One Network, Many Masks:
Towards More Parameter-Efficient Transfer Learning

Guangtao Zeng[*]   **Peiyuan Zhang**[*]   **Wei Lu**
StatNLP Research Group
Singapore University of Technology and Design
guangtao_zeng@mymail.sutd.edu.sg, {peiyuan_zhang, luwei}@sutd.edu.sg

Abstract

Fine-tuning pre-trained language models for multiple tasks tends to be expensive in terms of storage. To mitigate this, parameter-efficient transfer learning (PETL) methods have been proposed to address this issue, but they still require a significant number of parameters and storage when being applied to broader ranges of tasks. To achieve even greater storage reduction, we propose PROPETL, a novel method that enables efficient sharing of a single PETL module which we call prototype network (e.g., adapter, LoRA, and prefix-tuning) across layers and tasks. We then learn binary masks to select different sub-networks from the shared prototype network and apply them as PETL modules into different layers. We find that the binary masks can determine crucial information from the network, which is often ignored in previous studies. Our work can also be seen as a type of pruning method, where we find that overparameterization also exists in the seemingly small PETL modules. We evaluate PROPETL on various downstream tasks and show that it can outperform other PETL methods with approximately 10% of the parameter storage required by the latter.[1]

1 Introduction

With the release and wide application of numerous pre-trained language models (PLMs) (Devlin et al., 2019; Liu et al., 2019), pre-training and subsequently fine-tuning them becomes prevalent in natural language processing (NLP). This yields good performance in many downstream tasks. However, such a paradigm requires the entire model to be updated and saved after fine-tuning. As PLMs grow in size, traditional fine-tuning becomes costly in storage, limiting the application to multi-task scenarios. In order to ameliorate this issue, many Parameter-Efficient Transfer Learning (PETL) methods have
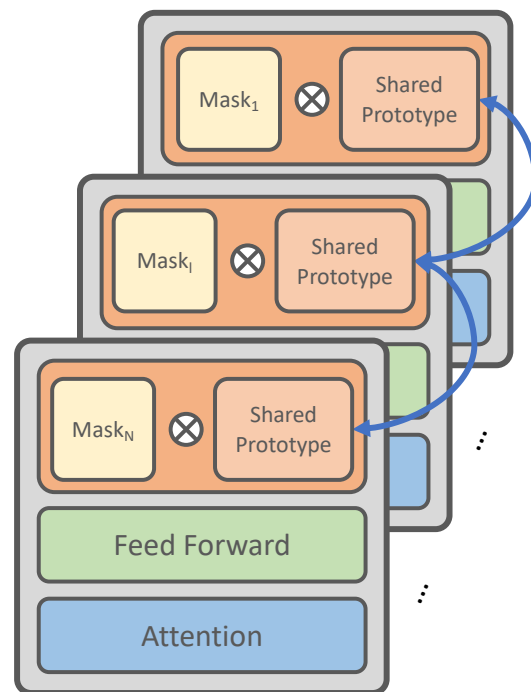

Figure 1: An illustration of our PROPETL_Adapter model. Note that PROPETL is orthogonal to the specific PETL architectures. LoRA and prefix-tuning are also implemented in our framework.

been proposed (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2022). Rather than fine-tuning the entire model, they introduce new parameters and only fine-tune those additional parameters on downstream tasks, which drastically reduces the storage of the parameters required by each task. However, they still require a significant number of parameters when more tasks are considered.

In this paper, we continue this line of research and target using even less storage per task. We observe that recent advancements in the field of PETL focus on finding better ways to apply the additional parameters, such as the adapter module after each feed-forward layer (Houlsby et al., 2019; Pfeiffer et al., 2021), or the low-rank matrices in the query and value projection of the self-attention

---

[*]The first two authors contributed equally.

[1]Our code is available at https://github.com/ChaosCodes/ProPETL.

networks (Hu et al., 2022). However, limited works examine the impact of sub-network structure and integrate pruning methods with PETL methods. In fact, studies in network pruning have shown that the modeling ability of neural networks relies not only on the parameters but also on the sub-network structures that are decided by the pruning masks. For instance, Zhou et al. (2019) discovered that the sub-network of an untrained model can yield great performance without any parameter update. In light of this, we seek to incorporate the structural information of sub-networks into PETL. We believe that when enough structural information is injected into the network, we no longer need that many parameters in PELT modules and further improve the parameter efficiency.

To this end, we propose a novel PETL method dubbed **PROPETL** that enables efficient sharing of a single *prototype* adapter, prefix, or LoRA across layers and tasks. When sharing the prototype network, PROPETL learns binary masks to prune different sub-networks in different layers and tasks (Figure 1). The connections of the prototype network pruned in one layer can be used in another with a different pruning mask. In this way, a parameter can be used multiple times across different modules, achieving higher parameter efficiency. Previous methods (He et al., 2022b; Ma et al., 2022) only consider simply discarding (pruning) the useless parameters, while we focus on the structural information in masks by strategically dispatching the parameters in the single prototype network to different modules. We evaluate PROPETL on various downstream tasks, including GLUE (Wang et al., 2018), XSum (Narayan et al., 2018), and WMT16 Ro-En (Bojar et al., 2016). Experiments show that PROPETL achieves better performance than other PETL methods while using significantly fewer parameters.

Our contributions are summarized as follows:

- We propose PROPETL, a highly parameter-efficient transfer learning method that injects structural information into PELT and allows for efficient sharing of a single prototype network across layers and tasks.

- Experiments show PROPETL is able to dramatically reduce the storage of the parameters while achieving better performance than conventional PETL methods.

- PROPETL offers an alternative view for network pruning and sharing, where we use binary masks to decide when to discard or share the parameters. We hope to inspire more intriguing explorations in this direction.

2 Related Work

In this section, we briefly survey ideas that are related to our work from three fields: parameter-efficient transfer learning, pruning methods, and multi-task learning.

2.1 Parameter-Efficient Transfer Learning

Recently, as the pre-trained language models get larger and larger, some parameter-efficient transfer learning methods that only update a few extra parameters while freezing the PLM backbone have been proposed. Adapter-tuning (Houlsby et al., 2019) fine-tuned adapter modules inserted after each attention and feed-forward layer. Prefix-tuning (Li and Liang, 2021) placed an additional trainable prefix to the keys and values matrix in the attention module. LoRA (Hu et al., 2022) injected tunable rank decomposition matrices into each Transformer layer. Based on these parameter-efficient transfer learning methods, He et al. (2022a) gave a unified framework that allows for the transfer of design elements across various PETL approaches. However, when applied to larger PLMs and a broader range of tasks, these methods still require a large storage space because the number of extra parameters is directly proportional to the number of layers and tasks. Inspired by the parameter-sharing techniques of AL-BERT (Lan et al., 2020a), we propose sharing the additional parameters in PETL modules across layers and tasks. Our method can thus obtain higher parameter-efficient efficiency with a significantly smaller portion of additional storage than existing PETL methods.

