NLP) tasks datasets. Our baseline follows the settings of classification selection in CrossFit that train pretrained language models (PLMs), like BART[13] and T5[15], in upstream learning stage with some classification tasks, and that fine-tune it with a few steps in downstream classification tasks. The selected tasks in upstream are totally different from those in downstream. CrossFit[17] shows that this framework has a great performance when using Model-Agnostic Meta-Learning(MAML)[7] or Multitask Learning(MTL)[5], and MTL is a stronger baseline. However, our work mainly focuses on parameter-efficient fine-tuning, no matter in upstream or downstream. The number of parameter in BART[13] or T5[15] is quite large, making the computational cost in this work consuming. Thus, we try to use the framework in CrossFit[17] and replace the tuning target(PLMs) with prefix prompt[12] or adapter[10]. Also, we compare between our experiment results and our baseline in analyzing parameter-efficiency and calculating performance with Average Relative Gain(ARG), which can demonstrate the few-shot learning ability and parameter efficiency of our proposed work.

## 3 Related Work

### 3.1 CrossFit

CrossFit[17] consists of several NLP few-shot tasks in different categories, like Classification and Question Answering, and all of these are unified to text-to-text format. Also, CrossFit[17] shows that the few-shot learning ability on unseen tasks can be improved via an upstream

learning stage using a set of seen tasks, and that the selection of upstream learning tasks can significantly influence few-shot performance on unseen tasks.

### 3.2 Pretrained Language Model

**BART**[1]
BART[13] is a denoising autoencoder for pretraining sequence-to-sequence models. It is trained by corrupting text with an arbitrary noising function, and learning a model to reconstruct the original text. It uses a standard Transformer-based neural machine translation architecture. It uses a standard seq2seq architecture with a bidirectional encoder (like BERT[6]) and a left-to-right decoder (like GPT[4]). This means the encoder's attention mask is fully visible, like BERT[6], and the decoder's attention mask is causal, like GPT[4].

**T5**[2]
T5[15] is a Transformer-based architecture using text-to-text format. Every task is cast as feeding the model text as input and training it to generate some target text. This allows for the use of the same model, loss function, hyperparameters, etc. across our diverse set of tasks.

### 3.3 Adapter

Adapter proposed by [10], is a lightweight module introduced for the transformer-based architecture. It has become popular in NLP tasks with several variants. Instead of fine-tuning the entire pretrained model, Adapter adds some extra trainable parameters and we train with these extra parameters with freezing the original PLMs to achieve parameter-efficient fine-tuning.

### 3.4 Prompt

Prefix prompt tuning, an effective mechanism for learning soft prompts to condition frozen language models to perform specific downstream tasks, is first proposed by [14]. Prefix prompt means adding prompt tokens in the front of each layer in transformer-based model. Their work finds that tuning prefix prompt, whose size is only lower than 0.1% of pretrained model parameter, can perform well in low-data settings with careful setting in initializing prompt tokens. Furthermore, in the work of [12], they claim that when the size of PLM exceeds billions of parameters, their work matches the strong performance of model tuning, i.e all model weights are tuned. This shows that prompt tuning becomes more competitive with scale.

## 4 Approach

### 4.1 Setup

**Upstream Learning Stage**
First, we select classification tasks listed in `train_classification_test_classification.json` in CrossFit Github repo[3] as our upstream training tasks. Next, we use MAML[7] or MTL[5] as our training strategy to update the PLM parameters(for baseline) or Prompt/Adapter parameters(for our work). The detailed settings for hyperparameters in upstream for MAML or MTL can be found in Sec. 3.4.

**Downstream Fine-tuning Stage**
In downstream, we also use the classification tasks listed in the json file as our downstream testing tasks. We follow the approach proposed by CrossFit[17], and do some modifications: For each combination of (learning rate, batch size, N-way-K-shot), fine-tune the prompt/adpater parameters with a little steps, inferencing, and record the best performance among all combinations for each task. The detailed setting for hyperparameters in downstream for MAML or MTL can found in Sec. 3.4.

