FPT: Improving Prompt Tuning Efficiency via Progressive Training

Yufei Huang[1,2,3*], **Yujia Qin**[1,2,3*], **Huadong Wang**[1,2,3], **Yichun Yin**[4], **Maosong Sun**[1,2,3,5†], **Zhiyuan Liu**[1,2,3†], **Qun Liu**[4]
[1]DCST, Tsinghua University, Beijing [2]BNRIST, Tsinghua University, Beijing
[3]IAI, Tsinghua University, Beijing [4]Huawei Noah's Ark Lab
[5]Jiangsu Collaborative Innovation Center for Language Ability, Xuzhou, China
{huang-yf20, qyj20}@mails.tsinghua.edu.cn

Abstract

Recently, prompt tuning (PT) has gained increasing attention as a parameter-efficient way of tuning pre-trained language models (PLMs). Despite extensively reducing the number of tunable parameters and achieving satisfying performance, PT is training-inefficient due to its slow convergence. To improve PT's training efficiency, we first make some novel observations about the prompt transferability of "partial PLMs", which are defined by compressing a PLM in depth or width. We observe that the soft prompts learned by different partial PLMs of various sizes are similar in the parameter space, implying that these soft prompts could potentially be transferred among partial PLMs. Inspired by these observations, we propose Fast Prompt Tuning (FPT), which starts by conducting PT using a small-scale partial PLM, and then progressively expands its depth and width until the full-model size. After each expansion, we recycle the previously learned soft prompts as initialization for the enlarged partial PLM and then proceed PT. We demonstrate the feasibility of FPT on 5 tasks and show that FPT could save over 30% training computations while achieving comparable performance. The codes are publicly available at https://github.com/thunlp/FastPromptTuning.

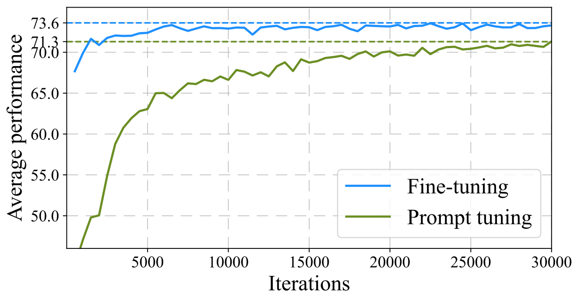

Figure 1: Average performance growth of T5_{LARGE} on 5 investigated tasks in this paper comparing fine-tuning and PT. The convergence speed of PT is much slower than fine-tuning in terms of training steps.

1 Introduction

The emergence of pre-trained language models (PLMs) has broken the glass ceiling for various NLP tasks (Han et al., 2021). Versatile semantic and syntactic knowledge acquired during pre-training could be leveraged when PLMs are adapted towards a specific downstream task to boost performance. The de facto strategy for such an adaptation is full-parameter fine-tuning, which is computationally expensive and profligate since it requires tuning and storing all the parameters in the PLM for each downstream task. To remedy this, several delta tuning (Ding et al., 2022) (also known as parameter-efficient tuning) algorithms are proposed in place of the vanilla fine-tuning (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2022; Ben Zaken et al., 2022), among which prompt tuning (PT) (Lester et al., 2021) has gained increasing attention recently. PT prepends a few *virtual tokens* to the input text, these tokens are tuned during training while all the other PLM parameters remain frozen. Despite its simple form, PT has been demonstrated to achieve remarkable performance in various NLP tasks. Especially when the scale of the PLM becomes extremely huge, PT could achieve comparable performance to fine-tuning (Lester et al., 2021). Despite extensively reducing the number of tunable parameters and achieving satisfying performance, PT is criticized to be training-inefficient due to the slow convergence (Su et al., 2022) as illustrated in Figure 1, and such incompetence would limit the practical application of PT. Hence in this paper, we explore how to improve PT's training efficiency.