2.2 Pruning Methods

Pruning is one of the most popular methods to reduce unnecessary weights from over-parameterized neural networks while maintaining comparable performance. Recently, Frankle and Carbin (2019) proposed Lottery Ticket Hypothesis and stated that in a randomly initialized dense model, a sparse sub-network exists that, when trained in isolation, can achieve performance comparable to dense models. Accompanied by this hypothesis, many pruning-before-training methods have emerged (Lee et al.,
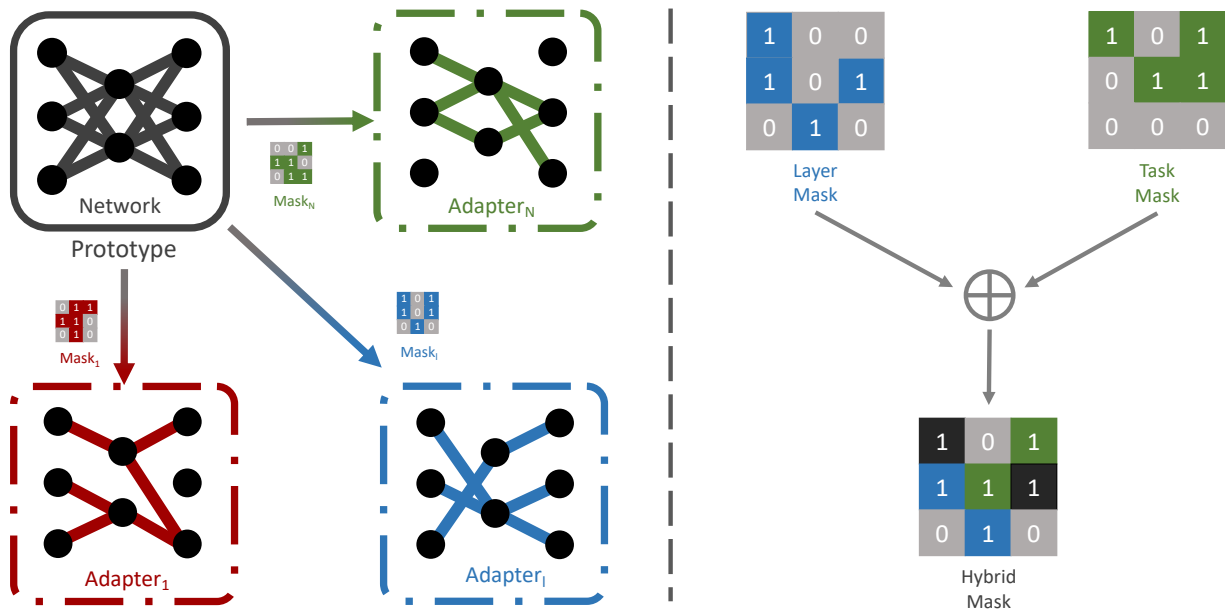
Figure 2: Overview of our PROPETL method. In the left part, the grey neural network indicates the prototype network. Using various binary masks, we can derive sub-networks by pruning certain connections, as depicted by the green (top right), red (down left), and blue (down right) networks. In the right part, we learn layer masks and task masks under multi-task learning. Given a specific Transformer (Vaswani et al., 2017) layer and task to handle, PROPETL generates a hybrid mask by performing an OR logical operation on the layer mask and the task mask. It then uses hybrid masks to generate different sub-networks from the prototype.

2019; Bai et al., 2022; Sreenivasan et al., 2022). Xu et al. (2021) further propose a method that prunes the backward gradient of the neural network, as opposed to pruning the network parameters themselves. Based on these methods, some works (He et al., 2022b; Ma et al., 2022) also proposed to combine pruning algorithms with parameter-efficient methods to further decrease the additional module size. However, those methods only focus on discarding redundant parameters. A parameter is either discarded or retained without any sharing. They fail to make full use of the additional parameters and cannot achieve highly sparse sub-networks without significantly compromising accuracy.

2.3 Multi-Task Learning

Multi-task learning (Zhang and Yang, 2022), which involves training a single model to perform well on multiple tasks, has gained popularity as a research direction in machine learning. However, this approach can be hindered by catastrophic forgetting (Kirkpatrick et al., 2016) and data imbalance among tasks, which will result in overfitting on low-resource tasks and underfitting on high-resource tasks (Arivazhagan et al., 2019). Houlsby et al. (2019) propose adapter tuning that only introduces and updates small additional parameters for each task while freezing the pre-trained model. Based on such a parameter-efficient method, Mahabadi et al. (2021) train a hyper-network named Hyperformer, which generates task-specific weights for the adapter modules when fed with different task embeddings.

3 Methods

In this section, we first give an introduction to parameter-efficient transfer learning (PETL). We then present our method PROPETL, depicted in Figure 2, which combines the techniques of parameter-sharing and pruning to further improve the parameter efficiency compared to existing PETL methods.

3.1 Preliminaries

In parameter-efficient transfer learning, we freeze the parameters θ_{lm} of the pre-trained language model and then introduce additional fine-tunable parameters denoted as θ_t. Given dataset $\{X_i, Y_i\}_{i=1}^N$, the goal of parameter-efficient fine-tuning is to maximize the following likelihood of the label Y by only updating the additional parameters θ_t:

$$\max_{\theta_t} \sum_{i=1}^N \log P(Y_i|X_i; \theta_{lm}, \theta_t) \qquad (1)$$

7566

Such parameter-efficient methods suggest a more effective way to adapt pre-trained language models over downstream tasks than fully fine-tuning. We give a brief introduction of the three most widely used PETL modules, namely adapter (Houlsby et al., 2019), prefix-tuning (Li and Liang, 2021), and LoRA (Hu et al., 2022) in Appendix A. However, there is still a storage limitation when we handle a large range of tasks using these methods. In this paper, we investigate the potential for further enhancing parameter efficiency in neural network models by reducing storage requirements. While previous PETL methods have primarily focused on decreasing the number of parameters to improve efficiency, our approach posits that employing varying bit lengths (e.g., 1-bit, 8-bit, 32-bit) during storage can lead to significant improvements in parameter efficiency by reducing the overall number of bits used by the parameters. To this end, we use bits to measure the storage, which we call Bit-Level Storage (BLS), to take into account the fact that different parameters may have different bit lengths. Consider a neural model, where each parameter has a specific bit length. Then we divide these parameters into N distinct groups based on their respective bit lengths. Let $\{\rho_i\}_{i=1}^N$ denote the number of parameters within the group i, with corresponding bit lengths $\{b_i\}_{i=1}^N$. The BLS for these parameters can subsequently be determined as follows:

$$\text{Bit-Level Storage} = \sum_{i=1}^{N} \rho_i b_i \qquad (2)$$

3.2 Shared Prototype Network

Parameter-efficient methods like adapter and prefix-tuning tend to introduce an additional module in each Transformer layer. Assuming the Transformer has L layers, we can split the parameters θ_t into $[\theta_{t,1}, ..., \theta_{t,L}]$ according to their layer indexes. Therefore, we can rewrite Equation 2 as:

$$\max_{\theta_{t,1},...,\theta_{t,L}} \sum_{i=0}^{N} \log P(Y_i|X_i; \theta_{lm}, [\theta_{t,1}, ..., \theta_{t,L}]) \qquad (3)$$

Inspired by ALBERT (Lan et al., 2020b), in our methods, we first introduce additional parameters for a single PETL module as our prototype network, denoted as θ_{pro}. Then, we share the prototype network across different layers. Assuming that the number of parameters in a single PETL module is

---

Algorithm 1: PROPETL training algorithm

Input: Dataset $\mathcal{D} = (x_i, y_i)$, prototype network learning rate λ_p, mask learning rate λ_m, number of layers L, sparsity $k\% \in [0, 1]$, pretrained parameters θ_{lm}
Output: Parameter of the prototype network $\theta_{pro} \in \mathbb{R}^n$, binary mask across layers $m \in \mathbb{R}^n$

