X-PEFT: eXtremely Parameter-Efficient Fine-Tuning for Extreme Multi-Profile Scenarios

Anonymous ACL submission

Abstract

Parameter-efficient fine-tuning (PEFT) techniques, such as adapter tuning, aim to fine-tune a pre-trained language model (PLM) using a minimal number of parameters for a specific task or profile. Although adapter tuning provides increased parameter efficiency compared to full-model fine-tuning, it introduces a small set of additional parameters attached to a PLM for each profile. This can become problematic in practical applications with multiple profiles, particularly when a significant increase in the number of profiles linearly boosts the total number of additional parameters. To mitigate this issue, we introduce X-PEFT, a novel PEFT method that leverages a multitude of given adapters by fine-tuning an extremely small set of compact tensors for a new profile, which serve as binary masks to adaptively select the given adapters. To efficiently validate our proposed method, we implement it using a large number of random adapters instead of learned ones. Remarkably, this can be understood as an adapter-based version of the supermask concept, aligning with the principles of the Lottery Ticket Hypothesis. We evaluate the performance of X-PEFT through GLUE tasks and demonstrate that it either matches or surpasses the effectiveness of conventional adapter tuning, despite reducing the memory requirements per profile by a factor of 10,000 compared to it.

1 Introduction

Transfer learning, utilizing various pre-trained language models (PLMs) based on transformers (Devlin et al., 2018; Liu et al., 2020; Lan et al., 2020; Radford et al., 2018), has demonstrated effectiveness across a wide range of NLP tasks. Consequently, it has become common practice to fine-tune a PLM for a specific task rather than training a model from scratch. However, as PLMs scale up, encompassing more than billions of parameters, a fine-tuning approach that updates all model parameters (i.e. full fine-tuning) becomes problematic. For instance, fully fine-tuning a large PLM for a specific task not only requires more computational resources for training but also necessitates additional management of the fine-tuned large PLM. Moreover, when this approach is repetitively applied for multiple tasks, the issues exacerbate. To mitigate these issues, parameter-efficient fine-tuning (PEFT) approaches, such as adapter tuning (Houlsby et al., 2019; He et al., 2022) and prompt tuning (Li and Liang, 2021; Lester et al., 2021), have been proposed as alternatives to fine-tuning the entire model. These approaches are structured similarly, introducing a small number of additional parameters for each task and only fine-tuning those, and Mao et al. (2022) demonstrated that various PEFT approaches can be summarized within a unified view.

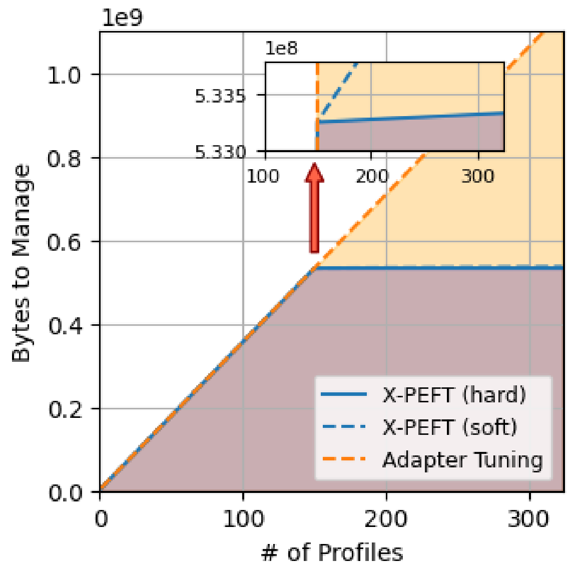

Figure 1: **Demonstrating the remarkable parameter efficiency in terms of memory requirements of X-PEFT in extreme multi-profile scenarios.** Additional details can be found in Section 3.1.

Although PEFT approaches have achieved parameter efficiency with PLMs compared to full-model fine-tuning, practical scenarios, such as multi-profile applications that require management of a large number of profiles, demand even higher parameter efficiency. Consider, for instance, a scenario depicted in the LaMP (Salemi et al., 2023) dataset, where a news organization utilizes an article category classifier to assign articles to their respective categories, such as politics, economics, entertainment, and more. There may be numerous authors or profiles to manage, each with distinct criteria and preferences for categorizing articles, necessitating the management of a large number of sets of author-specific parameters (i.e. adapters). Another example is a personalized chatbot service, where the number of users may continually increase and user-specific parameters must be managed to effectively improve user experiences.

To further improve parameter efficiency in extreme multiple-profile scenarios, we propose X-PEFT (eXtremely Parameter-Efficient Fine-Tuning), a novel PEFT method that leverages a multitude of given adapters by fine-tuning an extremely small set of compact tensors for a new profile, serving as binary masks to adaptively select the given adapters. As depicted in Figure 1, our X-PEFT drastically reduces the memory requirements of additional parameters for each profile, by a factor of 10,000 compared to existing PEFT methods such as adapter tuning. More specifically, after a certain number of profiles have been trained with adapter tuning and their adapter parameters have accumulated (e.g. 150 profiles in Figure 1), each new incoming profile is designed to reuse and adaptively select them, rather than training a new adapter from scratch. Therefore, our proposed method only necessitates learning and storing binary mask tensors at the byte level, which are substantially more compact than adapters.

We conduct extensive experiments to validate our proposed method efficiently, using a large number of random adapters instead of learned ones. Even with the use of random adapters, we demonstrate the proper applicability of X-PEFT. Interestingly, this can be viewed as an adapter-based version of the supermask concept (Zhou et al., 2019), aligning with the principles of the Lottery Ticket Hypothesis (Frankle and Carbin, 2019).

Overall, our contributions can be summarized as follows:

- We introduce a novel PEFT method, X-PEFT, which achieves higher parameter efficiency by a factor of 10,000 compared to adapter tuning without sacrificing performance.
- We experimentally implement X-PEFT using a large number of random adapters, based on the principles of the Lottery Ticket Hypothesis, and validate its effectiveness in extreme multi-profile scenarios.
- We additionally apply X-PEFT in a practical scenario using the LaMP Dataset, employing a large number of learned adapters, and demonstrate the efficiency of X-PEFT in terms of both performance and memory requirements.