Our motivation is based on novel observations about the prompt transferability among "partial PLMs". Here a partial PLM is defined by compressing a PLM in depth or width, which is im-

*Indicates equal contribution.
†Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 6877–6887
December 7-11, 2022 ©2022 Association for Computational Linguistics

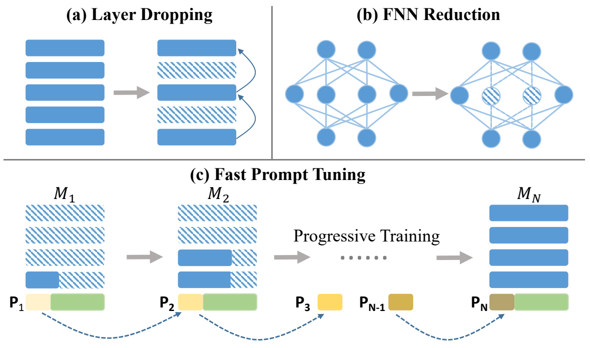

Figure 2: The framework of Fast Prompt Tuning (FPT). The top part (a,b) shows two methods to construct a partial PLM. The bottom part (c) shows FPT's training process, we conduct PT on a partial PLM, progressively expand its size and transfer the trained prompts.

plemented by dropping several layers or masking part of the connections in the feed-forward network (FFN) in each Transformer (Vaswani et al., 2017) layer. We observe that the soft prompts of the same task learned by different partial PLMs of various sizes tend to be close in the parameter space, implying that these soft prompts could potentially be transferred among different partial PLMs.

Inspired by the above observations, we propose Fast Prompt Tuning (FPT), which starts by conducting PT using a small-scale partial PLM to obtain the corresponding soft prompts. After that, we progressively expand the partial PLM's depth and width until the full-model size by rehabilitating the dropped layers and masked neurons. After each expansion, we recycle the previously learned soft prompts as initialization for the enlarged PLM and then proceed PT. Since the partial PLM requires fewer computations for each step, keeping the total training steps unchanged, we could reduce the overall computations consumed, and in the meantime, achieve comparable PT performance. In experiments, we demonstrate the feasibility of FPT on 5 NLP tasks. The experimental results show that FPT could save around 30% training computations and achieve satisfying downstream performance.

2 Prompt Tuning on a Partial PLM

2.1 Prompt Tuning

For a given input sequence $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ and its target label $\mathcal{Y}$, PT first converts $\mathcal{X}$ into a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where d is the hidden size. After that, PT prepends l tunable soft prompt tokens $\mathbf{P} \in \mathbb{R}^{l \times d}$ before $\mathbf{X}$, creating a new input matrix $[\mathbf{P}; \mathbf{X}] \in \mathbb{R}^{(l+n) \times d}$, which is then processed by the PLM. The training objective is to maximize $\mathcal{P}(\mathcal{Y}|[\mathbf{P}; \mathbf{X}])$, where only $\mathbf{P}$ is optimized during training and the parameters of PLM are frozen. Although PT is applied to the entire PLM by default, in this section, we investigate how the performance would become if we conduct PT on a partial PLM, i.e., only part of the parameters in the PLM participate in the computation.

2.2 Partial PLM Construction

Using partial parameters in a PLM is typically applied to reduce the **inference computation** for fine-tuning, such as early exit (Teerapittayanon et al., 2016; Xin et al., 2020) and model pruning (Chen et al., 2020; Sun et al., 2020; Fan et al., 2020), which assume that the features produced by a part of a PLM may already suffice to classify some input examples. In this paper, we investigate its application in reducing the **training computation** of PT, and propose to construct partial PLMs by shrinking the original PLM in both depth and width, as illustrated in Figure 2 (a, b). Details are listed in appendix B.

Layer Dropping. Based on previous findings (Clark et al., 2019; Jawahar et al., 2019) that adjacent layers in PLMs generally have similar functionalities, we hypothesize that removing part of these layers may not significantly hurt the overall performance, and we propose to drop a PLM's layers uniformly to construct a partial PLM consisting of fewer layers than the original PLM. After that, we directly build connections among the remaining layers keeping the original order, which is found empirically to work well although such connections do not exist during pre-training.