1   $\theta_{pro} \leftarrow$ randomly initialize in $\mathbb{R}^n$
2   $s \leftarrow$ randomly initialize in $\mathbb{R}^n$
3   **for** (x_i, y_i) *in* D **do**
     /* Apply masks in each layer   */
4   **for** l *in* $1, 2, ..., L$ **do**
5       $m_l \leftarrow h(s_l, k)$
6       $\theta_{sub,l} \leftarrow \theta_{pro} \odot m_l$
7     $\theta_{pro} \leftarrow \theta_{pro} - \lambda_p \nabla_{\theta_{pro}} \log P(y_i|x_i; \theta_{lm}, \theta_{sub})$
8     $s \leftarrow s - \lambda_m \nabla_s \log P(y_i|x_i; \theta_{lm}, \theta_{sub})$
9   **for** l *in* $1, 2, ..., L$ **do**
10    $m_l \leftarrow h(s_l, k)$
11   **return** θ_{pro}, m

---

n, we can decrease total parameters from nL to only L, which significantly improves the parameter efficiency. Therefore, we convert the objective function to a more concise one:

$$\max_{\theta_{pro}} \sum_{i=0}^{N} \log P(Y_i|X_i; \theta_{lm}, \theta_{pro}) \qquad (4)$$

3.3 Masked Sub-Networks

Sharing the parameters alone will reduce the model's capacity to capture meaningful representation in different layers, leading to suboptimal results. Inspired by Zhou et al. (2019) and Ramanujan et al. (2020), we believe that parameters and network structures are both crucial contributing factors to the model's representative capacity. To this end, we introduce different binary masks $m_l = \{0, 1\}^n$ in each Transformer layer l (left part of Figure 2), where n denotes the number of parameters in a single PETL module. Each mask represents a corresponding subnetwork of the shared prototype network. Even though the prototype is shared among all layers, we can use different masks to create different sub-networks for each layer l, whose parameter will be $\theta_{sub,l} = \theta_{pro} \odot m_l$, where $\odot$ indicates the element-wise product. With this, we can get our final objective function as:

$$\max_{\theta_{pro},m_1,m_2,...,m_L} \sum_{i=0}^{N} \log P(Y_i|X_i; \theta_{lm}, \theta_{sub}) \quad (5)$$

where $\theta_{sub} = [\theta_{pro} \odot m_1, \theta_{pro} \odot m_2, ..., \theta_{pro} \odot m_L]$.

To learn such masks, we develop our training algorithm based on the edge-popup approach (Ramanujan et al., 2020). Specifically, for each binary mask m_l, we introduce floating-point scores $s_l \in \mathbb{R}^n$. In the forward pass, we generate the binary mask m_l by setting the top-$k\%$ with the largest absolute value in s_l as 1 and the rest as 0. We denote such top-$k\%$ thresholding function as h, and $m_l = h(s_l)$. We refer to the hyperparameter $k\%$ as the sparsity ratio in subsequent sections of this paper. During the backpropagation, we use the straight-through gradient estimator (Bengio et al., 2013) to approximate the gradient of the scores, where function $h(\cdot)$ is treated as the identity function. In addition to training the masks, we also jointly optimize the prototype network.

Our approach employs learnable floating-point scores for each binary mask during fine-tuning, leading to nL parameters. When integrated with the prototype network, PROPETL updates a total of $nL + n$ parameters during the training phase, which is marginally more than the conventional PEFT methods which have nL extra parameters. A comprehensive overview of the PROPETL algorithm can be found in Algorithm 1. After training, we discard the floating-point scores and retain only the binary masks (1-bit) together with the shared prototype network (32-bit). Assuming that the 32-bit prototype network requires p bit-level storage and the binary mask of the same dimension demands $p/32$, our PROPETL can achieve a substantial decrease in storage from around pL to $p(1 + L/32)$. To reconstruct the network structure during inference, we adhere to the following steps: (1) first load the unpruned, shared 32-bit prototype network, (2) then load the binary masks (1-bit) for each layer/task, and (3) extract and use the pruned subnets from the shared prototype network based on specific binary masks during the inference step.

3.4 Hybrid Masks for Multi-Task Learning

Rather than just sharing a PETL module across layers under single-task learning, we can also allow for efficient sharing of the prototype network across multiple tasks. In our approach, we leverage layer masks, as introduced in the previous section, to support parameter sharing within the model. Additionally, we introduce task masks to support parameter sharing across multiple tasks. By performing a logical OR operation[2] on these masks, we can obtain a hybrid mask for a specific layer in a specific task, as shown in the right side of Figure 2.

$$m_{hybrid} = m_{layer} \vee m_{task} \qquad (6)$$

With the design of the hybrid mask, given T tasks and N layers in the pre-trained language models, we only require one PROPETL module, N layer masks, and T task masks, further reducing the BLS from the single task scenario (e.g., 0.011% BLS as in Table 2). In addition, layer masks and task masks, which can be combined as hybrid masks, will potentially help infuse the knowledge into the shared prototype network from layers and tasks.

4 Experimental Setup

We briefly summarize the experimental setup in this section. More details can be found in Appendix C.

Datasets We evaluate PROPETL on a wide range of benchmarks, including language understanding (GLUE (Wang et al., 2018)), text summarization (XSum (Narayan et al., 2018)), and machine translation (WMT16 Ro-En (Bojar et al., 2016)).

Backbones We use RoBERTa$_{\text{BASE}}$ (Liu et al., 2019) for single-task learning on GLUE. During fine-tuning, we only tune our PROPETL module and the text classification head. For generation and multi-task learning benchmark, we use T5$_{\text{BASE}}$ (Raffel et al., 2020) and only tune the PROPETL module. Note that some previous works also tune the layer norms (Houlsby et al., 2019; Mahabadi et al., 2021) while we keep them frozen during fine-tuning.

PETL Modules and PROPETL We use the Pfeiffer adapter (Pfeiffer et al., 2021) as our adapter module and set the bottleneck dimension as 64 by default. For prefix-tuning, we follow Li and Liang (2021) and choose the prefix length to be 64. As for LoRA tuning (Hu et al., 2022), the bottleneck dimension and the scaling factor α are both set to 32. In PROPETL, we increase the value of α to 48 to scale the representation, as the sparse network will decrease the norm of the output representation. We give a brief summary of these PETL modules and how PROPETL is implemented on top of them in Appendix A. Following Ramanujan et al. (2020), we choose the sparsity ratio $k\%$ of PROPETL as 0.5, which we will also further discuss in Section 6.

---

[2]We provide ablations regarding the choice of mask combining methods in Appendix D.1.

Model	% FT Params	%BLS	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
RoBERTa$_{\text{BASE}}$	100.00%	100.00%	60.94	94.04	87.25/90.76	**91.34/88.53**	90.96/90.70	**87.57**	92.53	73.14	86.16
Prefix (l=64)	0.95%	0.95%	63.27	**94.42**	**89.05/92.01**	88.86/85.18	90.46/90.39	85.76	91.46	63.79	84.97
LoRA (bn=32)	0.95%	0.95%	62.24	93.81	86.76/90.48	88.79/85.15	90.73/90.49	86.59	91.85	67.63	84.96
Adapter (bn=64)	0.95%	0.95%	63.15	94.00	86.93/90.49	89.78/86.52	90.84/90.65	87.10	92.23	70.50	85.65
PROPETL$_{\text{Prefix}}$ (l=64)	1.03%	0.11%	61.81	94.00	87.42/91.00	88.85/85.22	90.48/90.47	85.73	91.05	63.79	84.53
PROPETL$_{\text{LoRA}}$ (bn=32)	1.04%	0.11%	62.16	93.62	88.73/91.80	87.59/83.71	90.92/90.83	85.30	91.75	72.66	85.37
PROPETL$_{\text{Adapter}}$ (bn=64)	1.04%	0.11%	**65.43**	94.15	88.24/91.41	89.40/86.04	**91.34/90.95**	86.53	**92.58**	**76.50**	**86.60**

Table 1: Performance of all models based on RoBERTa on the GLUE tasks under single task settings. Bold fonts indicate the best results. "bn" stands for the bottleneck dimension and "l" refers to the number of prefixes. Here *% FT Params* refers to the percentage of fine-tunable parameters during training (including the underlying floating point score of each pruning mask). *%BLS* indicates the task-specific Bit-Level Storage (defined in Sec 3) calculated against the fully-finetuned counterparts when saving the model weights and during the inference time.