2 eXtremely-PEFT

The goal of X-PEFT is to achieve parameter efficiency with PLMs, especially in extreme multi-profile scenarios where an increasing number of profiles continuously necessitates additional memory. For this reason, we propose to simply reuse a large number of given adapter, often in the hundreds, for each new profile rather than training additional new adapters from scratch for it. More specifically, we introduce some learnable mask tensors for each new profile, which are utilized to select and aggregate existing adapters into new ones during inference.

In terms of combining multiple adapters, X-PEFT is similar to AdapterFusion (Pfeiffer et al., 2021), but it combines existing adapters, which are fixed and shared across profiles, with lightweight mask tensors, thereby avoiding additional fusion layers. Moreover, as our primary focus is on extreme multi-profile scenarios, X-PEFT aims to effectively leverage knowledges from a substantial number of existing adapters, as opposed to AdapterFusion, which utilizes a limited few.

In this work, we implement X-PEFT by defining adapters with LoRA (Hu et al., 2022). However, it is worth noting that X-PEFT can accommodate any other type of adapters. As a LoRA adapter consists of submodule A and B corresponds to the down-projection and up-projection feed-forward networks respectively, we define a model for each profile by adding a pair of $A^{(l)}$ and $B^{(l)}$ in each PLM block l. Based on this setting, we assume that N adapters $\{(A_i^{(l)}, B_i^{(l)})\}_{I=1}^{N}$ for each PLM block l are collected in advance by using regular adapter tuning with N profiles. After that, each new incoming profile is designed to adaptively select and reuse them instead of training new ones.

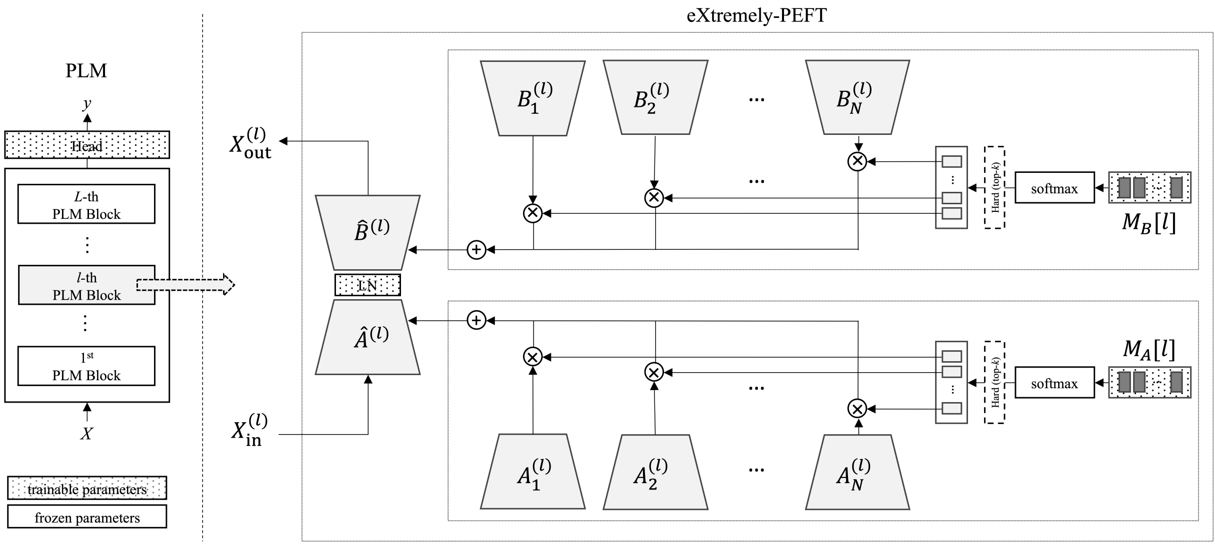

Figure 2: **Illustration of our proposed method, X-PEFT**. Additional details can be found in Section 2.

X-PEFT with soft masks To implement this in a lightweight manner, we introduce two types of mask tensors for each new profile, namely M_A and M_B. The former combines the submodule A's of the adapters, while the latter combines the submodule B's. Each mask tensor is a matrix $\mathbb{R}^{L\times N}$ in which each row corresponds to a mask for each PLM block, assigning different weights to N adapters, subsequently utilized to construct new adapters $\hat{A}^{(l)}$ and $\hat{B}^{(l)}$ for each PLM block l as follows:

$$\hat{A}^{(l)} = \sum_{i=1}^{N} M_A[l,i]A_i^{(l)}, \hat{B}^{(l)} = \sum_{i=1}^{N} M_B[l,i]B_i^{(l)},$$

where both adapters are applied to the input $X_{\text{in}}^{(l)}$ as $X_{\text{out}}^{(l)} = \hat{B}^{(l)}\hat{A}^{(l)}X_{\text{in}}^{(l)}$ [1] to compute the output $X_{\text{out}}^{(l)}$. Here, the weight vectors $M_A[l]$ and $M_B[l]$ can be viewed as soft masks. In this approach, we treat each mask tensor as a regular trainable weight, similar to adapters but more compact, and apply the softmax activation before aggregating adapters to ensure that the weights sum to 1. This method is quite straightforward and does not require any special tricks to learn the mask tensors. Therefore, during fine-tuning for each new profile, we simultaneously and only optimize mask tensors and task header and freeze all other parameters related to PLM and N adapters. In terms of memory requirements, it is more efficient than regular adapter tuning (see Table 1), but can be further improved by using hard masks instead of soft ones.

[1]We insert layer normalization (LN, Ba et al. (2016)) after multiplying $\hat{A}^{(l)}$ that experimentally improved the overall performance

X-PEFT with hard masks By defining the mask tensors with hard masks, our method achieves significant parameter efficiency, particularly in terms of memory requirements, in extreme multi-profile scenarios. Hard masks require only binary masking information during inference. This means that for each new profile, we need to maintain only two bit arrays: one for M_A and the other for M_B. We implement this by ensuring that the weight vectors $M_A[l]$ and $M_B[l]$ are k-hot vectors for all PLM layers, where k is the number of selected adapters within the given N ones. As k-hot vectors are non-differentiable, we employ the straight-through gradient estimation technique (Bengio et al., 2013) for optimizing it through gumbel softmax (Jang et al., 2017; Maddison et al., 2017) with top-k components (see Algorithm 1 in Appendix A). Therefore, the mask tensors with hard masks are similarly optimized and utilized as the soft ones, except they are binarized into k-hot vectors after the training and stored in a more compact way.