FFN Reduction. Jaszczur et al. (2021) and Zhang et al. (2022) indicate that only part of the neurons in the FFN layers will be activated for a given input. Such a sparse activation phenomenon inspires us to reduce the computation in FFN by shrinking the width of the FFN layer. Specifically, the FFN layer consists of two fully connected networks with a nonlinear activation function σ, and it processes an input representation $\boldsymbol{x} \in \mathbb{R}^d$ as: $\text{FFN}(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d'}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d' \times d}$ are the weight matrices, $\boldsymbol{b}_1 \in \mathbb{R}^{d'}$ and $\boldsymbol{b}_2 \in \mathbb{R}^d$ are the bias terms. We abandon a portion of $\boldsymbol{W}_1$ / $\boldsymbol{W}_2$'s columns / rows (i.e., reducing d') by masking the neurons that are seldom activated. In practice, before training, we feed a few downstream examples prepended

	Enc./Dec. Layer	FFN Dimension	MNLI (Acc)	QQP (Acc)	SQuAD2.0 (EM)	ReCoRD (EM)	XSum (ROUGE-L)	Avg.	Δ	FLOPs	Wall Clock
PT	24 / 24	2,816	**86.07**	**87.26**	**76.09**	**81.46**	**26.65**	**71.51**	-	100%	100%
LD	6 / 6	2,816	60.34	78.29	48.14	24.75	17.40	45.78	-25.73	30%	35%
	12 / 12	2,816	63.90	80.64	52.87	39.09	19.69	51.24	-20.27	54%	56%
	18 / 18	2,816	81.41	86.05	63.97	59.87	22.51	62.76	-8.75	77%	77%
FR	24 / 24	704	78.18	85.19	66.68	62.90	22.46	63.08	-8.43	58%	72%
	24 / 24	1,408	82.62	86.45	72.65	74.59	24.61	68.19	-3.32	72%	81%
	24 / 24	2,112	84.93	86.77	74.73	79.52	26.12	70.41	-1.10	86%	91%
CR	6 / 6	704	62.53	78.38	48.62	23.99	16.49	46.00	-25.51	20%	28%
	12 / 12	1,408	64.09	78.91	50.90	29.50	18.88	48.45	-23.06	40%	47%
	18 / 18	2,112	80.63	86.32	63.42	58.97	22.18	62.30	-9.21	66%	71%

Table 1: Average results for partial PLM PT on T5$_{\text{LARGE}}$ with layer dropping (**LD**), FFN Reduction (**FR**), and compound reduction (**CR**). Δ denotes the performance degeneration compared with vanilla **PT** of each setting. The "FLOPs" and "Wall Clock" columns are both relative values compared with **PT** and are averaged over 5 tasks.

by randomly initialized soft prompts into the full-size PLM and record the neuron activation of each dimension of d'.

Compound Reduction. Since the above methods are compatible with each other, we try to combine them to form a partial PLM smaller than the original PLM in both depth and width.

2.3 Observations

To explore PT's performance on a partial PLM, we conduct experiments on T5$_{\text{LARGE}}$ (Raffel et al., 2020). We choose 5 representative NLP datasets in English, covering the tasks of natural language inference (MNLI (Williams et al., 2018)), paraphrase (QQP (link)), reading comprehension (SQuAD2.0 (Rajpurkar et al., 2018) and ReCoRD (Zhang et al., 2018)), and summarization (XSum (Narayan et al., 2018)). For both layer dropping and FFN reduction, we evaluate the performance when we reduce the number of Transformer layers or FFN intermediate dimension to $\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$. We train all models using the same steps and the details are described in appendix B.

Overall Performance. The overall results are shown in Table 1. We observe that for each method, **despite abandoning a large portion of parameters, a partial PLM reserves most of the PT performance of the full-size PLM**. As expected, the performance becomes better when more parameters are retained. In addition, we find that the performance drop is less sensitive to FFN reduction than layer dropping. Specifically, there is only 1.10% performance drop on average when 25% neurons are masked. These results indicate that the resulting partial PLM still retains most of the functionalities of the original PLM.

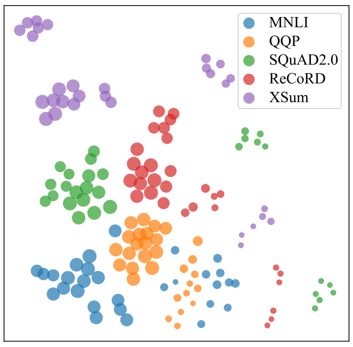

Figure 3: Visualization of 5 investigated tasks' soft prompts of different partial PLMs. A marker with a larger size means the performance of the corresponding soft prompts on the partial PLM is better.