In multi-task learning, we aim to maintain an expected k% around 0.5 for the hybrid mask, so we set the k% to 0.3 for both the layer and task masks.[3]

Evaluation For text generation, we report ROUGE-2 (Lin, 2004) on the XSUM test set and BLEU (Papineni et al., 2002) score on the Ro-En test set. Since the test sets of GLUE are not released publicly, following Zhang et al. (2021) and Mao et al. (2022), when the sample number of the datasets is fewer than 10k (RTE, MRPC, STS-B, CoLA), we divide the original validation set into halves – the first half for validation and the second for testing. As for the other datasets in GLUE, we randomly choose 1k samples from the training set as our validation data and test on the original validation set. we report both accuracy and F1 for MRPC and QQP in GLUE . For STS-B, we report both Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all remaining sub-tasks in GLUE, we report accuracy. Due to high training overhead for generation tasks, we report experimental results with one run for XSum and Ro-En. For GLUE, we report the mean of three different random runs.

5 Results

5.1 Single-Task Learning

Results in Language Understanding In Table 1, we report the performance of PROPETL and various baselines on the GLUE benchmark. Both PROPETL$_{\text{Adapter}}$ and PROPETL$_{\text{LoRA}}$ demonstrate superior performance compared to their respective counterparts (adapter and LoRA). Despite having

slightly more parameters during the fine-tuning process, PROPETL requires only 1/9 (0.11% v.s. 0.95%) of bit-level storage during inference, making them more efficient options. Specifically, PROPETL increases the average score of the adapter by 0.95 and improves the score of LoRA by 0.41. Besides, PROPETL$_{\text{Adapter}}$ remarkably outperforms the fully fine-tuned model (86.60 v.s. 86.16) while using 0.11% storage of the fully fine-tuned model. These results indicate that although with reduced parameters, PROPETL injected with the structure information from masks can make more use of the single prototype network and achieve better performance compared to their counterparts. However, we also found that PROPETL$_{\text{Prefix}}$ did not outperform prefix-tuning, which we believe is caused by the reparameterization in prefix-tuning that has a harmful effect on the mask learning.[4] Overall, PROPETL increases the performance of the adapter to the greatest extent. PROPETL$_{\text{Adapter}}$ also achieves the highest performance among the three PROPETL variants. We will stick to PROPETL$_{\text{Adapter}}$ for the rest of the experiments.

Results in Language Generation To verify PROPETL can also be applied to harder tasks, we evaluate our method on two language generation datasets, XSum and WMT16 Ro-En. The results are presented in Figure 3 (a) and (b). We find that PROPETL$_{\text{Adapter}}$ can perform just as well as the regular adapter method while using significantly less bit-level storage. Additionally, when consuming more than 1.6% of the storage, PROPETL$_{\text{Adapter}}$ is able to achieve competitive performance on the XSum dataset compared with the fully fine-tuned T5. However, both adapter tuning and PROPETL$_{\text{Adapter}}$ do not reach the level of the fully fine-tuned model on Ro-En. One potential explanation is Ro-En is

---

[3]Two random and independent binary masks whose elements have 30% probability to be one will produce a resulting mask with about 50% ones after the OR logical operation, which can be calculated using the equation $P(A \cup B) = P(A) + P(B) - P(A)P(B)$.

[4]See more details and explanation in Appendix A.3

Model	%FT Params per task	%BLS per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
Single-Task Learning											
T5$_{BASE}$ [†]	100.0%	100.000%	54.85	92.19	88.18/91.61	91.46/88.61	89.55/89.41	86.49	91.60	67.39	84.67
Adapter (bn=64)	1.070%	1.070%	62.64	94.07	87.36/91.06	90.25/87.28	89.88/89.55	85.76	92.85	71.01	85.61
Multi-Task Hypernetworks											
Hyperformer++[†]	0.290%	0.290%	63.73	94.03	89.66/92.63	90.28/87.20	90.00/89.66	85.74	93.02	75.36	86.48
Multi-Task Training											
T5$_{BASE}$ [†]	12.500%	12.500%	54.88	92.54	**90.15/93.01**	**91.13/88.07**	88.84/88.53	**85.66**	92.04	75.36	85.47
Adapter (bn=64)	0.130%	0.130%	**62.08**	93.57	89.49/92.64	90.25/87.13	87.54/87.41	85.14	92.80	72.22	85.48
Adapter (bn=6)	0.013%	0.013%	58.34	93.61	86.20/90.44	90.10/86.98	86.96/86.66	84.02	92.38	67.63	83.94
PROPETL$_{Adapter}$ (bn=64)	0.156%	0.011%	61.43	**94.22**	87.36/90.97	90.13/87.14	90.32/90.12	85.34	**93.01**	75.60	**85.97**
PROPETL$_{Adapter}$ (bn=6)	0.016%	**0.001%**	54.59	93.53	87.36/91.02	90.15/87.04	**90.70/90.50**	85.08	92.79	**75.86**	85.32

Table 2: GLUE Results on T5. Under single-task learning, we train each task with different model copies. As for multi-task training, we train a unified model or adapter. Results marked with † are from the implementation of Mahabadi et al. (2021). Bold fonts suggest the best results in the block.

harder because translation knowledge may not be covered a lot during the pre-training process of T5. To perform well on such tasks, the model needs to learn additional knowledge and requires more tunable parameters during fine-tuning. We note that such sub-optimal performance on hard generation tasks is not only a nature of PROPETL but generally exists in all PETL methods. Similar findings are also presented in Raffel et al. (2020) and He et al. (2022a). Overall, these experiments show that our PROPETL is also more parameter-efficient on text generation benchmarks compared to existing PETL methods.

5.2 Multi-Task Learning

We present the results of multi-task learning and also provide baseline results from single-task learning using the T5$_{BASE}$ in Table 2. Our best-performing model, PROPETL$_{Adapter}$ with bottleneck dimension of 64, surpasses the fully fine-tuned T5. We also compare PROPETL with Hyperformer++ (Mahabadi et al., 2021), a hyper-network that is specifically designed to transfer knowledge across tasks. The latter uses significantly more task-specific bit-level storage (26×: 0.011% v.s. 0.29% per task), while only increasing the average score by 0.51. Compared to the vanilla adapter, PROPETL$_{Adapter}$ can be marginally better under a similar fine-tuning parameter budget but with 1/9 of the original storage. Besides, we experiment with an extreme case, where we set the bottleneck dimension to 6. Our results show that the accuracy of adapter tuning decreases from 85.48 to 83.94, while PROPETL$_{Adapter}$ still maintains a comparable performance to the fully fine-tuned model (85.32 v.s. 85.47) with a remarkably small percentage

Model	%FT Params per task	%BLS per task	Avg
T5$_{3B}$	25%	25%	88.92
Adapter (bn=64)	0.0278%	0.0278%	88.31
PROPETL$_{Adapter}$ (bn=64)	0.0283%	0.0016%	**89.02**

Table 3: Multi-task learning results on the GLUE using T5$_{3B}$ as the backbone.