Parameter efficiency The number of trainable parameters for each new profile with X-PEFT is calculated as $2(N+b)\times L$. This calculation includes two mask weight vectors and LN affine parameters across all PLM blocks, and it entirely depends on the given number of adapters, denoted by N. Conversely, a conventional adapter tuning method necessitates the complete training of submodules A and B for each profile, which is quantified as

<table>
<tr><th rowspan="2">Mode</th><th colspan="2">Trainable Parameters</th><th colspan="2">Memory Requirements</th></tr>
<tr><th>Formula</th><th>Count</th><th>Formula</th><th>Byte</th></tr>
<tr><td>x_peft (hard)</td><td rowspan="2">$2(N+b)\times L$</td><td rowspan="2">$(N=100)$ 3.5K
$(N=200)$ 5.9K
$(N=400)$ 10.7K</td><td>$2\lceil N/8\rceil \times L$</td><td>$(N=100)$ 0.3K
$(N=200)$ 0.6K
$(N=400)$ 1.2K</td></tr>
<tr><td>x_peft (soft)</td><td>$2N\times L\times 4$</td><td>$(N=100)$ 10K
$(N=200)$ 20K
$(N=400)$ 40K</td></tr>
<tr><td>single_adapter</td><td>$2(d\times b)\times L$</td><td>884.7K</td><td>$2(d\times b)\times L\times 4$</td><td>3.5M</td></tr>
</table>

Table 1: **Trainable parameters and memory requirements per profile.** In this comparison, we use a bottleneck dimension $b = 64$, an adapter layer input dimension $d = 768$, the number of PLM blocks $L = 12$, and the number of given adapters $N = \{100, 200, 400\}$.

$2(d \times b) \times L$. As demonstrated in Table 1, the number of trainable parameters with X-PEFT remains constant regardless of the mask type and can be substantially reduced, by a factor of around 100, even when the number of adapters N is increased up to 400.

More interestingly, if we focus on the memory requirements to store these trainable parameters for each profile, X-PEFT with hard masks can further improve the parameter efficiency by a factor of around 10,000 compared to adapter tuning. This ultimately shows how X-PEFT can be used efficiently in extreme multi-profile scenarios.

Connection to Lottery Ticket Hypothesis The Lottery Ticket Hypothesis (Frankle and Carbin, 2019) asserts that a randomly-initialized dense neural network contains a subnetwork, initialized in a manner that enables it to achieve test accuracy comparable to that of the original network after training, for at most the same number of iterations. Furthermore, Zhou et al. (2019) introduced the concept of a supermask, represented as a weight-level mask, which can deliver better-than-chance test accuracy without any training. Similarly, yet distinctively, X-PEFT involves searching for a supermask within a set of given adapters. This search aims to discover adapter-level masks instead of weight-level ones. In other words, our masking granularity corresponds to an entire adapter, whereas Zhou et al. (2019)'s supermask operates at the level of individual weights.

Based on this interpretation, we validate our method with a large number of random adapters rather than learned ones. This allows us to efficiently simulate X-PEFT in extreme multi-profile scenarios by effortlessly increasing the number of adapters N with random adapters. Even with random adapters, our experimental results indicate that X-PEFT can function effectively without any severe performance degradation (see details in Section 3.1). Moreover, it is noteworthy that, even when employing entirely different sets of random adapters, performance is consistently guaranteed, as shown in Figure 7 in Appendix B.

3 Experiments

We evaluate the effectiveness of our proposed method, X-PEFT, by conducting experiments across a wide variety of settings and comparing it against baselines:

- `x_peft` (`xp`): Our proposed method, X-PEFT, with mask tensors.
- `single_adapter` (`sa`): Standard adapter tuning with a single adapter.
- `head_only` (`ho`): Fine-tuning only the downstream without any adapter.

Experimental settings In all experiments, we employ `bert-base-uncased` (Devlin et al., 2018) as the PLM and LoRA (Hu et al., 2022) with a reduction factor $r = 16$ (bottleneck dimension $b = 48$) for adapters. We set a random seed of 42 for all experiments and conduct a separate experiment to verify reproducibility by varying the random seed (see Figure 7 in Appendix B). The AdamW optimizer is used with a learning rate of 1.0×10^{-05}, which underwent linear decay, and the training duration for all experiments is set to 10 epochs. We use 4 GPUs (GeForce RTX 3090) and exploit data parallelism for all experiments. Moreover, we apply gradient checkpointing (Chen et al., 2016) to improve computational efficiency of `x_peft`. All experimental

Mode	Adapters	`cola` (MCC)	`sst2` (Acc)	`mrpc` (Comb)	`qqp` (Comb)	`stsb` (Comb)	`mnli` (Comb)	`qnli` (Acc)	`rte` (Acc)	`wnli` (Acc)
`x_peft`	100 (soft)	0.40	0.90	0.78	0.79	0.79	0.68	0.82	0.58	0.34
	100 (hard)	0.39	0.87	0.76	0.76	0.74	0.63	0.76	**<u>0.61</u>**	0.32
	200 (soft)	0.44	<u>0.91</u>	0.78	0.80	0.80	0.69	0.83	0.60	<u>0.37</u>
	200 (hard)	0.44	0.89	0.81	0.77	0.76	0.65	0.79	0.58	0.34
	400 (soft)	**<u>0.47</u>**	0.90	0.78	<u>0.81</u>	0.81	<u>0.72</u>	<u>0.83</u>	0.58	0.30
	400 (hard)	0.46	0.89	**<u>0.82</u>**	0.78	**<u>0.81</u>**	0.67	0.81	0.55	0.27
`head_only`	-	0.31	0.85	0.76	0.72	0.46	0.53	0.68	0.59	0.38
`single_adapter`	-	0.43	**0.91**	0.76	**0.85**	0.80	**0.80**	**0.88**	0.60	**0.42**

Table 2: **Evaluation of the GLUE tasks.** In the case of hard masking, we employ $k = 50$. The scores in the table are reported based on the official metrics provided by the GLUE dataset. When multiple official metrics exist for a task (indicated as 'Comb'), we present the combined score (i.e., mean). 'Acc' and 'MCC' denote accuracy and Matthew's Correlation, respectively. The full individual metric data can be found in the Appendix G. Underlined values represent the best among `x_peft` cases, and bold-faced values represent the best among all three modes.

cases are given an equal number of training samples to maintain fairness. We use a consistent batch size across all experiments to ensure that parameters received an equal number of updates. Unless otherwise specified, the majority of experiments employ a batch size of 64 and a token sequence length of 128. For implementation, we used the Hugging Face's `transformers` (Wolf et al., 2020) and AdapterHub's `adapter-transformers` (Pfeiffer et al., 2020) packages.