## 4.2 Evaluation Metric

Since we select the classification tasks for our experiment, the evaluation metric for each task performance is either Accuracy or F1-score, which is formulated as following respectively:

$$\text{Acc} = \frac{\text{pred} == \text{ground truth}}{len\,(\text{dataset})} \times 100\% \quad [4.1]$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad [4.2]$$

where

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad [4.3]$$

However, since we want to evaluate the efficiency of parameter-efficient fine-tuning, we adopt **Average Relative Gain(ARG)** to demonstrate the gap between baseline proposed by CrossFit[17] and our work. We define $P^i$ as the performance of our work's method in the ith task, and $P_0^i$ is the performance of baseline in the ith task, and there are n tasks in our $T_{test}$. ARG can be formulated as:

$$\text{ARG} = \frac{1}{n} \sum_{i=1}^{n} (\frac{P^i - P_0^i}{P_0^i} \times 100\%) \quad [4.4]$$

Also, the following tables shows the evaluation metrics we used in each task.

| Evaluation metrics of each task | | | | | |
|---|---|---|---|---|---|
| Tasks | anli | dbpedia_14 | emo | ethos-race | ethos-religion |
| Metric | F1 | F1 | F1 | F1 | F1 |

| Evaluation metrics of each task | | | | | |
|---|---|---|---|---|---|
| Tasks | financial_phrasebank | superglue-cb | tab_fact | wiki_qa | yelp_polarity |
| Metric | F1 | Acc | F1 | F1 | F1 |

Table 1: Evaluation metric for each task

## 4.3 Parameter-Efficient Fine-Tuning Algorithm

Refering to the algorithm proposed in [8], we do some modifications and reformulate the MAML algorithm as following:

**Algorithm 1** Parameter-Efficient Fine-Tuning Learning Algorithm

$\mathcal{T} = \{T_1, T_2, ...\}$: A set of training tasks
$\alpha, \beta$: Outer lr, Inner lr
$\theta$: PLM parameters
$\{\phi_1, \phi_2, ...\}$: Trainable elements of prompt and adapter
$\psi = [\theta; \phi_1; \phi_2; ...]$: All parameters of the model

Randomly initialize $\{\phi_1, \phi_2, ...\}$, and freeze $\theta$

**while** not done **do**
    **for** $T_i \in \mathcal{T}$ **do**
        Evaluate $\nabla_\psi \mathcal{L}_{T_i}(f_\psi)$ with respect to K samples
        Compute adapted parameters with gradient descent: $\psi_i' = \psi - \beta \nabla_\psi \mathcal{L}_{T_i}\left(f_\psi\right)$
        Update $\psi$ with $\psi_i'$
    **end for**
    $\psi' = \psi - \alpha \nabla_\psi \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{T_i}(f_{\psi_i'})$
    $\psi \leftarrow \psi'$
**end while**
**return** $\psi$

## 4.4 Hyperparameter Setting

**MAML**

- Upstream Learning Stage
  - Outer Learning Rate: 0.00003 for BART, 0.0001 for T5
  - Inner Learning Rate: 0.00001 for BART, 0.0001 for T5
  - Inner Batch Size: 4
  - Train Batch Size: 1
  - Epochs: 40
  - Adapter Hidden dim: 64
  - Prompt token length: 100
- Downstream Fine-tuning Stage
  - Learning rate: [1e-5, 3e-5, 5e-5] for BART, [1e-5, 3e-5, 1e-4] for T5, [1e-2, 5e-3, 1e-4] for prompt, [1e-4, 5e-5, 1e-6] for adapter
  - Total step: 1000
  - Eval step: 100

- Batch size: [4, 8]

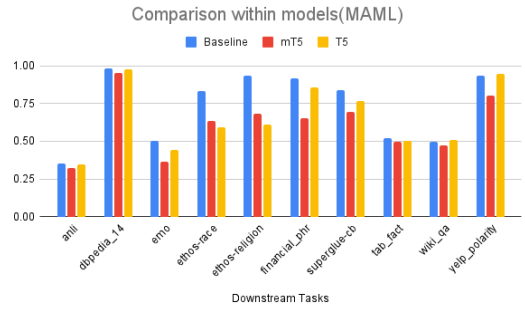
**MTL**

- Upstream Learning Stage
  - Learning Rate: 0.00003 for BART, 0.0001 for T5
  - Total Step: 34600
  - Train Batch Size: 32
  - Epochs: 10
  - Adapter Hidden dim: 64
  - Prompt token length: 100
- Downstream Fine-tuning Stage
  - Learning rate: [1e-4, 5e-5, 1e-5] for BART, [1e-5, 3e-5, 1e-4] for T5, [1e-2, 5e-3, 1e-3] for prompt, [1e-4, 5e-5, 1e-6] for adapter
  - Total step: 1000
  - Eval step: 100
  - Batch size: [4, 8]

