Revisiting Parameter-Efficient Tuning: Are We Really There Yet?

Guanzheng Chen[1], **Fangyu Liu**[2], **Zaiqiao Meng**[3], **Shangsong Liang**[1,4,*]
[1]Sun Yat-sen University [2]University of Cambridge [3]University of Glasgow
[4]Mohamed bin Zayed University of Artificial Intelligence
guanzzh.chen@gmail.com, fl399@cam.ac.uk
zaiqiao.meng@glasgow.ac.uk, liangshangsong@gmail.com

Abstract

Parameter-Efficient Tuning (PETuning) methods have been deemed by many as the new paradigm for using pretrained language models (PLMs). By tuning just a fraction amount of parameters comparing to full model finetuning, PETuning methods claim to have achieved performance on par with or even better than finetuning. In this work, we take a step back and re-examine these PETuning methods by conducting the first comprehensive investigation into the training and evaluation of them. We found the problematic validation and testing practice in current studies, when accompanied by the instability nature of PETuning methods, has led to unreliable conclusions. When being compared under a truly fair evaluation protocol, PETuning cannot yield consistently competitive performance while finetuning remains to be the best-performing method in medium- and high-resource settings. We delve deeper into the cause of the instability and observed that the number of trainable parameters and training iterations are two main factors: reducing trainable parameters and prolonging training iterations may lead to higher stability in PETuning methods.[1]

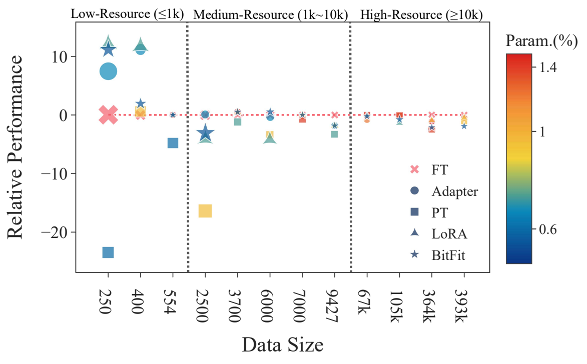

Figure 1: The relative performance difference of PETuning methods, i.e., Adapter, prefix tuning (PT), LoRA, BitFit, comparing with the full finetuning (FT) over different training data size of 12 tasks from GLUE and SuperGLUE. The tasks and their split into the three resource bands are illustrated in Appx. §B.1. The size of each point denotes the standard deviation and the colours of PETuning methods denote the percentage of trainable parameters over different tasks compared to full finetuning. The key takeaway message is that PETuning methods outperform finetuning only in the low-resource tasks but remain on par or behind in medium and high-resource settings.

1 Introduction

Pretrained Language Models (PLMs) such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have orchestrated tremendous progress in NLP in the past few years, achieving state of the art on a large variety of benchmarks such as GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019). Most successful applications of PLMs follow the pretraining-and-finetuning transfer learning paradigm (Devlin et al., 2019), where PLMs are used as backbones to be combined with additional parameters and finetuned on downstream tasks in an end-to-end manner. Whilst being simple and effective, such paradigm requires task-specific tuning of the full model that consists of hundreds of millions (Devlin et al., 2019; Liu et al., 2019), or even billions (Radford et al., 2019; Brown et al., 2020; Raffel et al., 2020) of parameters for each task, which is time-consuming and resource-intensive.

To avoid full model finetuning, there has been a surge of studies on **P**arameter-**E**fficient **Tuning** (PETuning) methods, which aim to tune the PLMs by adjusting lightweight trainable parameters while keeping most pretrained parameters frozen. Various ways have been used in these PETuning methods to introduce the lightweight trainable parameters. Adapter (Houlsby et al., 2019; Pfeiffer et al., 2020) is one of these that injects a small portion of *model-level* parameters within each trans-

[*]Corresponding author.
[1]Our code is available at `https://github.com/guanzhchen/PETuning`.

former (Vaswani et al., 2017) layer of the pretrained language model. Prompt-tuning (Qin and Eisner, 2021; Liu et al., 2021b; Lester et al., 2021) is another class of methods that introduce trainable continuous embeddings into the original sequences of input token embeddings to augment the PLMs on the *feature level*. Diff-pruning (Guo et al., 2021) learns and updates additional sparse diff-vector for all pretrained parameters, and LoRA (Hu et al., 2022) learns low-rank matrices to approximate the updated matrices, both of which update the PLMs on the *parameter level*. Moreover, BitFit (Ben Zaken et al., 2022) *partially tunes* the bias terms of PLMs, without even introducing any new parameters. More details for these methods can be seen in Appx. §A.

Given the various exciting progresses of PETuning methods that all seem to demonstrate their competitive performance with higher training efficiency, the idea that PETuning could be a new general paradigm in place of full finetuning for transfer learning in NLP becomes never more tempting (Liu et al., 2021a). We, however, argue that current evidences are insufficient to support the complete overthrow of full finetuning. First, we point out that the current evaluation strategy, i.e., the development set is used for both early stopping and reporting results, used in a number of studies for PETuning (Lester et al., 2021; Vu et al., 2022; Liu et al., 2022; Pfeiffer et al., 2021) does not provide fair model comparisons. This essentially causes data leakage that results in misleading conclusions (§2). Second, statistical significance is rarely reported when comparing PETuning methods. This is an especially crucial issue as we show that the finetuning and PETuning processes are inherently unstable due to various randomness, such as weight initialization and training data order (§3.3).

To fairly compare these tuning strategies, this study conducts a comprehensive re-examination on the effectiveness of PETuning methods. **Our main contributions** are: **1)** We conduct controlled experiments (§2) and reveal the fundamental flaw of the current evaluation scheme (i.e., its failure to assess generalisation) and how that leads to misinterpretations of the progress in the field. **2)** We offer a more reliable practice for model selection that is not prone to overfitting. **3)** We revisit the performance of PETuning in comparison with finetuning across tasks with various, and have reached very different conclusions on different data scales. **4)** We conduct the first comprehensive study to investigate the stability of off-the-shelf PETuning methods and identify the main contributing factors.

Key Findings: **1)** Finetuning cannot be fully replaced so far, since there is no PETuning method that can consistently outperform finetuning across all tasks and settings. We conclude that PETuning may be more suitable for low-resource tasks, but struggle on medium-resource tasks and fall behind finetuning across the board on high-resource tasks (see Figure 1). **2)** All the PETuning methods unanimously show instability across different random seeds similar to finetuning (Dodge et al., 2020), where the randomness comes from both weight initialisation and training data order. **3)** We found prompt-tuning lags far behind finetuning, which is a very different conclusion from previous studies. We show that prompt-tuning is highly unstable and cannot robustly and consistently re-produce its reported competitive performance (usually reported as a single run or the optimal run across multiple episodes (Lester et al., 2021; Liu et al., 2022)) in our fair evaluation setup. **4)** Within each PETuning method, reducing the size of trainable parameters is likely to yield better stability (but not necessary to yield better or poorer performance). **5)** The stability of PETuning methods is substantially proportional to the scale of training data, and we further highlight the most crucial factor behind is the number of training iterations.

For the rest of the paper, we begin with the analysis on why the current evaluation protocol can be flawed (§2), and follow with a rigorous re-examination with a fairer protocol to benchmark the performance and stability of PETuning (§3).

2 The Broken Protocol

GLUE[2] and SuperGLUE[3] have become the *de facto* benchmarks for verifying model effectiveness in Natural Language Understanding. For the sake of validity and fairness of the evaluation, the labels of test sets in these benchmarks are not released. Instead, web portals are provided for submitting and evaluating the prediction results. Due to the limited number of allowed evaluation submissions to these benchmarks, a large number of works have followed a common practice that the model performance is only reported and compared based on the dev sets rather than the real test sets, where the dev

[2] `https://gluebenchmark.com.`
[3] `https://super.gluebenchmark.com.`

set is treated as the "test set" (Lester et al., 2021; Vu et al., 2022; Liu et al., 2022; Pfeiffer et al., 2021).