Datasets To properly simulate multi-profile scenarios, we incorporate 9 tasks from the GLUE benchmark (Wang et al., 2019b) and 4 tasks from the more challenging SuperGLUE benchmark (Wang et al., 2019a). Our choice of SuperGLUE tasks (`cb`, `boolq`, `axb`, and `axg`) aligns with our use of the `bert-base-uncased` model, which supports single-sentence and sentence-pair input formats. We conduct evaluation for GLUE and SuperGLUE on the development (`dev`) sub-dataset using the evaluation metrics officially suggested by the benchmark. For the `axg` task in SuperGLUE, we also utilize the Winogender (Rudinger et al., 2018) test dataset recast by Poliak et al. (2018).

We additionally conduct experiments using the LaMP (Salemi et al., 2023) benchmark. However, since LaMP is originally designed for prompt tuning, modifications were necessary for our purposes. Further details regarding these modifications can be found in the Appendix D. In particular, we utilize the 'Personalized News Categorization' dataset from LaMP.

3.1 Experimental Results

GLUE with random adapters We conduct experiments on 9 tasks using their respective datasets. For the `x_peft` case, we utilize 100, 200, and 400 random adapters, to efficiently validate its effectiveness, each for both soft and hard masking. As baselines, we experiment the `single_adapter` and `head_only` cases. The overall results are presented in Table 2.

Our expectation was that the best evaluation score achieved by any `x_peft` experiment for each task would fall between the evaluation scores of `head_only` and `single_adapter`. `head_only` represents the lower bound, as `x_peft` involves training a downstream head in addition to mask tensors. On the other hand, `single_adapter` represents the upper bound, as our objective was to demonstrate that `x_peft` could achieve comparable or superior performance to `single_adapter` with significantly fewer trainable parameters.

As expected, for all tasks except `wnli`, `x_peft`'s best evaluation scores exceed those of `head_only`. Unexpectedly, in about half of the tasks, `x_peft` even outperforms `single_adapter` in terms of evaluation score. For tasks where `x_peft` falls between `head_only` and `single_adapter`, it is much closer to `single_adapter` in performance with a negligible gap. It is noteworthy given the significantly lower number of trainable parameters in `x_peft` in comparison to `single_adapter`.

SuperGLUE with random adapters We conduct experiments on tasks `cb` and `boolq` using their respective datasets. Additionally, we perform experiments on diagnostic tasks `axb` and `axg` using

M.	Adt.	cb (Acc)	boolq (Acc)	axb (MCC)	axg (Acc)	axg (GPS)
xp	100 (s)	.64	.67	.11	<u>**.53**</u>	92.7
	100 (h)	.68	.66	.09	.48	86.6
	200 (s)	.68	.66	.07	.52	<u>**96.1**</u>
	200 (h)	.68	.66	.02	.50	88.4
	400 (s)	.68	.66	.09	.51	93.5
	400 (h)	<u>.70</u>	<u>**.68**</u>	<u>**.12**</u>	.50	94.8
ho	-	**.71**	.64	.09	.50	82.3
sa	-	.68	.65	.10	.51	93.5

Table 3: **Evaluation of the SuperGLUE tasks.** For hard masking, we employ $k = 50$. 'GPS' denotes Gender Parity Score, and all other symbols and text decorations can be understood in the same context as in Table 2.

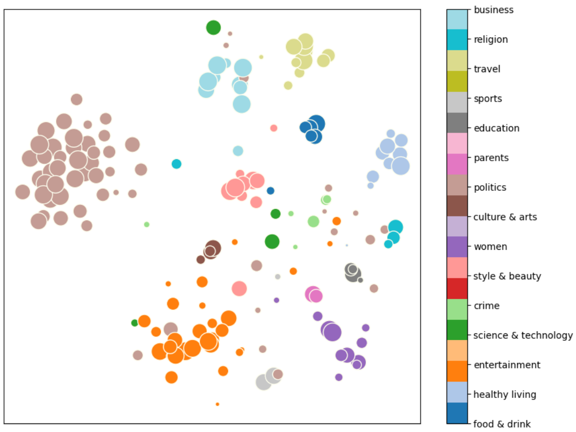

Figure 3: **Visualization of mask tensors with t-SNE.** Each point represents an author/profile, and the color and size of it represent the majority category assigned by each author and the majority ratio in an article. This shows how the mask tensors effectively capture the categorization diversity among authors.

GLUE's `rte` dataset for training. For the `x_peft` case, we apply the same setting of using random adapters as the evaluation of GLUE tasks. The overall results are presented in Table 3.

Similarly to the evaluation results of the GLUE experiments, in all cases, `x_peft`'s highest evaluation scores match or even surpass those of `single_adapter`. Unexpectedly, for `cb`, `head_only` performs the best. This performance from `x_peft` is noteworthy considering the significantly lower number of trainable parameters.

LaMP with learned adapters Our modified LaMP dataset follows the schema (`news_text`, `news_category`, `author_profile`), structuring each data point for text classification while incorporating the author's identity. It contains 17,005 news texts authored by 323 individuals / profiles. On average, each author contributed 52.65 news texts, with a standard deviation of 87.28 (ranging from a minimum of 6 to a maximum of 640).

In our experiment for `x_peft`, we first randomly select 150 authors and train adapters for each author with `single_adapter(sa)` to get $N = 150$ learned (not random) adapters (denoted by `x_peft with sa`). Subsequently, we conduct individual (per-profile) training by optimizing mask tensors with those 150 learned adapters. As a result, we obtain a collection of mask tensors for 173 new authors. These mask tensors serve as a highly efficient means of personalizing the model for multi-profile scenarios. The memory requirements for this setting are precisely shown in Figure 1. Essentially, these mask tensors encapsulate a unique signature of each author, specifically revealing how they categorize news texts in this context. We visualize the 173 sets of mask tensors using t-SNE (van der Maaten and Hinton, 2008) in Figure 3, along with heatmaps illustrating the mask tensors of the two most distant (Euclidean) profiles as shown in Figure 6. The averaged evaluation accuracy and F1 scores for all 323 authors can be found in Figure 4, where `x_peft with sa` with hard masks shows improvement compared to `single_adapter`.