(0.001%) of bit-level storage per task. This demonstrates that normal adapter tuning can not make full use of the parameters and may fail to perform well with a relatively small bottleneck dimension. However, in the case of PROPETL$_{Adapter}$, even with a bottleneck dimension that is only 6, it can still achieve a reasonable result. To further validate that PROPETL is effective for larger-sized models, we also carry out experiments on the T5 3B variant and present the findings in Table 3. The outcomes align with our conclusions drawn from the T5 base model. We believe that, in PROPETL$_{Adapter}$, the structure information learned by the mask can make up for the performance drop due to the shrink of the bottleneck dimension to a certain extent. We further compare adapter tuning with PROPETL$_{Adapter}$ with different percentages of bit-level storage by varying the bottleneck dimension. The results are presented in Figure 3 (c). It shows PROPETL$_{Adapter}$ is able to reach the fine-tuned performance with as few as 0.002% task-specific bit-level storage. We can also see that the curve shares a similar trend with those in Figure 3 (a) and (b), which we will further discuss in the next section.

6 Discussion

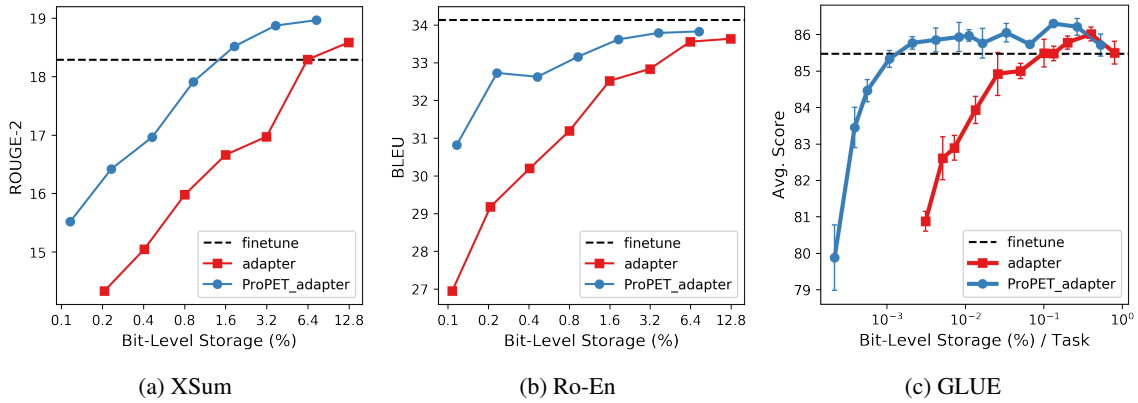**How does PROPETL Scale Differently to Adapter?** Figure 3 presents the results when we

Figure 3: Performance of adapter and PROPETL_Adapter on XSum (left), Ro-En (middle), and GLUE (right) with T5_BASE model. We train on XSum and Ro-En under single-task settings for 1 run. We train GLUE under multi-task learning and report the average score with 3 runs. We additionally provide these results in table format in Appendix E.

adjust the size of the adapter and PROPETL_Adapter on three different datasets. Despite the difference in tasks and methods, we discover that adapter and PROPETL_Adapter show similar scaling trends on all three datasets when we increase the proportion of the task-specific bit-level storage. Their performance increases linearly with respect to the log scale of the extra storage in the beginning. When the adapter and PROPETL_Adapter reach close to the performance of the fully fine-tuned model, their performance gradually converges. Even though PROPETL_Adapter and adapter tuning can slightly exceed the fully fine-tuned performance on some datasets, their performance is still bounded to the same level and cannot outperform the fully fine-tuned model by a large margin. For instance, the performance of both adapter and PROPETL starts to drop when the task-specific storage exceed 0.4% per task on GLUE (Figure 3 (c)). However, PROPETL_Adapter is able to reach the fully fine-tuned level much earlier in the scaling curves. Given a fixed amount of task-specific storage, PROPETL_Adapter is also able to achieve better results than the adapter. These results indicate that our share-and-mask method is overall more efficient than the adapter across almost all scales.

Which is More Important, Sharing or Masking? In this section, we discuss the effect of masking and sharing in the prototype network by comparing PROPETL with a random mask baseline and two alternative settings (*only mask* and *only share*). For the random mask setting, we randomly select the sub-network of the prototype module during the forward process rather than relying on the updated mask score. *Only mask* involves not sharing

the module across layers but only learning masks to prune different PETL modules into sparse sub-networks. *Only share* shares a single PETL module among layers without using masks for pruning. Masking without sharing will drastically increase the number of fine-tunable parameters if we keep the same bottleneck dimension and sparsity ratio. To keep the number of fine-tunable parameters on the same level, we either keep the bottleneck dimension the same and reduce the sparsity ratio $k\%$ or use the same sparsity ratio with reduced bottleneck dimension for the *only mask* setting. We also slightly increase the bottleneck dimension of the *only share* set up to compensate for the parameters of masks. Our results, as presented in Table 4, indicate that the random mask setting yields the poorest performance. Moreover, neither *only mask* nor *only share* reaches the same performance with PROPETL, highlighting the necessity to use **both** masking and sharing to achieve higher parameter efficiency.[5] We believe masking injects crucial structural information into the sub-networks, while sharing is necessary to expand the size of each sub-network when the number of fine-tunable parameters is fixed. Therefore, our method can use the parameters more efficiently.

How do Sub-Networks' Size and Structure Affect Each Other? The sparsity ratio $k\%$ is an important hyperparameter in PROPETL. We study the impact of such sub-network sparsity and present the results in Figure 4. The performance of our PROPETL improves as $k\%$ increases from 0.1 to 0.5 but then declines as $k\%$ continues to grow from

---

[5]Detailed setup can be found in Appendix D.2.

Method	PROPETL	Random	Only Mask (Same bn)	Only Mask (Same $k\%$)	Only Share
GLUE					
Adapter (0.11%)	**86.60**	82.29	85.40	84.70	84.32
Prefix (0.11%)	**84.53**	79.56	84.18	84.23	81.57
LoRA (0.11%)	**85.37**	82.48	83.46	84.75	82.53
Ro-En					
Adapter (0.46%)	**32.63**	30.02	31.58	30.68	31.30

Table 4: Ablation studies of the shared network and masks. We report the average score on GLUE based on RoBERTa$_{\text{BASE}}$ under single-task learning. For Ro-En, we report the BLEU score with T5$_{\text{BASE}}$ as the backbone. Numbers in parenthesis indicate the percentage of task-specific bit-level storage calculated against the fully-finetuned model.

0.5 to 1.0. Additionally, it can be seen that all these PETL methods can achieve the best accuracy when $k\%$ is set to 0.5. This is likely due to the fact that as the network becomes denser from 0.1 to 0.5, the sub-networks get to use more parameters and thus obtain better modeling ability. However, after 0.5, the network in different layers becomes more homogeneous as the sub-networks overlap more, leading to less distinctive structural information in each layer. The absence of enough structural information starts to harm the performance, which even outweighs the benefits of potential knowledge sharing across the layers. We also discover that sharing the PETL module without any pruning ($k\% = 1.0$) results in the worst performance among all sparsity levels. These results suggest that given the fixed percentage of tunable parameters, it is crucial to find a good balance between the distinctive structural information of each sub-network and the number of parameters used in each sub-network.