While this practice is a convenient approximation of model performance as it allows quickly obtaining results from large-scaled experiments, there has been a serious data leakage problem in this setting: a single set is often used for both validating and testing the model. Therefore, the reported results under such setting might come from overly-optimistic checkpoints since early stopping is applied on the same set. We argue that such practice breaches the standard train/dev/test paradigm and compromises fair and rigorous comparison, leading to unreliable conclusions and misunderstandings of the examined models.

To verify the above concerns, we scrutinise the broken status-quo protocol by comparing it with a newly defined rigorous evaluation protocol. The new protocol has strictly separated sets for validation and testing. We provide comprehensive analyses to reveal the negative effects of using the dev sets for both checkpoint selection (i.e., early stopping) and testing.

Compared Methods. We have chosen four representative PETuning methods: **Adapter**, **Prompt-tuning (PT)**, **LoRA**, and **BitFit**, which correspond to model-level, feature-level, parameter-level, and partial-finetuning PETuning methods, respectively[4].

For Adapter, we use the *Pfeiffer* architecture (Pfeiffer et al., 2020) since it has reported better performance than others. For Prompt-tuning, due to the poor performance of standard prompt tuning (Lester et al., 2021) on small PLMs, e.g., base versions of Bert and RoBERTa, we adopt the settings of **prefix tuning** (Li and Liang, 2021) (or P-Tuning v2 (Liu et al., 2022)) to add continuous prompts for each transformer layer of PLMs. For LoRA & BitFit, we take the architectures from their original papers (Hu et al., 2022; Ben Zaken et al., 2022).

Evaluation Setup. We adopt RoBERTa$_{\text{base}}$ (Liu et al., 2019) as our base model, and experiment on the RTE dataset, which is a textual entailment dataset included in both GLUE and SuperGLUE.

[4]Some works of Adapter (Pfeiffer et al., 2020) and Prompt-tuning (Lester et al., 2021; Vu et al., 2022; Liu et al., 2022) adopt the problematic early stopping strategy (described in their experimental settings (Lester et al., 2021; Vu et al., 2022; Pfeiffer et al., 2021) or code bases (Liu et al., 2022)), while LoRA and BitFit adopt the standard train/dev/test paradigm.

	Evaluation loss		*Accuracy*	
	RTE$_{1-2}$	RTE$_{2-2}$	RTE$_{1-2}$	RTE$_{2-2}$
FT	**78.89**$_{\pm 1.36}$	**78.89**$_{\pm 1.36}$	79.28$_{\pm 1.9}$	**79.62**$_{\pm 2.22}$
Adapter	75.1$_{\pm 1.60}$	**76.3**$_{\pm 4.26}$	76.55$_{\pm 3.57}$	**78.42**$_{\pm 3.7}$
PT	57.55$_{\pm 2.71}$	**66.19**$_{\pm 8.51}$	57.84$_{\pm 4.85}$	**67.19**$_{\pm 11.37}$
LoRA	75.22$_{\pm 2.77}$	**75.94**$_{\pm 3.39}$	75.11$_{\pm 3.3}$	**77.7**$_{\pm 4.57}$
BitFit	70.79$_{\pm 10.38}$	**71.3**$_{\pm 10.19}$	66.76$_{\pm 12.98}$	**68.2**$_{\pm 13.72}$

Table 1: Mean and standard deviation results with different dev/test splits for RTE task across 20 runs. *Evaluation loss* and *accuracy* are the stopping metrics. Bold denotes the highest mean value for corresponding method with specific stopping metric.

We divide the original dev set of the RTE dataset by a 50%/50% split[5] (denoted by *dev.1* and *dev.2* respectively), and compare the performance over finetuning and the four PETuning methods. In particular, we use the *dev.2* set as the test set, and use the *dev.1* set or the *dev.2* set as the dev set for model selection, respectively (denoted by RTE_{1-2} or RTE_{2-2}). We set the number of epochs to 50 and early stop when validation scores do not improve for 10 consecutive epochs following Mao et al. (2022). Results will be shown for using either *evaluation loss* or *accuracy* as the stopping metrics.[6]

Results and Analyses. From Table 1, we can see that using a single set as both the dev and test sets (i.e. RTE_{2-2}) can substantially boost the performances of PETuning models, comparing with using two separate ones (i.e., RTE_{1-2}). Particularly, prefix tuning (PT) gains $\sim$10% improvements using either evaluation loss or accuracy as the stopping metric. However, such performance boost does not mean genuine improvement in terms of better generalisation.

To demonstrate this in a more intuitive way, we plot the evaluation performance on dev sets (i.e. *dev.1* and *dev.2* respectively) over training steps in Figure 2.[7] For each model, its early stopped epochs over the two sets are drastically different, suggesting that there is significant behavioural difference of the models across sets and best check-

[5]A normal way to create new dev set is to separate part of training set while using original dev set as test set, as what we do in §3.1. However, to highlight the data leakage issue from the misused early stopping with a more controlled setting, we create the new dev and test sets from the same (original) dev set with similar size and distribution and fairly compare their impact for early stopping.

[6]See Appx. §B.2 for the full hyperparameters settings.

[7]The best-performing runs of RTE_{1-2} and RTE_{2-2} are used for this visualisation.

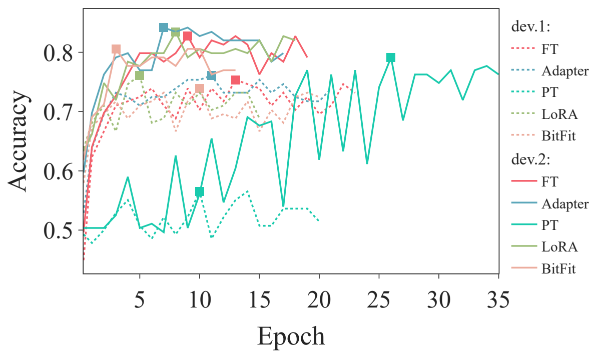

Figure 2: Comparing early stopped points selected by RTE_{1-2} and RTE_{2-2}, i.e., checkpoints with the best accuracy scores from *dev.1* and *dev.2* over training epochs. The markers denote the epochs selected by early stopping. Comparing the two checkpoint results on dev.2 (i.e. test performance), the RTE_{2-2} (same set for test and dev) checkpoint usually shows higher performance than the checkpoint selected in RTE_{1-2} by a large gap.

point selected on one set does not necessarily generalise well on the other set. In fact, the ability of models to mitigate such gap (e.g., from the best-performing checkpoints on *dev.1* to the best-performing ones on unseen *dev.2*) precisely denotes corresponding ability of generalisation, which is the most essential criteria to measure the models' effectiveness (Raschka, 2018). However, the evaluation scheme RTE_{2-2}, i.e., the broken protocol, reuses the test set multiple times during training stage, which is tantamount to leaking the test information to erase this gap, resulting in unreliable evaluation.

These observations motivates us to re-examine these PETuning methods with a fairer evaluation.

3 Experiments with Fair Evaluation

In this section, we use a fairer evaluation protocol that strictly separates dev and test sets. Based on this protocol, we conduct extensive experiments to investigate the effectiveness of PETuning methods (concluded in Figure 1). First, we experiment over a wide range of tasks under various levels of resource abundance to comprehensively compare the performance of PETuning with finetuning (§3.2). Further, we provide in-depth analyses for the instability of PETuning methods, investigating the possible causes and provide practical suggestions of using PETuning methods (§3.3).

3.1 Experimental Setup

Data Setup. We conduct experiments on 12 datasets from GLUE and SuperGLUE, which are divided into three levels according to their sizes: (1) *low-resource* (< 1k data points), including CB (de Marneffe et al., 2019), COPA (Roemmele et al., 2011), and WSC (Levesque et al., 2012); (2) *medium-resource* (1k ~10k data points), including RTE (Wang et al., 2018), MRPC (Dolan and Brockett, 2005), WiC (Pilehvar and Camacho-Collados, 2019), STS-B (Cer et al., 2017), and BoolQ (Clark et al., 2019); (3) *high-resource* (> 10k data points), including SST-2 (Wang et al., 2018), MNLI (Williams et al., 2018), QNLI (Wang et al., 2018), and QQP[8].