Moreover, we conduct additional experiments for `x_peft`. We assume that the first 150 authors can be trained jointly, rather than independently, as part of the warm-start procedure. Therefore, a multi-task learning framework is used to train separate adapters for each author but a shared header. In this approach, `x_peft` only has to train mask tensors and reuse the shared header without any fine-tuning. This setting (`x_peft with mtl`) can not only further improve the parameter efficiency than the previous setting (`x_peft with sa`), but also significantly improve the averaged evaluation accuracy and F1 scores as depicted in Figure 4.

3.2 Ablation Studies and Analysis

The number of given adapters (N) When analyzing training curves, X-PEFT with a higher number of adapters outperforms its counterparts. As shown in Figure 5 (a), the training curves for the `sst2` task consistently position lower when more adapters are used. This trend is consistent across various tasks in the GLUE benchmark.

In terms of evaluation scores, utilizing a higher

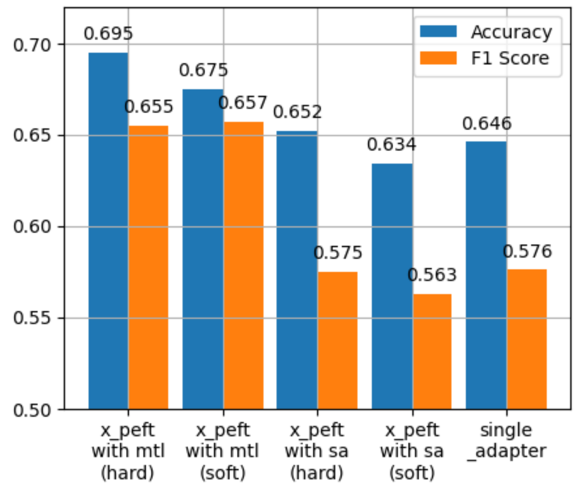

Figure 4: **Evaluation of the Modified LaMP 'Personalized News Categorization' Dataset.** Averaged evaluation accuracy and F1 score over 323 authors are presented (on 30% holdout sets).

number of adapters generally corresponds to better evaluation performance,even though they are random adapters, as demonstrated in Table 2. However, there are some exceptions, notably in tasks such as `rte`, `sst2`, and `wnli`, where an abundance of adapters can potentially lead to overfitting.

Soft masks vs. hard masks We introduce X-PEFT with mask tensors, which can be implemented by either soft or hard masks. Each type has its own advantages and disadvantages. To validate these observations, we compare the two settings across our experiments. In most experiments, X-PEFT with hard masks demonstrated superior generalization performance compared to the soft ones (refer to Table 2 and Figure 4). However, as depicted in Figure 5 (a), soft masks consistently display a lower training loss than their hard ones. From these results, we infer that soft masks are more prone to overfitting, whereas hard masks enhance generalization capabilities.

Separate mask tensors for submodules How can a small number of additional trainable parameters (e.g., mask tensor) in X-PEFT achieve performance that matches adapter tuning? The key factor is the use of two mask tensors instead of just one. When we use only one mask tensor for aggregating adapters (i.e., including only M_B and discarding M_A from the bottleneck), the expressive capacity for a new adapter is limited to N. However, when we use M_A and M_B together in sequence(i.e., separate mask tensors for submodules A and B) can express N^2 cases. We conducted experiments to investigate this aspect on the `sst2` task as shown in Figure 5 (b). Through these experiments, the results confirm that the combination of these two tensors can improve the performance of X-PEFT.

Top-k selection for hard masks For X-PEFT with hard masks, $k = 50$ typically yields favorable training performances. For $N = 200$ and $N = 400$, increasing k improves training performance until reaching $k = 50$, after which it begins to decline. For $N = 100$, a training performance peak is observed at $k = 30$, deviating from the pattern observed with larger N values. Refer to Figure 5 (c) for further insights. As k diverges further from the value of $k = 50$, the loss curves progressively deviate from the curve of $k = 50$. In general, regardless of the value of N, $k = 50$ proves to be a highly reasonable choice.

4 Related Works

AdapterFusion (Pfeiffer et al., 2021) is similar to X-PEFT in using multiple adapters, but it relies on attention tensors (referred to as fusion layers) for combining adapters, making it computationally heavy. This characteristic limits the scalability of AdapterFusion in multi-profile scenarios.

Parallel adapters (Rücklé et al., 2021) and its scaled variant (He et al., 2022) are essentially implementations of AdapterFusion, designed to enable parallel computation throughout adapters. Consequently, they inherit the limitations associated with AdapterFusion. AdapterDrop (Rücklé et al., 2021) is built upon the AdapterFusion architecture but focuses on pruning less significant adapters based on their activation strength. However, the number of adapters to be pruned varies from task to task, and it still employs attention tensors, making it computationally intensive.

AdapterSoup (Chronopoulou et al., 2023) trains a set of adapters and selects a subset of them for inference. As this subset selection occurs at test time, the selection changes with different inputs, requiring the retention of all trained adapters at test time, which is not scalable.