How is PROPETL Conceptually Related to PETL? Other than the explanation of prototype network sharing and masking, our proposed PROPETL can also be considered as a PETL-on-PETL approach, which we refer to as PETL^2. In other words, the mask is to the prototype network (in our approach) as the PETL module is to the PLM (in conventional PETL approaches). Vanilla PETL methods, such as adapter and prefix-tuning, update specific additional parameters for each layer and downstream task while only sharing the parameters of the PLM. In contrast, PROPETL extends this approach by sharing not only the PLM but also the prototype PETL module among layers and tasks, resulting in a higher degree of parameter sharing. Our method uses binary masks that function like PETL modules on top of the prototype
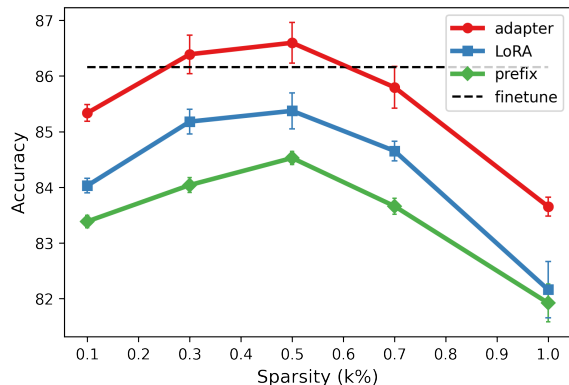

Figure 4: Average score of PROPETL on GLUE with different sparsity ratios under single-task learning on RoBERTa$_{\text{BASE}}$.

PETL module to prune different structures in different sub-networks. These task-specific tunable parameters are thus an order of magnitude smaller than conventional PETL modules.

7 Conclusion and Future Work

In this paper, we introduce PROPETL, a method for sharing prototype PETL modules across different layers and tasks. Our method significantly improves the parameter efficiency by utilizing the prototype network and maintaining a binary mask for each layer and task. Extensive experiments show that our method achieves comparable performance to fully fine-tuned models and conventional PETL methods with a much smaller fraction of storage. For future works, we aim to study the interpretability of the masks in different layers and explore their potential relationships. We also intend to apply our method to the pre-training process of large language models to reduce the overall number of parameters.

Limitations

Although our masks in different layers are binary and require significantly less storage compared to other PETL networks, we still need the underlying 32-bit scores for each mask during the training process. Therefore, PROPETL consumes slightly more memory in the training time than existing PETL methods. To fine-tune PROPETL, it takes a similar training time to conventional PETL modules, which means our method will normally take a longer time to converge compared to the fully fine-tuned model.

Acknowledgements

References

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George F. Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *CoRR*, abs/1907.05019.

Yue Bai, Huan Wang, Xu Ma, Yitian Zhang, ZHIQIANG TAO, and Yun Fu. 2022. Parameter-efficient masking networks. In *Proceedings of NeurIPS*.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Findings of the 2016 Conference on Machine Translation*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL*.

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

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

Shwai He, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. 2022b. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *CoRR*, abs/2210.04284.

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 ICML*.

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 *Proceedings of ICLR*.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2016. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796.

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

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

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2019. Snip: single-shot network pruning based on connection sensitivity. In *Proceedings of ICLR*.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of EMNLP*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*.

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. *CoRR*, abs/1907.11692.

Fang Ma, Chen Zhang, Lei Ren, Jingang Wang, Qifan Wang, Wei Wu, Xiaojun Quan, and Dawei Song. 2022. Xprompt: Exploring the extreme of prompt tuning. *CoRR*, abs/2210.04457.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of ACL*.

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 ACL*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of EMNLP*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of EACL*.

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 EMNLP*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*.

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. *Journal of Machine Learning Research*, 21(140):1–67.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2020. What's hidden in a randomly weighted neural network? In *Proceedings of CVPR*.

Kartik Sreenivasan, Jy yong Sohn, Liu Yang, Matthew Grinde, Alliot Nagle, Hongyi Wang, Eric Xing, Kangwook Lee, and Dimitris Papailiopoulos. 2022. Rare gems: Finding lottery tickets at initialization. In *Proceedings of NeurIPS*.

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 *Proceedings of NeurIPS*.

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 EMNLP*.

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 EMNLP*.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9514–9528, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2021. Revisiting few-sample {bert} fine-tuning. In *Proceedings of ICLR*.

Yu Zhang and Qiang Yang. 2022. A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.*, 34(12):5586–5609.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proceedings of NeurIPS*.

A Implementation Details of the Three Variants of PROPETL

A.1 PROPETL_Adapter

An adapter module modifies a model's hidden representation through a down-sampling projection and an up-sampling projection with a non-linear layer in between:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}W_{\text{down}} + b_{\text{down}})W_{\text{up}} + b_{\text{up}} \quad (7)$$

where W represents the weight matrix, f denotes the non-linear layer and b is the bias term. In PROPETL_Adapter, we apply our binary pruning masks on the up-sampling and down-sampling weights, respectively:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}\widetilde{W}_{\text{down}} + b_{\text{down}})\widetilde{W}_{\text{up}} + b_{\text{up}} \quad (8)$$

where $\widetilde{W}_{\text{down}} = W_{\text{down}} \odot m_{\text{down}}$ and $\widetilde{W}_{\text{up}} = W_{\text{up}} \odot m_{\text{up}}$.

The original Houlsby adapter (Houlsby et al., 2019) introduces the adapter module after each

multi-head attention and feed-forward layer. Pfeiffer et al. (2021) later propose a more efficient variant of the adapter that is only inserted after the feed-forward layer.

A.2 PROPETL$_{\text{LoRA}}$

LoRA (Hu et al., 2022), like adapter tuning, also includes a down-sampling projection and an up-sampling projection. The difference is that LoRA does not have any non-linear layers, but it does have an additional scaling factor $\alpha \geq 1$:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + \alpha \cdot \boldsymbol{x} W_{\text{down}} W_{\text{up}} \quad (9)$$

To modify PLMs' hidden representation, LoRA is applied to the query and value representations of the attention modules in Transformers. In PROPETL$_{\text{LoRA}}$, the network is pruned with binary masks:

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + \alpha \cdot \boldsymbol{x} (W_{\text{down}} \odot m_{\text{down}})(W_{\text{up}} \odot m_{\text{up}}) \quad (10)$$

The scaling factor α is an important parameter for LoRA and Hu et al. (2022) use different α for different datasets in their released code. In PROPETL$_{\text{LoRA}}$, applying the pruning masks m will reduce the norm of the output of the LoRA module. We find that applying a larger α for PROPETL$_{\text{LoRA}}$ than that used in LoRA will remedy the issue of reduced feature norm and result in better performance.