Since using a single set for both early stopping and testing could result in unreliable results (§2), we use separate dev and test sets for all our experiments. Specifically, the original training set of each dataset is split into new train set and dev set by a 90%/10% proportion, and the original dev set is used as the test set.[9]

Evaluation Setup. For the aforementioned four PETuning methods, i.e., Adapter, prefix tuning, LoRA, and BitFit, we again experiment with the RoBERTa$_{\text{base}}$ model (Liu et al., 2019) on our 12 datasets. All experimental results are reported across 20 runs for low- and medium-resource tasks, and 10 runs for high-resource tasks with different random seeds, respectively. We train for 50 epochs and early stop the training when evaluation loss does not decrease for 10 consecutive epochs.[10]

3.2 Analysis of Performance

From the average performance for all tasks in Table 2, we can observe that most of the PETuning methods (i.e., Adapter, LoRA, and BitFit) indeed have some performance gains when compared with finetuning. It is known that PETuning methods have far better tuning efficiency, with significantly less tuning parameters (< 2% of full model parameters), comparing with full finetuning (Mao et al., 2022). However, it remains questionable whether PETuning methods are more advantageous as the overall comparison may neglect important divergences in the wide range of tasks with different scales of training data. To provide a finer-

[8]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

[9]Ideally, the standard train-dev-test splits of GLUE and SuperGLUE should be used. However, due to the amount of experiments and evaluations need to be done in our ultra-large-scale investigation, we create our own splits instead of submitting models to the learderboards.

[10]See Appx. §B.2 for the full hyperparameters settings.

Dataset↓, Model→	FT	Adapter	PT	LoRA	BitFit
			Low-Resource		
CB	$70.00_{\pm 13.32}$	$77.49_{\pm 13.20}$	$46.55^{\downarrow}_{\pm 5.74}$	$\mathbf{82.05}^{\uparrow}_{\pm 9.62}$	$81.12^{\uparrow}_{\pm 8.94}$
COPA	$54.70_{\pm 3.36}$	$65.90^{\uparrow}_{\pm 5.42}$	$55.35_{\pm 5.07}$	$\mathbf{66.4}^{\uparrow}_{\pm 9.05}$	$56.65_{\pm 3.72}$
WSC	$\mathbf{63.46}_{\pm 0.0}$	$\mathbf{63.46}_{\pm 0.0}$	$58.7^{\downarrow}_{\pm 4.69}$	$\mathbf{63.46}_{\pm 0.0}$	$\mathbf{63.46}_{\pm 0.0}$
Avg. (Low)	$62.72_{\pm 4.58}$	$68.95^{\uparrow}_{\pm 4.83}$	$53.53^{\downarrow}_{\pm 3.22}$	$\mathbf{70.64}^{\uparrow}_{\pm 4.32}$	$67.08^{\uparrow}_{\pm 3.57}$
			Medium-Resource		
RTE	$73.77_{\pm 3.17}$	$\mathbf{73.88}_{\pm 1.88}$	$57.36^{\downarrow}_{\pm 8.01}$	$69.69^{\downarrow}_{\pm 7.89}$	$70.67_{\pm 10.77}$
MRPC	$90.54_{\pm 1.05}$	$\mathbf{91.06}_{\pm 0.63}$	$89.35^{\downarrow}_{\pm 1.31}$	$91.03_{\pm 0.95}$	$\mathbf{91.06}_{\pm 0.71}$
WiC	$65.47_{\pm 2.04}$	$65.12_{\pm 1.88}$	$62.12^{\downarrow}_{\pm 1.32}$	$61.29^{\downarrow}_{\pm 6.7}$	$\mathbf{66.0}_{\pm 1.41}$
STS-B	$90.42_{\pm 0.26}$	$90.23^{\downarrow}_{\pm 0.1}$	$89.64_{\pm 0.39}$	$\mathbf{90.47}_{\pm 0.11}$	$90.44_{\pm 0.15}$
BoolQ	$\mathbf{78.75}_{\pm 0.72}$	$76.93_{\pm 0.92}$	$75.44_{\pm 0.47}$	$76.92_{\pm 1.33}$	$76.9_{\pm 0.84}$
Avg. (Medium)	$\mathbf{79.79}_{\pm 0.99}$	$79.44_{\pm 0.74}$	$74.78^{\downarrow}_{\pm 1.62}$	$77.88^{\downarrow}_{\pm 2.02}$	$79.01_{\pm 2.22}$
			High-Resource		
SST-2	$\mathbf{94.15}_{\pm 0.0}$	$93.34^{\downarrow}_{\pm 0.31}$	$\mathbf{94.15}_{\pm 0.0}$	$\mathbf{94.15}_{\pm 0.0}$	$93.92_{\pm 0.07}$
QNLI	$\mathbf{92.40}_{\pm 0.12}$	$92.31_{\pm 0.09}$	$92.31_{\pm 0.27}$	$91.00_{\pm 0.69}$	$91.60_{\pm 1.01}$
QQP	$\mathbf{91.38}_{\pm 0.06}$	$90.28_{\pm 0.0}$	$88.90^{\downarrow}_{\pm 0.32}$	$90.45^{\downarrow}_{\pm 0.17}$	$89.28^{\downarrow}_{\pm 0.0}$
MNLI	$\mathbf{87.42}_{\pm 0.20}$	$86.88^{\downarrow}_{\pm 0.17}$	$86.30^{\downarrow}_{\pm 0.08}$	$86.96_{\pm 0.24}$	$85.50^{\downarrow}_{\pm 0.32}$
Avg. (High)	$\mathbf{91.34}_{\pm 0.09}$	$90.70^{\downarrow}_{\pm 0.12}$	$90.42^{\downarrow}_{\pm 0.14}$	$90.64^{\downarrow}_{\pm 0.21}$	$90.08^{\downarrow}_{\pm 0.29}$
Avg. (All)	79.37	**80.57**	74.68	80.32	79.72

Table 2: Mean and standard deviation results for each of the 12 tasks across finetuning (FT) and four PETuning methods. We report the F1 score for CB and MRPC, Pearson correlation for STS-B, and accuracy for other tasks (matched accuracy for MNLI). Higher is better for all metrics. One-tailed t-test is used for the comparison between PETuning and finetuning. One PETuning method outperforms (↑) or falls behind (↓) finetuning when accepting the corresponding alternative hypothesis, where p-value < 0.05 (meaning the difference is significant).

	Low	Medium	High
Adapter	↗	→	↘
PT	↘	↘	↘
LoRA	↗	↘	↘
BitFit	↗	→	↘

Table 3: Performance comparison between PETuning and finetuning on low-, medium-, and high-resource settings, respectively. Arrows indicate whether corresponding PETuning method significantly outperforms finetuning (↗), falls behind (↘), or their results across multiple runs without significant differences (→).

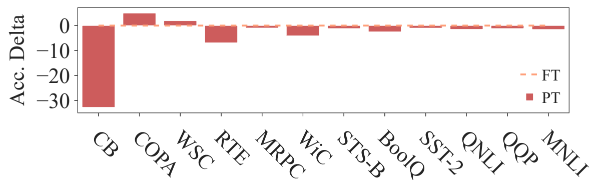

Figure 3: Relative performance differences of prefix tuning (PT) over full finetuning (FT) on the upper bounds of multi-run results. PT achieves close upper bounds compared with FT on most of the 12 tasks.

grained view for the comparison between finetuning and PETuning, we group the results of the 12 tasks in Table 2 into low-, medium-, and high-resource tasks. Whilst most PETuning methods outperform finetuning on low-resource settings, the best PETuning is merely comparable to finetuning in medium-resource tasks and lags behind finetuning in high-resource tasks. We summarise the trend in Table 3, and provide more detailed analyses in the following.

***Adapter & LoRA & BitFit* only perform better on low-resource tasks.** From Table 2, we observe that Adapter, LoRA, and BitFit obtain outstanding performance on the low-resource tasks and significantly outperform finetuning by large margins[11] (especially LoRA obtains ~8% performance gains on average). However, the trend changes when training data size gets larger. For the medium-resource tasks, only Adapter and BitFit can maintain a comparable performance with finetuning. LoRA and prefix tuning lags behind substantially. For the high-resource setting, finetuning performs consistently better than all PETuning methods.[12] In

[11]Similar observation for Adapter was previously reported in He et al. (2021). We extend it to more PETuning methods.