AdaMix (Wang et al., 2022) employs the Mixture-of-Experts concept in adapter tuning. It trains a route policy among layers of adaptation modules, which comprise a mixture of adapters. It combines the weights of adaptation modules selected by an input batch for test time efficiency but requires the retention of all the trained adapters for

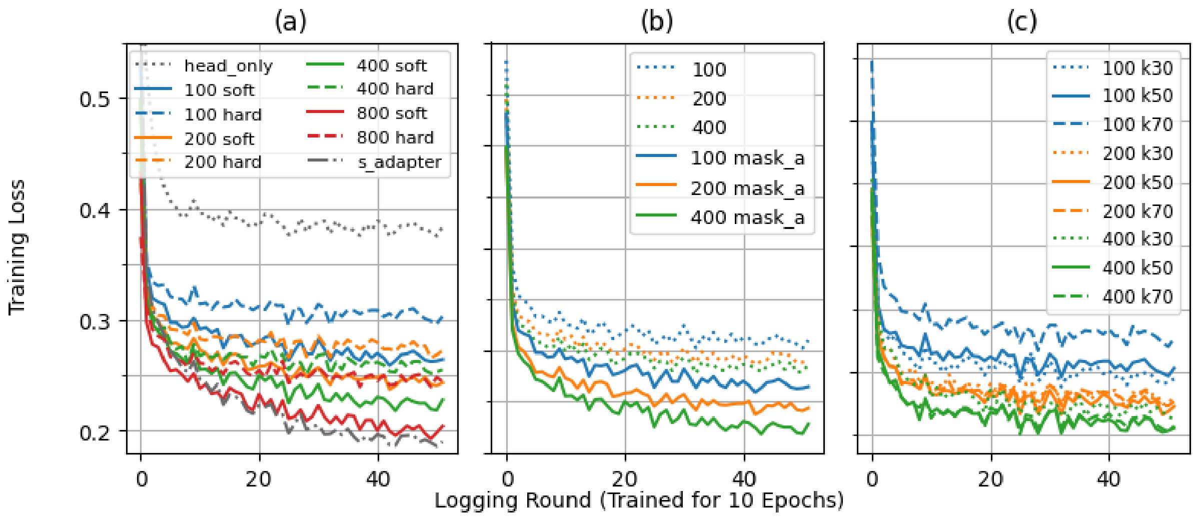

Figure 5: **Training curves for `sst2` with various settings.** (a) Varying the number of adapters and comparing soft / hard masks: more adapters lead to improved loss, and soft masks generally show lower loss than hard ones. (b) Effectiveness of separate mask tensors: the impact of having M_A and M_B is evident. (c) Varying k for hard masks: $k = 50$ consistently shows best performance irrespective of the specific value of N.

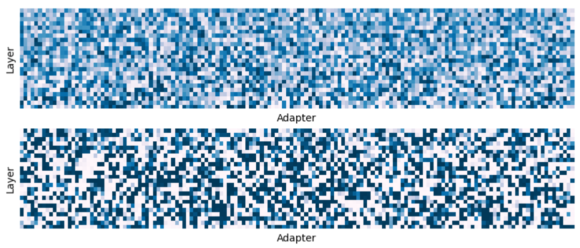

Figure 6: **Heatmaps for mask tensors of most distant authors.** These distinct heatmaps capture the unique characteristics of news categorization.

inference, making it less scalable.

Wu et al. (2022) employ the original approach proposed by the Lottery Ticket Hypothesis (Frankle and Carbin, 2019), applied at the adapter level within the AdapterFusion configuration. They iteratively prune a portion of the adapters until winning tickets are discovered. In contrast, X-PEFT can be viewed as the process of identifying supermasks (Zhou et al., 2019) among parallelly arranged adapter submodules.

Including the aforementioned works, as far as we know, X-PEFT is the first to involve hundreds of adapters (up to 800), except for Wu et al. (2022), which uses a maximum of 192 adapters (while others use fewer than 100 adapters). Additionally, as far as we know, X-PEFT is the first to apply the supermask (Zhou et al., 2019) concept in PEFT, particularly for multi-profile scenarios.

5 Conclusion

In this paper, we introduce eXtremely-PEFT, X-PEFT, a groundbreaking approach to Parameter-Efficient Fine-Tuning for pre-trained language models (PLMs). Our work achieves an unprecedented 1/100 reduction in parameters compared to adapter tuning while maintaining task performance. We also optimize the memory requirements, minimizing them to the byte level by a factor of 10,000, which is crucial for extreme multi-profile scenarios.

Furthermore, we delve deeper into PEFT, significantly reducing trainable parameters, thus reducing resource and computational costs. By incorporating the principles of the Lottery Ticket Hypothesis into adapter-level PEFT, X-PEFT opens new possibilities for resource-efficient natural language processing with PLMs.

Our work not only advances PEFT but also sets the stage for future research in NLP, inspiring novel applications and resource-efficient natural language processing breakthroughs. In conclusion, X-PEFT is a transformative development in PEFT, offering remarkable parameter efficiency without performance compromise.

Limitations

Due to the extensive number of adapters involved in X-PEFT, training can be time-consuming. For hard masking, it is possible to reduce training time by disabling gradients for out-of-top-k adapter submodules. We can also explore concepts from Parallel adapters (Rücklé et al., 2021) about parallel computation for AdapterFusion (Pfeiffer et al., 2021).

There are almost no datasets available for multi-profile benchmarking. LaMP (Salemi et al., 2023) is currently the only dataset that exists for such purposes, but it is primarily designed for prompt tuning. While we did conduct multi-profile experiments on LaMP, these experiments necessitated some modifications. Unfortunately, the scarcity of multi-profile benchmark datasets limited our ability to carry out more comprehensive multi-profile experiments.

Regarding language, our research is constrained to English texts. The PLM utilized in our study has been specifically trained in English, and the datasets we employed are also in English. Future work will need to explore an extended approach to enhance parameter efficiency and multi-profile scalability, especially for low-resource languages.

Ethics Statement

Gender bias in NLP models is a serious problem that can have far-reaching consequences, potentially undermining social integration and peace. Researchers in the NLP field have a responsibility to consider and address potential gender bias in all their efforts. The SuperGLUE benchmark includes a diagnostic dataset focusing on gender bias, namely `axg`. This is why we have included SuperGLUE in our experimental dataset.

X-PEFT offers the advantage of enabling multi-profile service providers to operate with minimal memory or storage requirements, ultimately reducing the strain on data centers and contributing to a reduction in carbon dioxide emissions. However, it's worth noting that the extended training times involving multiple GPUs can be environmentally problematic. Therefore, our ongoing research efforts are dedicated to achieving more efficient training methods and conserving computational power from an ecological viewpoint.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. In *arXiv:1607.06450 [stat.ML]*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. In *arXiv:1308.3432 [cs.LG]*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. In *arXiv:1604.06174 [cs.LG]*.

Alexandra Chronopoulou, Matthew Peters, Alexander Fraser, and Jesse Dodge. 2023. AdapterSoup: Weight averaging to improve generalization of pretrained language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2054–2063, Dubrovnik, Croatia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR 2019*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *ICLR 2022*.