Prompt Embedding Visualization. Taking a step further, we visualize the learned prompt embeddings of different partial PLMs using t-SNE (van der Maaten and Hinton, 2008) in Figure 3, and describe the details in appendix C. We observe that **for the same task, the soft prompts obtained by different partial PLMs tend to form a compact cluster in the parameter space**. This phenomenon implies that the soft prompts corresponding to the same task (1) have a great potential of transferring among different partial PLMs, and (2) could serve as a better initialization that leads to faster convergence. Apart from the visualization, we further report the cosine similarity of the learned prompts in appendix D to verify the above phenomenon from another aspect.

3 Fast Prompt Tuning

In this section, we propose Fast Prompt Tuning (FPT), which aims at accelerating PT via *progressive training* (Gong et al., 2019). Progres-

Method		MNLI (Acc)	QQP (Acc)	SQuAD2.0 (EM)	ReCoRD (EM)	XSum (ROUGE-L)	Avg.	FLOPs	Wall Clock	Improve↑
T5_{LARGE}	PT	86.07	**87.26**	76.09	**81.46**	**26.65**	**71.51**	100%	100%	-
	FPT_{LD}	85.72	86.51	75.89	80.23	26.27	70.92	72%	74%	0.11
	FPT_{FR}	**86.49**	87.11	**76.26**	81.07	26.55	71.50	83%	89%	**0.49**
	FPT_{CR}	85.13	86.40	75.63	81.00	26.21	70.87	**65%**	**70%**	0.38
T5_{XL}	PT	89.00	88.20	81.08	**88.48**	**30.53**	75.46	100%	100%	-
	FPT_{LD}	88.99	88.09	**82.18**	88.06	30.52	**75.57**	86%	86%	**0.78**
	FPT_{FR}	88.84	**88.21**	81.74	88.46	30.52	75.55	84%	87%	0.76
	FPT_{CR}	**89.18**	87.34	80.88	87.82	30.43	75.13	**74%**	**76%**	0.48

Table 2: Performance of the vanilla PT and three variants of our method. FPT_{LD}, FPT_{FR}, and FPT_{CR} refer to constructing partial PLMs by layer dropping, FFN reduction, and compound reduction. The column "Improve↑" denotes the performance improvement of each FPT_* method over PT when PT uses the same FLOPs as FPT_*.

sive training is typically leveraged for improving pre-training efficiency (Chen et al., 2022; Qin et al., 2022), instead, we focus on its application in PLM's downstream adaptation.

3.1 Methodology

Formally speaking, as visualized in Figure 2 (c), we split the original PT training process into N stages. We start with a small-size partial PLM $\mathcal{M}_1$ and then progressively rehabilitate its depth and width until the full-size model $\mathcal{M}_{\text{N}}$, creating a series of partial PLMs $\{\mathcal{M}_i\}_{i=1}^{\text{N}-1}$ with growing sizes. The architectures of the partial PLMs are constructed using the same method in § 2.2.

During each training stage i, we conduct PT on a partial PLM $\mathcal{M}_i$ and obtain the learned soft prompts $\mathbf{P}_i$. Based on the observation that $\mathcal{M}_i$ retains a large portion of functionalities of the full-size PLM $\mathcal{M}_{\text{N}}$, we conjecture that $\mathcal{M}_i$ could serve as a perfect substitute for $\mathcal{M}_{\text{N}}$ and learn how to deal with the downstream task. In addition, considering that the soft prompts learned by different partial PLMs are close in the parameter space, we could transfer the knowledge learned by $\mathcal{M}_i$ to $\mathcal{M}_{i+1}$ through recycling $\mathbf{P}_i$. Specifically, after each model expansion, we directly use $\mathbf{P}_i$ as initialization for training $\mathcal{M}_{i+1}$ in the next stage. Since for each partial PLM, fewer parameters participate in both the forward and backward process, the computations could be reduced. Keeping the total number of training steps the same, FPT could accelerate training compared with vanilla PT.