[12]These findings are also observed on the same task with different number of training instances. See Appx. §C.1 for more details.

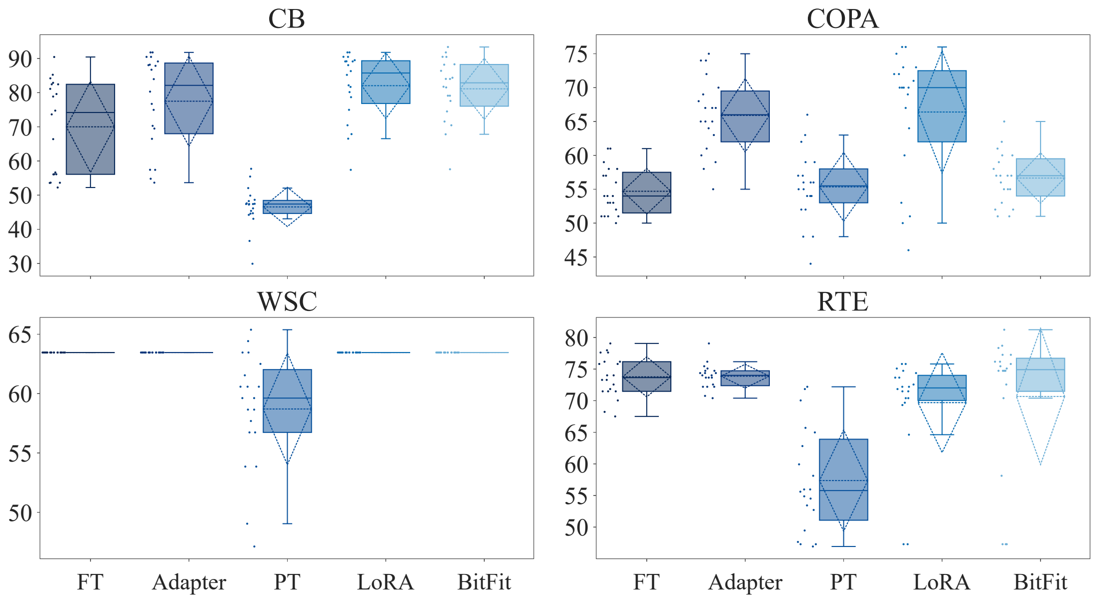

Figure 4: The experimental results over 20 different random seeds across CB, COPA, WSC, and RTE datasets, where finetuning and PETuning methods show large instability. The dashed rhombuses denote the mean (horizontal dashed line) and standard deviation (vertical distance).

particular, among the PETuning methods, Adapter obtains the highest scores on high-resource tasks. These results suggest that low-resource is the only setting where PETuning methods could outperform full finetuning.

Prefix tuning **consistently underperforms finetuning.** According to Table 2 and Table 3, finetuning beats prefix tuning by large margins on most tasks across multiple runs, contradicting to what has been reported in Liu et al. (2022). One possible reason is that prefix tuning is highly unstable to train and thus may have exploited the broken protocol more than other PETuning methods (see Figure 2). Besides using a flawed evaluation protocol, previous works on prefix tuning only report their result of a single run (Liu et al., 2022; Lester et al., 2021; Vu et al., 2022), which might lead to biased conclusion. In Figure 3 we further plot the upper bounds of these runs, and we indeed observe that the optimal run from prefix tuning achieves competitive performance compared with finetuning on many tasks. However, the results in Table 2 verify that this competitiveness would plummet across different runs by varying the random seeds. Such instability of prefix tuning leads to its poor average performance in our experiments. We further discuss this in §3.3.

Finetuning cannot be fully replaced. To summarise, PETuning has exceptional performance in resource-poor scenarios and usually outperform the more expensive full-model finetuning. However, when dataset size increases, finetuning regains dominance in medium- and high-resource setups. This indicates that finetuning cannot be fully replaced so far. We also delved deeper into understanding why finetuning lags behind PETuning on low-resource settings. Our investigation points to the different fitting capabilities of finetuning and PETuning. Specifically, finetuning is more prone to overfitting on low-resource tasks.[13]

	WI	DO	Global
FT	**55.40**$_{\pm 4.55}$	55.35$_{\pm \mathbf{3.32}}$	54.70$_{\pm 3.36}$
Adapter	**67.15**$_{\pm \mathbf{5.40}}$	66.35$_{\pm 7.36}$	65.90$_{\pm 5.42}$
PT	55.00$_{\pm 5.13}$	54.75$_{\pm \mathbf{4.97}}$	**55.35**$_{\pm 5.07}$
LoRA	63.60$_{\pm \mathbf{7.93}}$	64.60$_{\pm 8.56}$	**66.40**$_{\pm 9.05}$
BitFit	**58.40**$_{\pm \mathbf{2.29}}$	56.00$_{\pm 4.00}$	56.65$_{\pm 3.72}$

Table 4: Performance over 20 runs on COPA task, controlled by global random seeds, weight initialization (WI) random seeds, and data order (DO) random seeds, respectively. (Visualised in Figure 12 in the Appendix.)

3.3 Analysis of Stability

By revisiting the results in Table 2, we can observe that both finetuning and all PETuning methods exhibit large standard deviations on several tasks, i.e.,

[13]See Appx. §C.2 for more details and analyses.

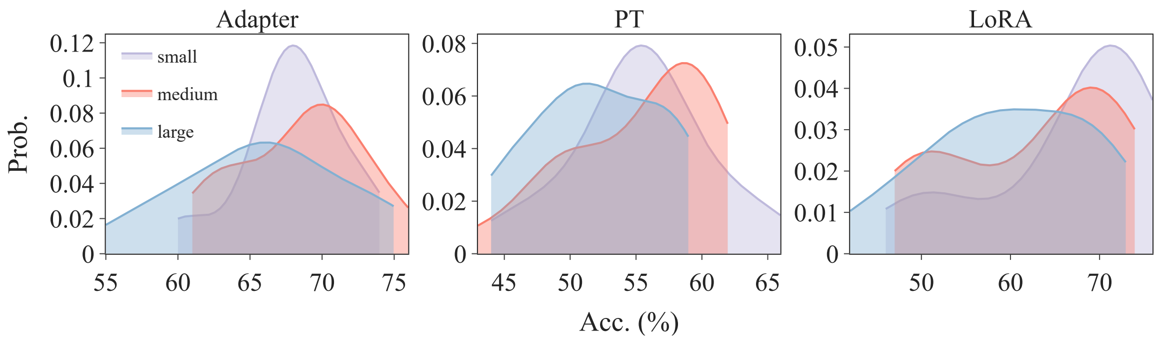

Figure 5: Performance probability density curves of Adapter, prefix tuning (PT), and LoRA over small, medium, and large parameter scales on COPA task across 20 runs. (See the numerical results and analyses in Appx. §C.3.)

CB, COPA, WSC, and RTE. To further understand this phenomenon, in Figure 4, we visualise the performance distribution of 20 runs of finetuning and PETuning methods on these tasks. Surprisingly, large fluctuations are seen on all four tasks across all methods, where the margins between the lower and upper bounds could reach over 30%. While Dodge et al. (2020); Mosbach et al. (2021) have previously identified such variation exists for finetuning, our experiments further validate that such instability also occurs in all PETuning methods and could be even more prominent in certain tasks.

This level of instability severely hampers the application of PETuning and there is a pressing need to understand the underlying cause. However, to the best of our knowledge, no previous studies have systematically discussed the instability issue in PETuning methods. In this section, we provide the first comprehensive investigation on this matter. While instability is measured as the performance differences between random seeds, we further disentangle two randomness sources (weight initialisation and training data order) to better describe model instability. We then investigate two factors that might affect model instability: (1) trainable parameter size; and (2) training data size and training iterations. Through controlled experiments, we find that model instability is reflected by both changing data order and changing weight initialisation. Reducing model size and increasing training iteration seems to have positive impact on model stability. We discuss all these points in detail in the followings.