A.3 PROPETL$_{\text{Prefix}}$

Li and Liang (2021) propose to insert tunable matrices, which they call prefixes, into the key-value pair of the Transformers' attention modules:

$$\begin{aligned} H &= \text{Attention}(Q, [P_k; K], [P_v; V]) \\ &= \text{softmax}(\frac{Q[P_k; K]^T}{\sqrt{d}})[P_v; V] \end{aligned} \quad (11)$$

where $[\cdot; \cdot]$ stands for matric concatenation. They also find that directly updating the P parameters will lead to unstable training and a bit drop in performance. In order to ameliorate the problem, they propose to reparametrize the matrix:

$$P = P'W \quad (12)$$

where P' is a smaller learnable matrix and W is the weight of an up-projecting feedforward neural

Module	Vanilla	Prune 50%
Prefix (l=64)	**84.97**	84.50
LoRA (bn=32)	84.96	**85.44**
Adapter (bn=64)	85.65	**86.56**

Table 5: Average score between pruning or not on the GLUE dataset based on RoBERTa$_{\text{BASE}}$ single-task learning under conventional PETL methods.

network. In PROPETL$_{\text{Prefix}}$, we apply our binary pruning masks on the reparametrized prefixes:

$$H = \text{Attention}(Q, [P_k \odot m_k; K], [P_v \odot m_v; V]) \quad (13)$$

Note that different from adapter and LoRA tuning, the pruning masks in PROPETL$_{\text{Prefix}}$ do not directly operate on the parameters of the network. Instead, the masks are applied on P_k and P_v that are output activation depending collectively on W and P'. Thus, it might be hard for the mask training process to identify good structures from P_k and P_v, which potentially explains the sub-optimal results of PROPETL$_{\text{Prefix}}$ in Table 1. To verify this claim, we further compare 3 PETL modules against their counterparts pruned with a sparsity of 0.5 in Table 5. We find that both adapter and LoRA tuning improve their performance when 50% of parameters are pruned, which substantiates the idea that structural information of sub-networks is important. However, for prefix tuning that uses reparameterization, its performance drops when we learn pruning masks on top of it. This shows that the masks fail to locate suitable structures for prefix tuning.

B Parameter Efficiency in PROPETL

B.1 Overview

Learning Setup	Bit-Level Storage of PROPETL
Single-task Learning	$p + 1/32 pL$
Multi-task Learning	$p + 1/32 p(L + T)$

Table 6: Storage calculation.

We present an approximate calculation of the bits (space) required by PROPETL during inference in Table 6, in which we assume that a single shared PETL network contains p BLS and the model has L layers. In addition, the storage space required for the binary mask makes up $1/32$ (≈ 0.031) of the 32-bit PETL module. Depending on the specific PETL module used, the calculations may vary a bit, and we detail them in the following sections.

Datasets	XSum	Ro-En	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE
#Train	204k	997k	8.6k	66k	3.7k	363k	5.7k	392k	104k	2.5k
#Valid	11k	2.6k	0.5k	1k	0.2k	1k	0.8k	1k	1k	0.1k
#Test	11k	3k	0.5k	0.9k	0.2k	40k	0.8k	10k	5k	0.1k

Table 7: Dataset statistics

B.2 PROPETL_Adapter

Given the number of layers (L), hidden dimension (d) of the pre-trained language model, and bottleneck dimension (bn) of the adapter module, the BLS consumed by PROPETL_Adapter is:

$$\underbrace{32 \cdot (2 \cdot bn \cdot d + bn + d)}_{\text{Prototype adapter}} + \underbrace{2 \cdot bn \cdot d \cdot L}_{\text{Mask}} \quad (14)$$

Note that our method does not apply any masks on the bias terms in the prototype adapter. Since our PROPETL will reuse the prototype network, we do not consider the pruning ratio $k\%$ when calculating the storage. Here we also discuss how to calculate the storage cost of parameters used under the *only mask* setting in Table 4, in which we do not share the PETL module across layers but only learn masks to prune them. In that case, we do not count the pruned parameters because they will never be reused. Therefore, the formula for the BLS required by *only mask* is:

$$32 \cdot (2 \cdot k\% \cdot bn \cdot d + bn + d) \cdot L \quad (15)$$

B.3 PROPETL_LoRA

Similarly, given the number of layers (L), hidden dimension (d) of the pre-trained language model, and bottleneck dimension (bn) of the LoRA module, the formula for the storage needed by PROPETL_LoRA is:

$$\underbrace{32 \cdot 4 \cdot bn \cdot d}_{\text{Prototype LoRA}} + \underbrace{4 \cdot bn \cdot d \cdot L}_{\text{Mask}} \quad (16)$$

Note that there is no bias term in LoRA. Under the *only mask* setup, given the sparsity ratio $k\%$, the storage of parameters is:

$$32 \cdot 4 \cdot k\% \cdot bn \cdot d \cdot L \quad (17)$$

B.4 PROPETL_Prefix

Given the number of layers (L), hidden dimension (d) of the pre-trained language model, and the prefix length (l) of the prefix module, the formula to calculate the BLS of PROPETL_Prefix is:

$$\underbrace{32 \cdot 2 \cdot l \cdot d}_{\text{Prototype prefix}} + \underbrace{2 \cdot l \cdot d \cdot L}_{\text{Mask}} \quad (18)$$

Under *only mask* settings, given the sparsity ratio $k\%$, the approximate required storage is:

$$2 \cdot k\% \cdot l \cdot d \cdot L \quad (19)$$

C Experimental Details

We briefly introduce the benchmark datasets used in this work. Their statistics can be found in Table 7.

C.1 Datasets

GLUE The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is widely used to benchmark models' language understanding ability. It consists of a broad range of sentence-level tasks, including natural language inference (MNLI, QNLI, RTE), sentence similarity (MRPC), paraphrase detection (QQP, STS-B), and single sentence classification (SST-2, CoLA).

XSum The Extreme Summarization dataset (XSum) (Narayan et al., 2018) is designed to evaluate systems' abstractive summarization ability of a single document. It is collected from the online articles of the British Broadcasting Corporation (BBC). The input is a single document with an average token count of 431.07, and the model is expected to generate a short, single-sentence summarization of this document.

WMT16 Ro-En WMT16 is the 2016 edition of the Workshop on Machine Translation (WMT) dataset (Bojar et al., 2016). This dataset is widely used to evaluate machine translation systems. To benchmark on WMT16 Ro-En, the model is supposed to take in a Romanian sentence and output translation in English.

Model	Mask_lr
PROPETL_Adapter	3e-3
PROPETL_LoRA	3e-2
PROPETL_Prefix	1e-4

Table 8: Learning rates in GLUE under single task settings

Datasets	Training time
MNLI	90min
QQP	81min
QNLI	24min
SST-2	15min
CoLA	4min
STS-B	3min
MRPC	2min
RTE	1min

Table 9: The approximate training time in GLUE with PROPETL under single-task training

C.2 Implementation Details

Single-Task Learning on RoBERTa$_{BASE}$ Our models are implemented based on the Adapter-Hub package (Pfeiffer et al., 2020). We use the *datasets* (Lhoest et al., 2021) library to calculate each sub-task's scores in GLUE.

The learning rate of the PETL module is set to 1e-4, and we detail the learning rate of pruning masks in Table 8. We find that it is important to set a higher learning rate for the masks than the learning rate of the PELT network. The batch size is set to 128 and the weight decay is 0.1 in all experiments with Roberta. When conducting experiments on GLUE datasets, we train 10 epochs when the dataset is large (MNLI, QNLI, QQP, SST-2), while training 20 epochs on the small dataset (RTE, MRPC, STS-B, CoLA). Table 9 shows the training time taken on a single A100 GPU.