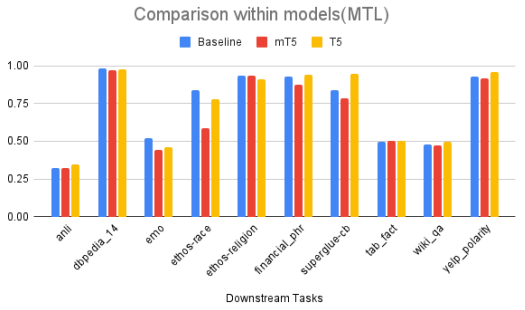
# 5 Experiment

## 5.1 Change PLM

Here we compared our baseline model BART with alternative pretrained language models including T5[15] and mT5 [16] in MAML(Fig.1a) and MTL(Fig.1b) respectively. The overall performance of BART is better than T5 and mT5. We also found that T5 did better than mT5 in most of the tasks for both settings.




(a) Comparison within models tuned by MAML

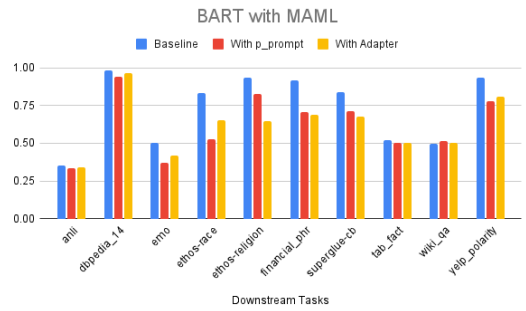



(b) Comparison within models tuned by MTL

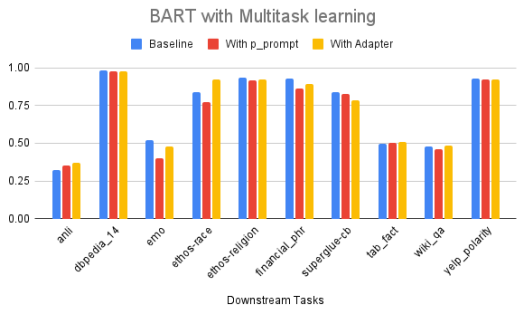
Figure 1: Experiment results of changing PLM for MAML and MTL

## 5.2 Parameter-Efficient Fine-Tuning

Also, we compared our baseline with the prefix prompt setting and adapter setting(Fig.2). Here we can find that the baseline methods outperform than all the other two settings. However, the number of tuned parameters has a large gap, and we will discuss this in the next section.




(a) Comparison within baseline, prompt, and adapter tuned by MAML




(b) Comparison within baseline, prompt, and adapter tuned by MTL

Figure 2: Experiment results of using Parameter-Efficient Fine-tuning for MAML and MTL

# 6 Discussion

In this part, we want to discuss about the experiment results of changing PLMs, and the ARG of our experiment combinations are listed in Table 2. In our experiments on MAML, we find that T5-v1.1-base has an average performance drop by 9% from baseline, and mT5-base has 13% drop, in MTL, T5-v1.1-base exceeds the baseline by 0.7% and mT5-base has a drop of 5.8%. From the results, we can not directly conclude that BART is better than T5 or mT5, because in our experiments, we do not ensure whether T5 or mT5 model has converged to its best performance in upstream training within limited time. We also lack computational resources to find the hyperparameters setting that is suitable for them in both downstream and upstream. Even though we might not find the best settings for them, we fix the training steps they used in CrossFit[17], so we can derive that BART is more easily to converge than T5 or mT5 within a limited training steps.

| ARG of different PLMs of MAML/MTL | |
|---|---|
| T5 MAML | -15.9% |
| mT5 MAML | -9.2% |
| T5 MTL | 0.7% |
| mT5 MTL | -5.8% |

Table 2: ARG of different PLMs of MAML/MTL

From Table 3, we can see that in MAML, both average performance of Adapter and Prompt have about 13% drop from the baseline. On the other hand, in MTL, the average performances are close to the baseline. But this result doesn't imply that MTL is definitely better than MAML, because we didn't do validation during upstream training and we only save the last-stepped model, maybe we miss a better model of MAML. Even though the choice of the model may not

| ARG and tunable parameter ratio of MAML/MTL with Prompt/Adapter | | |
|---|---|---|
| Combination | ARG | tunable parameter ratio |
| MAML prompt | -13.8% | 0.05% |
| MAML adapter | -13.2% | 4% |
| MTL prompt | -3.5% | 0.05% |
| MTL adapter | 0.7% | 4% |

Table 3: ARG and tunable parameter ratio of MAML/MTL with Prompt/Adapter

be the best, we can still claim that the result of using MTL with Adapter or Prompt is a great success due to the baseline-closed performances and the tunable parameters being only 4% and 0.05% of the number of parameters in BART, respectively.