Weight initialisation and data order work together. Instability is measured from performance changes due to randomness introduced by random seeds. Two key things impacted by random seeds are (a) the initialisation of trainable weights (including extra parameters of PETuning methods and the classification head), and (b) the order of training data fed to the model. To disentangle these two factors, following the setting in Dodge et al. (2020), we use two separate random seeds to control weight initialisation and training data order respectively, comparing with using one global random seed to control these two factors simultaneously.

Table 4 demonstrates that each of the two factors could individually lead to large standard deviations, which means the instability of PETuning methods are sensitive to either training data order, or weight initialisation, or both. This observation indicates that the sources of instability for PETuning can be multifaceted – isolating and enhancing stability via controlling individual factor can be challenging.[14]

Models with fewer trainable parameters are more stable. To investigate the impact of model size on model stability, we define three sizes, *small*, *medium*, and *large*, for each PETuning method. The three sizes correspond to the reduction factor of $\{64, 16, 2\}$ for Adapter[15], the prompt length of $\{32, 64, 128\}$ for prefix tuning, and the rank of $\{8, 16, 32\}$ for LoRA. We conduct a set of controlled experiments on the COPA task where PETuning methods exhibit high instability. We perform 20 runs for each setting and use kernel density estimation (KDE) (Chen, 2017) to estimate the probability density curves of the multi-run results.

As shown in Figure 5, for all PETuning methods, we consistently observe that the probability density curves would be progressively flatter (having lower

[14]Prior works mainly focused on obtaining better prior (e.g., prompt/weight initialisation) to improve model performance/stability but did not touch upon the multifaceted nature of instability (Pfeiffer et al., 2021; Lester et al., 2021; Vu et al., 2022).

[15]The smaller the reduction factor, the more parameters the model has.

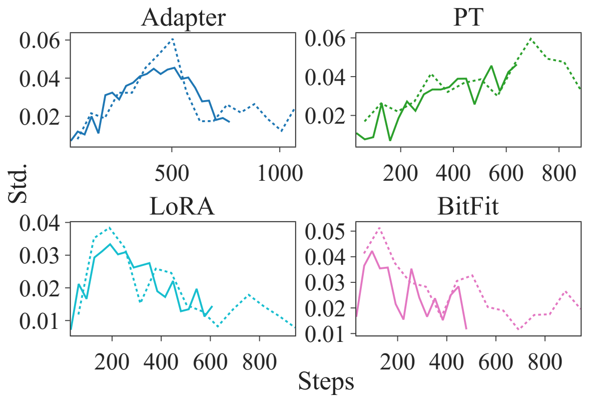

Figure 6: Standard deviations of data size in {1k (solid line), 2k (dashed line)} over training steps on WiC task across 20 runs. (See that on BoolQ task in Figure 13 in the Appendix.)

peak) as the number of parameters increase from small to large. This suggests that more trainable parameters for PETuning leads to a wider range of performance distribution, resulting in higher instability. That said, when it comes to model performance, the best-performing model usually is not the smallest one. We conjecture that models with fewer trainable parameters converge quickly to the rough global minima but could be underfitting the real data manifold.

Data size does not affect instability directly, but training steps do. Figure 1 and Table 2 suggest that PETuning methods almost always have larger standard deviations on lower-resource tasks.[16] To investigate if training data size directly affects the stability of PETuning, inspired by Mosbach et al. (2021), we compare models that are trained with randomly sampled 1k and 2k training instances from WiC training set and validated with another separately sampled 1k dev set.

In Figure 6, we observe that the solid and dashed lines of each PETuning method are substantially intertwined, which means the standard deviations (instability) of PETuning methods trained by 1k or 2k samples would not have significant differences with the same number of steps. The true underlying variable that leads to the discrepancy of instability across different training data sizes is essentially the number of training iterations/steps. As shown in Figure 6, the standard deviations of PETuning methods have an initial ascent stage where models are fitting to the training data and thus having fluctuating performance. After the ascent stage, the standard deviations substantially decrease as the number of training steps get larger. With number of epochs being fixed, the total number of iterations on small datasets is small and the standard deviation has yet to decrease, causing the higher instability in lower-resource tasks. In particular, due to the weaker fitting capabilities (Ding et al., 2022), prefix tuning (PT) has a longer ascent stage, which might need more training iterations to obtain more stable performance. That said, prolonging the training on small datasets does not necessarily enhance model performance, and the best checkpoint may still be only appearing when the standard deviation is high.

[16]This is further confirmed in Appx. §C.1.

4 Related Work

Instability of finetuning PLMs. While our study is, to the best of our knowledge, the first to systematically investigate PETuning instability, prior studies have looked into the instability of finetuning PLMs. Dodge et al. (2020) illustrated the inherent instability of finetuning by controlling random seeds and provided a new early stopping strategy to improve instability. Lee et al. (2020) proposed a new regularisation method by mixing two models based on dropout to prevent catastrophic forgetting and to improve instability. More recently, Mosbach et al. (2021) revisited the hypotheses of finetuning instability proposed by previous studies and found that optimisation difficulties can lead to vanishing gradients, which further causes finetuning instability. Zhang et al. (2021) also revealed that optimisation significantly affects the instabilities in few-sample fine-tuning.

Analysis of PETuning. As PETuning methods have become a prominent research direction, a great number of studies aim to analyse the characteristics of these methods. He et al. (2021) investigated the effectiveness of Adapter across different scales and Han et al. (2021) provided a robust strategy for training Adapter. Recently, He et al. (2022) and Mao et al. (2022) proposed a unified view to connect various PETuning methods. However, there has not been reliable validation and comparison for off-the-shelf PETuning methods in terms of stability and effectiveness, and this is where our paper bridges the gap.

5 Conclusion

This work conducted a rigorous re-examination on the current Parameter-Efficient Tuning (PETuning) methods. We demonstrated that performing early stopping and evaluation on the same dataset (a common practice used in many past studies) could lead to unreliable conclusions. This issue is more pronounced when accompanied by the instability nature of PETuning, leading to inflated results and overly optimistic estimates of PETuning approaches. We re-evaluated these PETuning methods on the performance and stability aspects on a rigorous evaluation protocol that strictly separates validation and test sets. By conducting a set of fine-grained comparisons between PETuning and finetuning, we found that PETuning methods are not consistently competitive with finetuning. Namely, prefix tuning performs poorly across tasks and most PETuning methods perform worse than finetuning on higher-resource settings. By systematically investigating the instability of PETuning methods, we found that models' instability is sensitive to both weight initialisation and training data order. We identify two major factors behind such instability: 1) models with fewer parameters are more stable within each PETuning method class; 2) more training iterations can usually reduce instability. Our overall re-examination conclude that finetuning still cannot be fully replaced by PETuning so far, and there are many key challenges for PETuning in terms of both performance and instability, which need to be addressed in future work.

Limitations

This work provides a comprehensive study and analysis for the existing popular PETuning methods, i.e., Adapter, Prompt-Tuning (prefix tuning), LoRA, and BitFit, focusing on their performance and stability. Empirically, we use standard deviations to measure the stability of these PETuning methods across multiple runs. Standard deviation is more reliable when having more number of runs. A larger number of runs would contribute to more precise estimation of such stability. We chose 20 runs for low- and medium-resource tasks and 10 runs for high-resource tasks. However, larger numbers of runs can consolidate our conclusions.

Besides, we used the available train and dev sets from GLUE and SuperGLUE to simulate a standard train/dev/test split. The conclusion would be more comparable to existing works if having access to the real testing data.

Last but not least, we covered four representative PETuning methods. However, PETuning is a fast-moving field and our conclusions do not necessarily generalise to all existing and upcoming models.

Acknowledgements

This research was partially supported by the National Natural Science Foundation of China (Grant No. 61906219), and the Mohamed bin Zayed University of Artificial Intelligence, United Arab Emirates.

References

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, pages 1877–1901.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14.

Yen-Chi Chen. 2017. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, (1):161–187.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2924–2936.