Single and Multi-Task Learning on T5$_{BASE}$ We implement T5 based on the transformers library (Wolf et al., 2020). We use the ROUGE package (Lin, 2004) for ROUGE-2 calculation and sacrebleu (Post, 2018) for the BLEU score. Table 10 shows the training time on a single A100 GPU and the detailed hyperparameters used. We mainly follow He et al. (2022a) and Mahabadi et al. (2021) to select the hyperparameters and do not perform an exhaustive search. The same set of hyperparameters is used across the fully-finetuning, adapter tuning, and PROPETL models. For GLUE, we follow Mahabadi et al. (2021) to sample data from each GLUE sub-task with a temperature of 10. Specifically, a sub-task is sampled with probability $p_\tau^{1/T}$ where $p_\tau = \frac{N_\tau}{\sum_{i=1}^{T} N\tau}$ and $T = 10$.

D Additional Ablation Studies

D.1 Choice of Mask Combining Methods

As shown in Table 11, using the OR logical operation to combine the layer mask and the task mask

Hyperparameters	XSum	Ro-En	GLUE
Batch size	64	64	128
Total steps	100k	60k	20k
Learning rate	1e-4	3e-4	3e-4
Mask's learning rate	1e-3	3e-3	3e-3
Learning rate schedule	linear	linear	linear
Label smoothing	0.1	0.1	0.1
Weight decay	0.01	0.01	0.0
Sampling temperature	n.a.	n.a.	10
Training time	1 day	6 hours	2 hours

Table 10: Hyperparameters on XSum, Ro-En, and GLUE when T5$_{BASE}$ is used as the backbone.

Combining Method	Avg. GLUE Score
OR	**85.97**
ADD	85.66
AND	85.57

Table 11: Results of different mask combining methods based on T5 multi-task learning on GLUE. We set the bottleneck dimension to 64.

Method	PROPETL	Only Mask (Same bn)	Only Mask (Same k%)	Only Mask (Same k% and bn)	Only Share
GLUE					
Adapter	64/50%	64/11%	14/50%	64/50%	88/100%
Prefix	64/50%	64/15%	15/50%	64/50%	88/100%
LoRA	32/50%	32/15%	8/50%	32/50%	44/100%
% BLS	(0.11%)	(0.11%)	(0.11%)	(0.48%)	(0.11%)
Ro-En					
Adapter	384/50%	384/8%	55/50%	384/50%	673/100%
% BLS	(0.46%)	(0.46%)	(0.46%)	(3.2%)	(0.46%)

Table 12: The bottleneck dimension/sparsity ratio used in Table 13.

Method	**PROPETL**	Only Mask (Same bn)	Only Mask (Same k%)	Only Mask (Same k% and bn)	Only Share
GLUE					
Adapter	**86.60**	85.40	84.70	86.56	84.32
Prefix	**84.53**	84.18	84.23	84.50	81.57
LoRA	85.37	83.46	84.75	**85.44**	82.53
% BLS	(0.11%)	(0.11%)	(0.11%)	(0.48%)	(0.11%)
Ro-En					
Adapter	32.63	31.58	30.68	**33.28**	31.30
% BLS	(0.46%)	(0.46%)	(0.46%)	(3.2%)	(0.46%)

Table 13: Ablation studies of the shared network and masks. We report the average score on GLUE based on RoBERTa$_{BASE}$ under single-task learning. For Ro-En, we report the BLEU score with T5$_{BASE}$ as the backbone.

achieves the best performance. This is intuitive because given a specific task and a Transformer layer, parameters that contain the layer information and parameters that contain the task information should be **both** used in the forward pass.

D.2 Choice of Sharing and Masking

We provide additional details of the experiments in Table 4 in this section. With the equations shown in Appendix B, we calculate the bottleneck dimen-

Adapter							
Bottleneck dimension	12	24	48	96	192	384	769
% Bit-Level Storage	0.207%	0.405%	0.803%	1.597%	3.186%	6.363%	12.72%
ROUGE-2	14.33	15.05	15.98	16.66	16.97	18.29	18.58
PROPETL_Adapter							
Bottleneck dimension	96	192	384	768	1536	3042	6144
% Bit-Level Storage	0.116%	0.232%	0.464%	0.927%	1.853%	3.706%	7.412%
ROUGE-2	15.52	16.42	16.96	17.91	18.52	18.87	18.96

Table 14: Performance of adapter and PROPETL_Adapter on XSum under single-task learning when varying the bottleneck dimension. We report the BLEU and the proportion of the bit-level storage.

Adapter								
Bottleneck dimension	6	12	24	48	96	192	384	768
% Bit-Level Storage	0.108%	0.207%	0.405%	0.803%	1.597%	3.186%	6.363%	12.72%
BLEU	26.95	29.18	30.20	31.20	32.52	32.83	33.56	33.63
PROPETL_Adapter								
Bottleneck dimension	–	96	192	384	768	1536	3072	6144
% Bit-Level Storage	–	0.116%	0.232%	0.464%	0.927%	1.853%	3.706%	7.412%
BLEU	–	30.82	32.72	32.63	33.16	33.62	33.79	33.83

Table 15: Performance of adapter and PROPETL_Adapter on En-Ro under single-task learning when varying the bottleneck dimension. We report the BLEU and the proportion of the bit-level storage.

Adapter														
Bottleneck dimension	–	–	–	–	1	2	3	6	12	24	48	64	96	192	384
% BLS per task	–	–	–	–	0.003%	0.005%	0.007%	0.013%	0.026%	0.051%	0.100%	0.133%	0.200%	0.398%	0.795%
Avg. Score	–	–	–	–	80.88	82.61	82.89	83.94	84.92	85.00	85.49	85.48	85.78	86.01	85.50
PROPETL_Adapter														
Bottleneck dimension	1	2	3	6	12	24	48	64	96	192	384	768	1536	3072	–
% BLS per task	0.0002%	0.0004%	0.0006%	0.001%	0.002%	0.004%	0.008%	0.011%	0.017%	0.033%	0.066%	0.132%	0.265%	0.529%	–
Avg. Score	79.88	83.45	84.46	85.32	85.77	85.85	85.93	85.97	85.76	86.05	85.73	86.3	86.21	85.71	–

Table 16: Performance of adapter and PROPETL_Adapter on GLUE under multi-task learning when varying the bottleneck dimension. We report the average score and the proportion of the bit-level storage per task.

sion (bn) and sparsity ratio ($k\%$) to ensure all the settings have a similar BLS to PROPETL. We also add one more setup, *only mask* with the same $k\%$ and bn as PROPETL. Note that this new setup will result in more storage than the other settings. The hyperparameters are listed in Table 12 and the results are detailed in Table 13.

We find that on the GLUE dataset, PROPETL achieves matched performance to this new setup, even though the latter has used more than 4 times the bit-level storage (0.11% v.s. 0.49%). We believe that, with masks, models with only 0.11% of storage can be enough to have a good performance while more storage cost will not have a significant improvement or lead to overfitting under simple tasks like GLUE. However, on the more challenging dataset Ro-En, when we keep the $k\%$ and bn unchanged and only mask the adapter modules, the model indeed improves its performance by 0.65, but this comes at a cost of around $7\times$ more storage of parameters.

E   Additional Results

We additionally present the experimental results from Figure 3 in table format in Table 14, 15, and 16.

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section Limitations.

☑ A2. Did you discuss any potential risks of your work?
Section Ethics Statement.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 4 and Appendix C

☑ B1. Did you cite the creators of artifacts you used?
Section 4 and Appendix C

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Apache License 2.0 gives permission on Commercial use, Modification, Distribution, Patent use and Private use.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. The dataset we applied is a commonly used open-source benchmarks datasets in NLP.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix C

C ☑ Did you run computational experiments?

Section 5 and 6

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Appendix C

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix C

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4, 5 and 6

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section C

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.