# 7 Conclusion

Overall, we think that our work has a great performance in parameter-efficiency, especially in MTL. Although in CrossFit[17], they claimed that MTL is a stronger baseline than MAML, we think that MAML still works well in our work. However, we also try to explain the passable performance of Parameter-Efficient Fine-tuing with MAML. From previous work[9][11], we know that both MAML and prompt are sensitive to random seeds, but we don't have a sufficient time to experiment it with several seeds. Thus, one of our future work is doing a deep insight to how the seeds influence the results. The other one is the issue in upstream learning stage. We don't do any validation in upstream learning stage, and our checkpoint is the last step of upstream. This may lead to that our choice is an overfitting or not a well-performed model, or luckily, a well-performed model, and thus cause the results in downstream have a drop from MTL(a well-performed model) to MAML(not a well-performed model). Therefore, the other one of our future work is doing validation in upstream learning stage, to minimize the probability of the not well-selected issue, and see whether it helps or not.

# 8 Work Distribution

1. 石旻翰: Baseline, BART with prefix prompt, presentation, report.
2. 顏伯傑: BART with Adapter, presentation, report.
3. 陳光銘: Pretraining, mT5, T5, presentation, report.
4. 陳志臻: mT5, T5, presentation, report.

# References


[1] URL: https://paperswithcode.com/method/bart.

[2] URL: https://paperswithcode.com/method/t5.

[3] URL: https://github.com/INK-USC/CrossFit.

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/ARXIV.2005.14165. URL: https://arxiv.org/abs/2005.14165.

[5] Rich Caruana. "Multitask learning." In: *Machine learning* 28.1 (1997), pp. 41–75.

[6] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. DOI: 10.48550/ARXIV.1703.03400. URL: https://arxiv.org/abs/1703.03400.

[8] Mozhdeh Gheini, Xuezhe Ma, and Jonathan May. *Know Where You're Going: Meta-Learning for Parameter-Efficient Fine-Tuning*. 2022. DOI: 10.48550/ARXIV.2205.12453. URL: https://arxiv.org/abs/2205.12453.

[9] Yuxian Gu et al. *PPT: Pre-trained Prompt Tuning for Few-shot Learning*. 2021. DOI: 10.48550/ARXIV.2109.04332. URL: https://arxiv.org/abs/2109.04332.

[10] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP*. 2019. DOI: 10.48550/ARXIV.1902.00751. URL: https://arxiv.org/abs/1902.00751.

[11] Yukun Huang, Kun Qian, and Zhou Yu. *Learning a Better Initialization for Soft Prompts via Meta-Learning*. 2022. DOI: 10.48550/ARXIV.2205.12471. URL: https://arxiv.org/abs/2205.12471.

[12] Brian Lester, Rami Al-Rfou, and Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. 2021. DOI: 10.48550/ARXIV.2104.08691. URL: https://arxiv.org/abs/2104.08691.

[13] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. DOI: 10.48550/ARXIV.1910.13461. URL: https://arxiv.org/abs/1910.13461.

[14] Xiang Lisa Li and Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. DOI: 10.48550/ARXIV.2101.00190. URL: https://arxiv.org/abs/2101.00190.

[15] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019. DOI: 10.48550/ARXIV.1910.10683. URL: https://arxiv.org/abs/1910.10683.

[16] Linting Xue et al. *mT5: A massively multilingual pre-trained text-to-text transformer*. 2020. DOI: 10.48550/ARXIV.2010.11934. URL: https://arxiv.org/abs/2010.11934.

[17] Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. *CrossFit: A Few-shot Learning Challenge for Cross-task Generalization in NLP*. 2021. DOI: 10.48550/ARXIV.2104.08835. URL: https://arxiv.org/abs/2104.08835.