Marie-Catherine de Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The CommitmentBank: Investigating projection in naturally occurring discourse. *Proceedings of Sinn und Bedeutung*, (2):107–124.

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*, pages 4171–4186.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models. *arXiv preprint arXiv:2203.06904*.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. *arXiv preprint arXiv:2002.06305*.

William B. Dolan and Chris Brockett. 2005. Automatically Constructing a Corpus of Sentential Paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2022. PPT: Pre-trained prompt tuning for few-shot learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8410–8423, Dublin, Ireland. Association for Computational Linguistics.

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-Efficient Transfer Learning with Diff Pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 4884–4896.

Wenjuan Han, Bo Pang, and Ying Nian Wu. 2021. Robust Transfer Learning with Pretrained Language Models through Adapters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 854–861.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a Unified View of Parameter-Efficient Transfer Learning. In *International Conference on Learning Representations*.

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jiawei Low, Lidong Bing, and Luo Si. 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 2208–2222.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799.

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*.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2020. Mixout: Effective Regularization to Finetune Large-scale Pretrained Language Models. In *8th International Conference on Learning Representations*.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning. *arXiv preprint arXiv:1911.03090*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, page 552–561.

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*, pages 4582–4597.

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*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized Bert Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 2022. UniPELT: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, Dublin, Ireland. Association for Computational Linguistics.

Zaiqiao Meng, Fangyu Liu, Thomas Clark, Ehsan Shareghi, and Nigel Collier. 2021. Mixture-of-Partitions: Infusing Large Biomedical Knowledge Graphs into BERT. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4672–4681.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines. In *9th International Conference on Learning Representations*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pages 487–503.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A Framework for Adapting Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. WiC: the Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1267–1273.

Guanghui Qin and Jason Eisner. 2021. Learning How to Ask: Querying LMs with Mixtures of Soft Prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, (8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*, pages 140:1–140:67.

Sebastian Raschka. 2018. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. *arXiv preprint arXiv:1811.12808*.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pages 90–95.

Timo Schick and Hinrich Schütze. 2021. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*.

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.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1112–1122.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2021. Revisiting Few-sample BERT Fine-tuning. In *International Conference on Learning Representations*.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual Probing Is [MASK]: Learning vs. Learning to Recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5017–5033.

Appendix

A PETuning Methods

PETuning methods are unique in keeping (most) pretrained parameters of PLMs frozen and finetuning only light-weight additional parameters or a fraction of the PLM's parameters for downstream tasks.

To achieve efficient tuning of PLMs, existing PETuning methods are generally designed by two different manners: (1) training additional parameters on different levels of PLMs, including model-level (Appx. §A.1), feature-level (Appx. §A.2), and the parameter-level (Appx. §A.3), or (2) tuning partial parameters of the base model (Appx. §A.4). Figure 7 shows the difference of these PETuning methods.

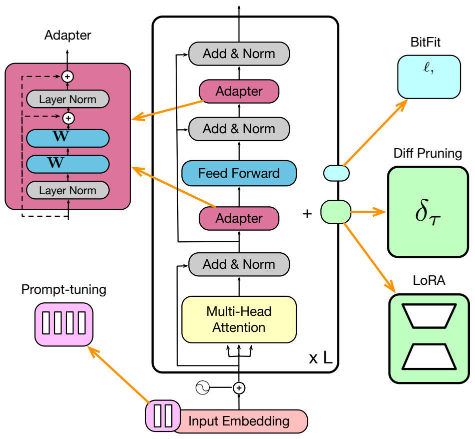

Figure 7: Different PETuning methods by adjusting trainable parameter on model level (Adapter), feature level (Prompt-tuning), parameter level (Diff Pruning and LoRA), and partial-tuning level (BitFit).

A.1 Model-Level

Adapter-Tuning. *Adapters* (Houlsby et al., 2019; Pfeiffer et al., 2020, 2021; Meng et al., 2021) are a type of PETuning approaches that insert small newly initialised parameter modules on the model-level (i.e., each transformer layer) of PLMs. In particular, these adapter modules are normally moulded by a two-layer feed-forward neural network with a bottleneck: (1) a down-projection with $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times r}$ to project the input $\mathbf{h}_i$ to a lower-dimensional space specified by bottleneck dimension r; (2) an up-projection with $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r\times d}$ to project back to the input size. Mathematically, the adapter can be defined as:

$$\mathbf{h}_a = \mathbf{W}_{\text{up}}^{\top} f\left(\mathbf{W}_{\text{down}}^{\top}\mathbf{h}_i\right), \tag{1}$$

where $\mathbf{h}_a$ is the output and $f(\cdot)$ is the activation function. During the finetuning, the model only updates the parameters of the adapter modules while keeps the underlying pretrained model fixed.

A.2 Feature-Level

Prompt-Tuning. Prompt-Tuning (Lester et al., 2021) is another type of PETuning approaches that introduce additional tunable parameters on the feature-level. Specifically, prompt-tuning introduces additional tunable prefix (or suffix) vectors, namely prompts (Zhong et al., 2021; Schick and Schütze, 2021), to extend the input text features (or the input of each transformer layer (Li and Liang, 2021; Liu et al., 2022)), and tunes only the prompts. Besides its simplicity and lightness, prompt-tuning could achieve on par performance, particularly in billions-size PLMs, and even better performance, comparing with the full finetuning (Liu et al., 2022).

A.3 Parameter-Level

Diff-Pruning. Diff-pruning (Guo et al., 2021) works on all parameters of PLMs, which aims to learn additional trainable sparse parameters for the entire PLMs. Specifically, for the pretrained parameters Θ, diff-pruning reparameterizes the task-specific model parameters Θ_τ as:

$$\Theta_\tau = \Theta + \delta_\tau, \tag{2}$$

where δ_τ denotes the trainable diff vector, which is regularised to be sparse.

LoRA. LoRA (Hu et al., 2022) focuses on the updating procedure of the language model parameters. For a pretrained weight matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$, LoRA uses trainable low-rank matrices to approximate the updates ($\Delta\mathbf{W}$) by:

$$\mathbf{W} + \Delta\mathbf{W} = \mathbf{W} + \mathbf{BA}, \tag{3}$$

where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$.

A.4 Partial Finetuning

BitFit. *Partial finetuning* aims to tune a fraction of PLMs parameters without introducing any additional ones. For example, Lee et al. (2019) only tunes the top layers, however, which usually performs much worse than full finetuning. With the principle of efficiency and effectiveness, BitFit (Ben Zaken et al., 2022) turns to tune the bias terms of PLMs to obtain competitive performance.

B General Experimental Setup

In this section, we illustrate the general task and hyperparameter settings. Apart from that, in §2 and §3, we will additionally illustrate their specific data and evaluation setups, respectively.

B.1 Task Setup

In order to extensively compare the performance and stability of PETuning methods with the full-model finetuning, we select a full set of 12 tasks across low-, medium- and high-resource scales of GLUE and SuperGLUE, including natural language inference (CB, RTE, MNLI, QNLI), question answering (COPA, BoolQ), paraphrasing (MRPC, QQP), sentiment analysis (SST-2), sentence similarity (STS-B), word sense disambiguation (WiC), and coreference resolution (WSC) tasks. According to the dataset sizes, we divide these tasks into three levels:

- **Low-Resource:** the tasks with training data size smaller than 1k, including CB, COPA, and WSC.
- **Medium-Resource:** the tasks with training data size between 1k and 10k, including RTE, MRPC, WiC, STS-B, and BoolQ.
- **High-Resource:** the tasks with training data size larger than 10k, including SST-2, QNLI, QQP, and MNLI.

B.2 Hyperparameter Setup

We adopt Roberta$_{\text{base}}$ as the base model released by Huggingface[17]. The grid search is used to select the learning rate from {1e-6, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2} and batch size from {16, 32}. We search the reduction factor from {2, 16, 64} following (Pfeiffer et al., 2021) for Adapter, the prompt length from {8, 16, 32, 64} for prefix tuning, and the scaling factor α and rank from {8, 16} for LoRA following its origin paper. There are many studies focusing on achieving better initialization by post pretraining for PETuning methods such as Adapter (Pfeiffer et al., 2021) and prompt (Vu et al., 2022; Gu et al., 2022), however, to be a fair comparison, the extra parameters of all PETuning methods are initialized randomly.