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, PMLR 97:2790-2799, 2019.*

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 *ICLR 2022*.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *ICLR 2020*.

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, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Roberta: A robustly optimized bert pretraining approach. In *ICLR 2020*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

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.

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: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

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*.

Adam Poliak, Aparajita Haldar, Rachel Rudinger, J. Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. 2018. Collecting diverse natural language inference problems for sentence representation evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. 2018. Gender bias in coreference resolution. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. Adapterdrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2023. LaMP: When large language models meet personalization.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. In *Journal of Machine Learning Research 9 (2008)*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint 1905.00537*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

Jiarun Wu, Qingliang Chen, Zeguan Xiao, Yuliang Gu, and Mengsi Sun. 2022. Pruning adatperfusion with lottery ticket hypothesis. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1632–1646, Seattle, United States. Association for Computational Linguistics.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*.

A Algorithms

We suggest the hard softmax algorithm employing straight-through gradient estimation, as outlined in Algorithm 1.

Algorithm 1 Hard (top-k) Softmax (Straight-Through Gradient Estimation)

Require: Input `logits`, noise level `nu`, temperature `tau` and `k` for top-k selection

Ensure: Vector `y` representing the top-k elements

```
logits = logits + nu * Gumbel(0, 1)
y_soft = softmax(logits / tau)
indices = topk(y_soft)
y_hard = khot_encoding(indices)
y_hard = y_hard / k
y = y_hard - y_soft.detach() + y_soft
```

B Figures

Figure 7 illustrates our experiments with different random seeds, revealing consistent trends in the results. It also includes two runs with the same random seed, demonstrating the reproducibility of our experiments based on the random seed.

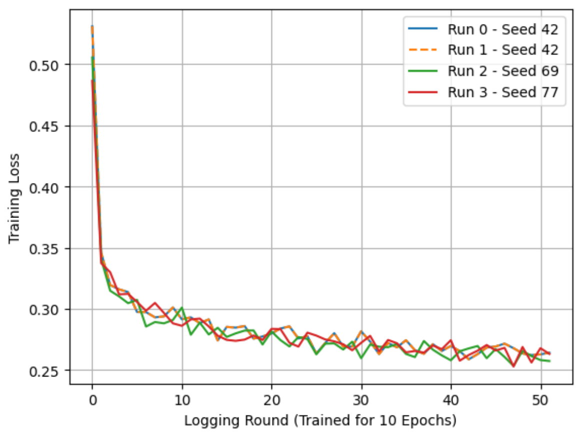

Figure 7: **Training Loss Curves for `sst2` ($N = 100$, soft) with Varying Random Seeds.** While local fluctuations in the loss curves differ from each other, they tend to follow a similar trajectory globally. It's evident that our experiments guarantee reproducibility, as two different runs with random seed 42 yield identical loss curves. (The blue solid line and the orange dashed line are completely overlapped.) In terms of evaluation scores, run 0 and 1 both recorded 0.8956, run 3 recorded 0.8865, and run 4 recorded 0.8968, with little variation among them.

C Hyper-Parameters

The major hyper-parameters are as follows:

- N (The number of adapters attached to an X-PEFT model): Generally, the more adapters, the better the performance. However, considering training budgets, 100, 150, or 200 adapters are quite good choices. Our search space for N was $\{100, 200, 400, 800\}$.
- k (k for top-k selection for hard masking): We found that $k = 50$ is a reasonably good choice regardless of other settings, as discussed in the ablation studies. Our search space for k was $\{1, 10, 20, 30, 40, 50, 60, 70, 80, 100\}$.
- b (Bottleneck dimension of adapters used in an X-PEFT model): The bottleneck dimension of adapters used in an X-PEFT model has no significant impact on the model's performance. We used a default value (48) provided by AdapterHub. Our search space for b was $\{12, 24, 48, 96\}$.
- Batch size: The batch size has no significant impact on the model's performance. A batch size of 64 is suitable for our technical environment. Our search space for batch size was $\{8, 16, 32, 64, 128\}$.

We used `bert-base-uncased`, so the following hyperparameters were consistent across all experiments:

- L (The number of blocks of the PLM)
- d (Input dimension into adapter layers)

D Modification Details for the LaMP dataset

In our research, we utilized the LaMP-2 dataset, specifically the 'Personalized News Categorization' dataset, which is part of the LaMP benchmark (Salemi et al., 2023). However, several modifications were necessary to adapt this dataset for our specific purposes.

The original LaMP-2 dataset was primarily designed for prompt tuning, aiming to understand how specific authors categorize given news articles. Each data point in this dataset consisted of the news article text and the author's profile. It's essential to clarify that the author's profile is not an identifier but rather a collection of news article texts authored by that particular individual, along with the categories assigned by the author to these articles.

As our experiments focused on standard supervised classification, we needed datasets containing pairs of news texts and their corresponding labels, alongside the author's identity. In other words, our data schema needed to be in the format of (`news_text`, `news_category`, `author_id`). Here, `author_id` simply refers to a numerical identifier that can also be used as a label.

To meet these requirements, we exclusively extracted the author profile data from the original LaMP-2 dataset and proceeded to modify it according to the specified format. Given that the same author's data may appear more than once in the original LaMP-2 dataset, we took care to remove any duplicates in our modified version.

Out of the 8,090 data points in the LaMP-2 dataset, we extracted 17,005 news texts, each categorized into one of 15 categories, authored by 323 unique authors, eliminating any duplicates.

E Training Time

Information regarding the training time for our GLUE and SuperGLUE experiments can be found in Table 8 and Table 9.

Here are the training times for our LaMP experiments:

- `x_peft with mtl` (hard): 5.06 hours
- `x_peft with mtl` (soft): 4.90 hours
- `x_peft with sa` (hard): 8.36 hours
- `x_peft with sa` (soft): 8.40 hours

F Trained Parameters

The parameter count of `bert-base-uncased` is known to be 110M. All the X-PEFT configurations that we used in the experiments and their parameter counts including `bert-base-uncased` is as follows (with c representing the label count for a downstream head):

- $N = 100$ and $c = 2, 3, 15$: 200M
- $N = 150$ and $c = 2, 3, 15$: 245M
- $N = 200$ and $c = 2, 3, 15$: 290M
- $N = 400$ and $c = 2, 3, 15$: 468M
- $N = 800$ and $c = 2, 3, 15$: 826M

The counts of trained parameters, both including and excluding the downstream head, are provided in Table 4.