3.2 Experiments and Analyses

We follow most of the experimental settings in § 2 and also describe the training details in appendix B. We report FLOPs and training wall clock for the vanilla PT and FPT to compare training efficiency. We evaluate both T5_{LARGE} and T5_{XL} (a larger T5 model) on each task and train for 30k and 15k steps, respectively. We test FPT's performance when we progressively expand the model's depth, width, and both of them. Unless otherwise specified, for most of FPT's methods, we split the training process into 4 stages. Each of the first three stages takes 20% steps, while the last stage takes 40% steps.

Results. We list the results in Table 2, from which we observe that (1) on average, all three variants of FPT achieve comparable performance with PT and utilize fewer computations (e.g., FPT_{CR} saves around 30% FLOPs). On several tasks (e.g., MNLI and SQuAD2.0), FPT even exceeds PT's performance; (2) combining both layer dropping and FFN reduction (i.e., FPT_{CR}) is more training-efficient. However, we also observe that saving more computations generally leads to poorer performance. Among all three variants of FPT, FPT_{FR} strikes the best balance between performance and training efficiency; (3) moreover, we compare both PT and FPT's performance when PT consumes the same computations as each variant of FPT. As reflected in the column "Improve↑", controlling the training computations the same, our FPT outperforms PT, and the improvement is more significant for T5_{XL} than T5_{LARGE}, showing that FPT has a great potential to apply to large-scale PLMs. (4) except for using FLOPs as a theoretical analysis of computation resources, we also compare wall clock training time among different FPT methods and vanilla PT. The wall clock time can be also saved at most 30% with our FPT_{CR} method. Besides, the gap between relative FLOPs and relative wall clock time shrinks with the model's size increasing for each FPT method.

We also verify the effectiveness of our partial model construction designs in appendix E, and show in appendix F that the performance of FPT is not sensitive to the duration of each training stage. We leave the explorations on other tasks and the effect of training budgets as future work.

4 Conclusion

In this work, towards improving PT's training efficiency, we first make several insightful observations by conducting PT on partial PLMs, and then propose FPT based on the observations. The results on 5 datasets demonstrate the feasibility of FPT in saving the training computations. Being the first attempt towards accelerating PT, we encourage future work to design more sophisticated algorithms to further improve PT's training efficiency.

Acknowledgements

This work is supported by the National Key R&D Program of China (No. 2020AAA0106502) and Institute Guo Qiang at Tsinghua University.

Yufei Huang and Yujia Qin designed the methods, conducted the experiments, and wrote the paper. Huadong Wang and Yichun Yin participated in the discussion and provided valuable suggestions. Maosong Sun, Zhiyuan Liu, and Qun Liu advised the project.

Limitations

For the current FPT method, there exist two main limitations:

(1) FPT requires choosing a proper hyperparameter of the progressive training steps (i.e. duration of each training stage). For each experiment, we have to pre-define the duration of each stage empirically. Although in appendix F, we have shown that within a reasonable range, the duration of each training stage is not that important.

(2) FPT can not be directly applied to other delta tuning methods (e.g., adapter and prefix-tuning). Since prompt tuning only adds trainable parameters in the embedding layer, when partial model's size increases, the trained soft prompt can be directly transferred to a larger partial model without any modification. But for other popular delta tuning methods, when the layer of partial model increases, we have to add newly initialized parameters.

References

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2022. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148, Dublin, Ireland. Association for Computational Linguistics.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained BERT networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning:

A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. 2021. On the transformer growth for progressive BERT training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5174–5180, Online. Association for Computational Linguistics.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

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, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. 2021. Sparse is enough in scaling transformers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 9895–9907.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

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. 2019. RoBERTa: A robustly optimized BERT pretraining approach.

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 the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Yujia Qin, Xiaozhi Wang, Yusheng Su, Yankai Lin, Ning Ding, Zhiyuan Liu, Juanzi Li, Lei Hou, Peng Li, Maosong Sun, et al. 2021. Exploring low-dimensional intrinsic task subspace via prompt tuning. *arXiv preprint arXiv:2110.07867*.

Yujia Qin, Jiajie Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. ELLE: Efficient lifelong pre-training for emerging data. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2789–2810, Dublin, Ireland. Association for Computational Linguistics.