We set the number of epochs to 50 and adopt the early stopping strategy with the patience of 10 worse-performing epochs on our new development set following (Mao et al., 2022). In particular, for §2, to fully investigate the effects of early stopping on the task RTE, we use both *evaluation loss* and *accuracy* as the stopping metrics; for §3, due to the variety of evaluation metrics for the tasks, we use the *evaluation loss* as the common stopping metric.

[17] https://github.com/huggingface/transformers

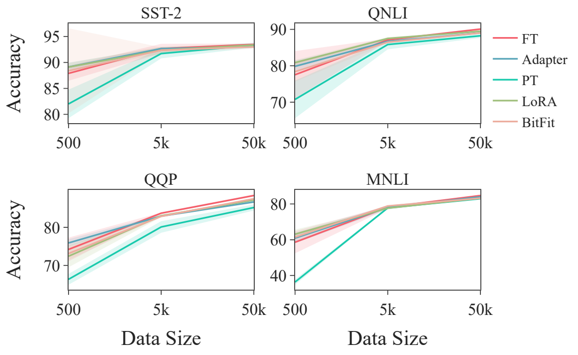

Figure 8: Performance over data scale in 500, 5k, 50k on SST-2, QNLI, QQP, and MNLI. The shaded regions are the standard deviations.

C Additional Experiments and Analyses

C.1 The Same Task with Different Training Data Sizes

To make the above conclusions in §3.2 more convincing, we conduct fine-grained experiments following (He et al., 2021). Specifically, we separately sample 500, 5k and 50k training instances from the original training data as representatives of low-, medium- and high-resource settings, in addition to draw another 1k samples as development set for each task. We report experimental results for WiC, STS-B, BoolQ, SST-2, QNLI, QQP, and MNLI, which have more than 6k training samples, and following the settings illustrated in §3.1.

Confirming our conclusions, in Table 5, we obtain fully consistent findings with §3.2 and Table 3, that prefix tuning consistently falls behind finetuning on various-resources tasks; Adapter&LoRA&BitFit significantly outperforms finetuning on low-resource tasks; Adapter&BitFit keep competitive with finetuning and LoRA lags behind; and all PETuning methods falls behind on high-resource tasks.

In addition, we plot the mean and std. values with different data scales on the same task in Figure 8, to confirm the std. is substantially proportional to training data size.

Model↓, Dataset→	WiC	STS-B	BoolQ	SST-2	QNLI	QQP	MNLI	Avg.
				500				
FT	$56.12_{\pm 2.13}$	$83.81_{\pm 1.34}$	$62.17_{\pm 0.0}$	$87.89_{\pm 1.42}$	$77.54_{\pm 6.5}$	$74.21_{\pm 3.11}$	$58.61_{\pm 6.21}$	$71.47_{\pm 1.44}$
Adapter	$58.36^{\uparrow}_{\pm 3.98}$	$85.74^{\uparrow}_{\pm 0.64}$	$61.56_{\pm 1.51}$	$89.12^{\uparrow}_{\pm 0.84}$	$79.85^{\uparrow}_{\pm 1.55}$	$75.91^{\uparrow}_{\pm 1.01}$	$60.92^{\uparrow}_{\pm 2.86}$	$73.07^{\uparrow}_{\pm 0.84}$
PT	$56.25_{\pm 1.07}$	$73.98^{\downarrow}_{\pm 3.79}$	$57.55^{\downarrow}_{\pm 4.43}$	$82.01^{\downarrow}_{\pm 2.75}$	$70.77^{\downarrow}_{\pm 5.16}$	$66.37^{\downarrow}_{\pm 1.42}$	$36.35^{\downarrow}_{\pm 1.51}$	$63.32^{\downarrow}_{\pm 1.28}$
LoRA	$58.64^{\uparrow}_{\pm 1.16}$	$85.06^{\uparrow}_{\pm 0.83}$	$60.48_{\pm 4.74}$	$89.1^{\uparrow}_{\pm 0.78}$	$80.86^{\uparrow}_{\pm 0.7}$	$72.43_{\pm 2.9}$	$63.1^{\uparrow}_{\pm 2.57}$	$72.81^{\uparrow}_{\pm 0.84}$
BitFit	$57.92^{\uparrow}_{\pm 2.4}$	$84.39^{\uparrow}_{\pm 1.98}$	$62.16_{\pm 0.04}$	$88.38_{\pm 8.19}$	$78.38_{\pm 2.74}$	$73.13_{\pm 1.63}$	$61.47^{\uparrow}_{\pm 2.08}$	$72.26^{\uparrow}_{\pm 1.23}$
				5k				
FT	$66.98_{\pm 1.26}$	$90.76_{\pm 0.03}$	$73.81_{\pm 1.11}$	$92.66_{\pm 0.92}$	$87.12_{\pm 0.37}$	$83.72_{\pm 0.17}$	$78.18_{\pm 0.82}$	$81.89_{\pm 0.44}$
Adapter	$65.15_{\pm 2.85}$	$90.24_{\pm 0.05}$	$73.51_{\pm 1.63}$	$92.66_{\pm 0.25}$	$86.69_{\pm 0.49}$	$82.97_{\pm 0.32}$	$78.0_{\pm 1.17}$	$81.32_{\pm 0.71}$
PT	$66.25_{\pm 1.29}$	$89.14_{\pm 0.66}$	$62.32^{\downarrow}_{\pm 0.14}$	$91.7_{\pm 0.92}$	$85.83^{\downarrow}_{\pm 1.32}$	$80.11^{\downarrow}_{\pm 1.52}$	$77.78_{\pm 0.42}$	$79.02^{\downarrow}_{\pm 0.83}$
LoRA	$62.46^{\downarrow}_{\pm 1.38}$	$90.57_{\pm 0.15}$	$71.9^{\downarrow}_{\pm 0.41}$	$92.35_{\pm 0.7}$	$87.49_{\pm 0.39}$	$83.03_{\pm 0.36}$	$77.67_{\pm 0.26}$	$80.78^{\downarrow}_{\pm 0.45}$
BitFit	$67.61_{\pm 0.49}$	$90.37_{\pm 0.19}$	$75.08^{\uparrow}_{\pm 0.56}$	$92.35_{\pm 0.56}$	$86.47_{\pm 0.28}$	$82.9_{\pm 0.25}$	$78.87_{\pm 0.1}$	$81.95_{\pm 0.15}$
				50k				
FT	-	-	-	$93.46_{\pm 0.18}$	$90.07_{\pm 0.22}$	$88.36_{\pm 0.18}$	$84.71_{\pm 0.41}$	$89.15_{\pm 0.27}$
Adapter	-	-	-	$93.02_{\pm 0.25}$	$89.03^{\downarrow}_{\pm 0.17}$	$86.67^{\downarrow}_{\pm 0.26}$	$84.03_{\pm 0.56}$	$88.19^{\downarrow}_{\pm 0.18}$
PT	-	-	-	$93.32_{\pm 0.47}$	$88.23^{\downarrow}_{\pm 0.59}$	$85.21^{\downarrow}_{\pm 0.79}$	$82.96^{\downarrow}_{\pm 0.20}$	$87.43^{\downarrow}_{\pm 0.41}$
LoRA	-	-	-	$93.35_{\pm 0.27}$	$89.49_{\pm 0.13}$	$87.20^{\downarrow}_{\pm 0.33}$	$83.26^{\downarrow}_{\pm 0.11}$	$88.33^{\downarrow}_{\pm 0.20}$
BitFit	-	-	-	$92.99_{\pm 0.38}$	$89.00^{\downarrow}_{\pm 0.09}$	$87.51^{\downarrow}_{\pm 0.14}$	$83.25^{\downarrow}_{\pm 0.08}$	$88.19^{\downarrow}_{\pm 0.12}$

Table 5: Mean and standard deviation results for the 7 tasks by 500, 5k, and 5k samples of training data sets across 20 runs.