G Detailed GLUE and SuperGLUE Evaluations

Here are the complete evaluations for the GLUE and SuperGLUE benchmarks. Refer to Table 5 and Table 6.

	Including Header			Excluding Header
N	$c = 2$	$c = 3$	$c = 15$	
100	0.596M	0.596M	0.606M	0.004M
150	0.597M	0.598M	0.607M	0.005M
200	0.598M	0.599M	0.608M	0.006M
400	0.603M	0.604M	0.613M	0.011M
800	0.612M	0.613M	0.622M	0.020M

Table 4: **Trained Parameter Count Including and Excluding Header.** c denotes the label count for a downstream head.

Mode	Adapters	`cola` (MCC)	`sst2` (Acc)	`mrpc` (Acc)	`mrpc` (F1)	`qqp` (Acc)	`qqp` (F1)
`x_peft`	100 (soft)	0.3977	0.8956	0.7353	0.8291	0.8132	0.7643
	100 (hard)	0.3891	0.8716	0.7132	0.8146	0.7824	0.7307
	200 (soft)	0.4422	0.9106	0.7328	0.8278	0.8266	0.7793
	200 (hard)	0.4446	0.8911	0.7745	0.8521	0.7933	0.7480
	400 (soft)	0.4654	0.8991	0.7328	0.8250	0.8345	0.7845
	400 (hard)	0.4592	0.8899	0.7843	0.8562	0.8011	0.7515
`head_only`	-	0.3122	0.8521	0.7059	0.8187	0.7575	0.6884
`single_adapter`	-	0.4277	0.9140	0.7034	0.8130	0.8688	0.8263

Table 5: **Evaluation of the GLUE tasks (part 1).** In the case of hard masking, we employ $k = 50$ for top-k selection. The scores in the table are reported based on the official metrics provided by the GLUE dataset. 'Acc,' 'MCC,' and 'F1' denote accuracy, Matthew's Correlation, and F1 score, respectively.

Mode	Adapters	`stsb` (PCC)	`stsb` (SRC)	`mnli` (Acc)	`mnli` (AMM)	`qnli` (Acc)	`rte` (Acc)	`wnli` (Acc)
`x_peft`	100 (soft)	0.7888	0.7948	0.6663	0.6894	0.8182	0.5776	0.3380
	100 (hard)	0.7404	0.7492	0.6186	0.6372	0.7626	0.6101	0.3239
	200 (soft)	0.8001	0.8076	0.6863	0.7013	0.8343	0.5957	0.3662
	200 (hard)	0.7506	0.7646	0.6320	0.6597	0.7891	0.5776	0.3380
	400 (soft)	0.8028	0.8089	0.7074	0.7275	0.8349	0.5848	0.2958
	400 (hard)	0.8115	0.8148	0.6569	0.6789	0.8083	0.5487	0.2676
`head_only`	-	0.4687	0.4482	0.5307	0.5335	0.6842	0.5884	0.3803
`single_adapter`	-	0.7995	0.8057	0.7934	0.8034	0.8812	0.5993	0.4225

Table 6: **Evaluation of the GLUE tasks (part 2).** In the case of hard masking, we employ $k = 50$ for top-k selection. The scores in the table are reported based on the official metrics provided by the GLUE dataset. 'Acc,' 'PCC,' and 'SRC' denote accuracy, Pearson correlation, and Spearman correlation, respectively. For `mnli`, 'Acc' and 'AMM' denote accuracy matched and accuracy mismatched, respectively.

Mode	Adapters	cb (Acc)	boolq (Acc)	axb (MCC)	axg (Acc)	axg (GPS)
x_peft	100 (soft)	0.6429	0.6676	0.1111	0.5253	92.6724
	100 (hard)	0.6786	0.6569	0.0943	0.4831	86.6379
	200 (soft)	0.6786	0.6599	0.0721	0.5197	96.1207
	200 (hard)	0.6786	0.6648	0.0244	0.5028	88.3621
	400 (soft)	0.6786	0.6599	0.0916	0.5084	93.5345
	400 (hard)	0.6964	0.6792	0.1203	0.5000	94.8276
head_only	-	0.7143	0.6358	0.0869	0.4972	82.3276
single_adapter	-	0.6786	0.6489	0.1027	0.5084	93.5345

Table 7: **Evaluation of the SuperGLUE tasks.** In the case of hard masking, we employ $k = 50$ for top-k selection. 'GPS' denotes Gender Parity Score, and all other symbols can be understood in the same context as in Table 5 and 6

Mode	Adapters	cola	sst2	mrpc	qqp	stsb	mnli	qnli	rte	wnli
x_peft	100 (soft)	0.55	4.32	0.25	26.07	0.38	24.20	6.71	0.17	0.05
	100 (hard)	0.57	6.12	0.26	26.69	0.38	24.32	7.02	0.17	0.05
	200 (soft)	1.11	8.10	0.48	43.67	0.71	47.12	12.61	0.32	0.10
	200 (hard)	1.11	8.51	0.50	44.13	0.71	47.33	12.54	0.33	0.09
	400 (soft)	2.16	16.93	0.92	90.43	1.40	104.57	29.29	0.62	0.19
	400 (hard)	2.07	16.91	0.91	91.45	1.41	108.14	26.15	0.78	0.19
head_only	-	0.04	0.47	0.02	1.08	0.04	2.70	0.46	0.01	0.00
single_adapter	-	0.09	0.55	0.03	2.97	0.05	4.22	1.61	0.01	0.01

Table 8: **Computation Cost of the GLUE tasks (Training Time, Hours).**

Mode	Adapters	cb	boolq	axb	axg
x_peft	100 (soft)	0.02	0.60	0.18	0.18
	100 (hard)	0.02	0.61	0.18	0.18
	200 (soft)	0.03	1.17	0.33	0.33
	200 (hard)	0.03	1.19	0.35	0.35
	400 (soft)	0.06	2.29	0.64	0.64
	400 (hard)	0.06	2.41	0.68	0.68
head_only	-	0.00	0.07	0.02	0.02
single_adapter	-	0.00	0.08	0.03	0.03

Table 9: **Computation Cost of the SuperGLUE tasks (Training Time, Hours).**