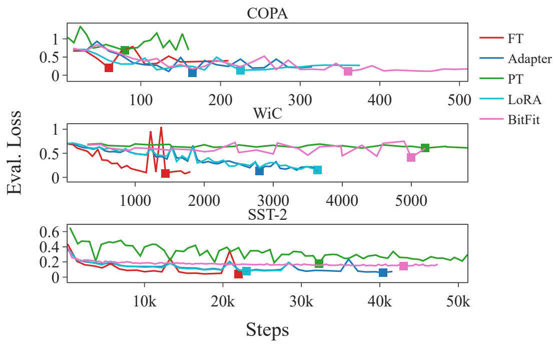

Figure 9: Evaluation loss over training steps on COPA, WiC, and SST-2.

C.2 Finetuning is More Prone to Overfit

To investigate the reasons behind the performance discrepancy of finetuning and PETuning under different training resources, we plot the evaluation loss over training steps for COPA, WiC, and SST-2, as the representatives of low-, medium-, and high-resource tasks, respectively in Figure 9. We observe that finetuning always converges faster than PETuning, especially on low-resource task COPA, where the training steps are less than 100. One possible explanation for the aforementioned discrepancy is that finetuning could converge faster than PETuning methods, which might cause the overfitting issue on low-resource settings, subsequently leading to the poorer performance.

	Small	Medium	Large
Adapter	$68.0_{\pm \mathbf{3.42}}$	$\mathbf{68.45}_{\pm 4.17}$	$65.9_{\pm 5.42}$
PT	$\mathbf{55.35}_{\pm \mathbf{4.71}}$	$\mathbf{55.35}_{\pm 5.07}$	$52.1_{\pm 5.24}$
LoRA	$\mathbf{66.4}_{\pm 9.05}$	$62.4_{\pm \mathbf{8.99}}$	$59.8_{\pm 9.24}$

Table 6: Performance over 20 runs on COPA task, controlled by global random seeds, weight initialization (WI) random seeds, and data order (DO) random seeds, respectively.

C.3 High Stability on Fewer Trainable Parameters.

The probability density curves (Figure 5 and Figure 10) have statistically confirmed PETuning methods tend to exhibit higher stability with fewer trainable parameters. In Table 6, we also directly list the numerical results of Adapter, PT, and LoRA over small, medium, and large parameter scales across 20 runs. While the results substantially support our conclusion that Adapter and PT achieve lowest standard deviations on the small parameter scale, except LoRA obtains slightly lower std. on the medium one. To gain more understanding about the multi-run results, we visualise them in Figure 11. Confirming our conclusion, we can observe that PETuning methods indeed show a trend towards clustering points and smaller boxes on small parameter scale, which means probably higher stability. However, there are also likely to generalise outliers on small parameter scale as shown in Fig-

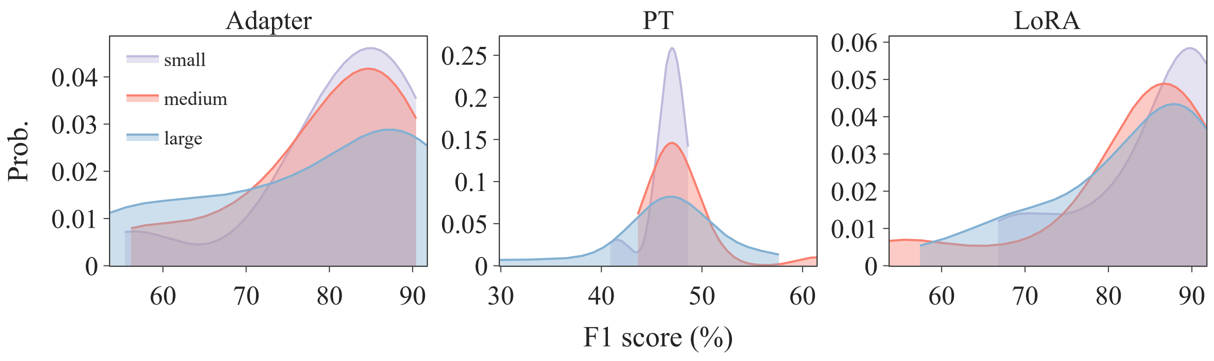

Figure 10: Performance probability density curves of Adapter, prefix tuning (PT), and LoRA over small, medium, and large parameter scales on CB task across 20 runs.

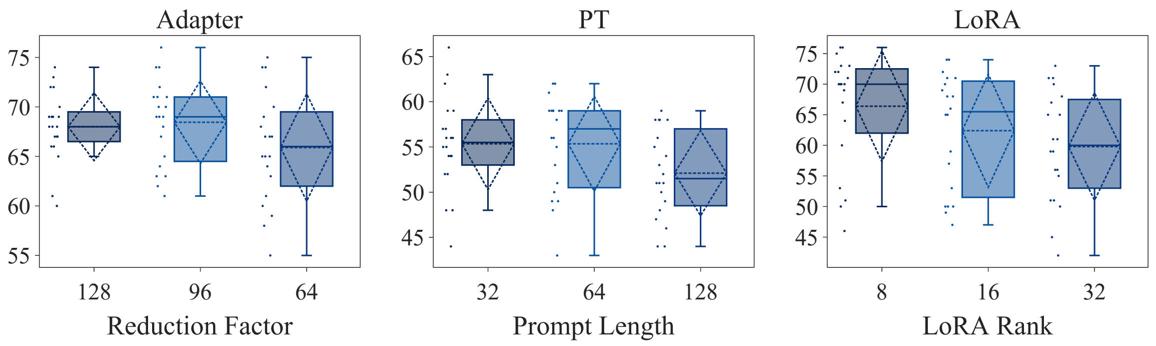

Figure 11: Performance over Adapter, prefix tuning (PT), and LoRA over small, medium, and large parameter scales on COPA task.

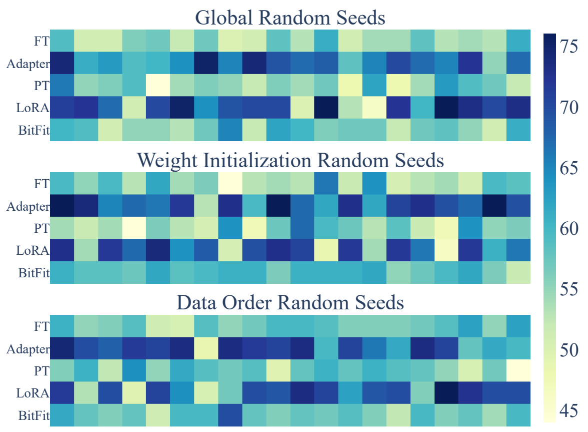

Figure 12: Performance over 20 runs on RTE, controlled by global random seeds, weight initialization (WI) random seeds, and data order (DO) random seeds, respectively.

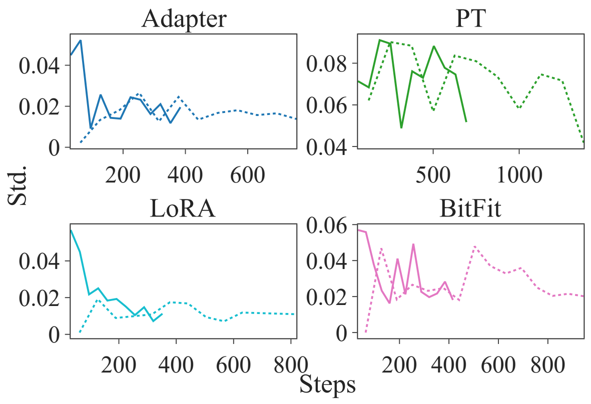

Figure 13: Standard deviations of data size in {1k (solid line), 2k (dashed line)} over training steps on BoolQ task across 20 runs.

ure 11, especially under our limited 20 runs, which could lead to the increasing variance. This special case might result in the inconsistent phenomenon of LoRA with other PETuning methods, nonetheless, the results and phenomena in Table 6 and Figure 11 generally further support the conclusion that PETuning are likely to have high stability on fewer trainable parameters.