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.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3949–3969, Seattle, United States. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. 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*, pages 38–45, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems*, volume 33, pages 14011–14023. Curran Associates, Inc.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. ReCoRD: Bridging the gap between human and machine commonsense reading comprehension.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, Dublin, Ireland. Association for Computational Linguistics.

Appendices

A Related Work

Prompt Tuning. PLMs have achieved excellent performance on many NLP tasks relying on their powerful natural language understanding and generation capabilities (Devlin et al., 2019; Liu et al., 2019). However, with the emergence of large-scale PLMs like T5 (Raffel et al., 2020) and GPT-3 (Brown et al., 2020), tuning all the parameters of a PLM (i.e., full-parameter fine-tuning), which requires huge storage and memory costs, is not flexible for real-world applications on massive downstream tasks. Therefore, parameter-efficient delta tuning methods (Ding et al., 2022; Houlsby et al., 2019; Hu et al., 2022; Ben Zaken et al., 2022; He et al., 2022) attract more and more attention, among which prompt tuning (PT) (Lester et al., 2021) is a simple and effective one. By prepending a few trainable embeddings before the input sequence, PT can achieve comparable performance to full-parameter fine-tuning. With the size of PLM getting larger, the performance of PT gets closer to vanilla fine-tuning (Lester et al., 2021), showing great potential to utilize extremely large PLMs in future. Besides, PT is also shown to have excellent cross-task transferability (Su et al., 2022; Vu et al., 2022), and thus gains more and more attention in exploring the relation among tasks (Qin et al., 2021). However, due to the slow convergence shown in Figure 1, PT's training efficiency becomes a serious drawback and may limit its practical application.

Progressive Training. Considering that pre-training usually requires tremendous computational resources, researchers propose *progressive training* to improve training efficiency (Gong et al., 2019; Zhang and He, 2020). Progressive training starts training using a shallow model, and gradually grows the depth of the model along the training process by replicating existing layers (parameter recycling). In this way, the pre-training efficiency can be improved a lot. To further improve training efficiency, later works propose to progressively grow PLMs in both depth and width (Gu et al., 2021), and design better initialization methods to inherit the functionality of existing models (Chen et al., 2022). Instead of leveraging progressive training during the process of pre-training, we apply it to PLM's downstream adaptation, with a focus on PT. Furthermore, conventional progressive training duplicates existing parameters to grow a PLM's size until the full-model's size. Instead, we have already obtained a full-size PLM, and propose to construct partial models with growing sizes by dropping / masking existing parameters.

B Implementation Details

Our implementation is based on PyTorch (Paszke et al., 2019) and transformers (Wolf et al., 2020). The experiments are conducted with 8 NVIDIA 32GB V100 GPUs, and each experiment requires fewer than 10 hours to finish.

Partial PLM Construction. As mentioned in § 2.2, we design three methods to construct partial PLMs. Specifically, for **layer dropping**, we select layers uniformly. For example, to select 3 layers out of a 24-layer PLM, we will select layer $\{1, 12, 24\}$ to construct the partial PLM. For **FFN reduction**, to estimate the activation of each neuron (dimension) in FFN layer l, we first randomly sample $1,000$ examples to form a small dataset $\mathcal{D}$. We prepend each example $\mathcal{X}$ (without the label) in $\mathcal{D}$ with randomly initialized soft prompts and feed it into the full-size PLM to obtain the input representation $\boldsymbol{x}^l$ of FFN layer. After that, we obtain the activation score of each neuron using the following equation $S = \sum_{\mathcal{X}\in\mathcal{D}} \sum_{i=1}^{|\mathcal{X}|} \left|\sigma(\boldsymbol{x}_i^l \boldsymbol{W}_1^l + \boldsymbol{b}_1^l)\right|$, where $\boldsymbol{W}_1^l, \boldsymbol{b}_1^l$ are the parameters in FFN layer l, and $|\mathcal{X}|$ denotes the sequence length. The neurons (dimensions) with smaller activation score (i.e., seldom activated) will be masked. Note that the T5 model is composed of both an encoder and a decoder, due to the difference in the input length and output length on various tasks, the computation overload of the encoder and decoder may vary a lot. Therefore, for the tasks (MNLI and QQP) that have a lighter computation overload on the decoder (i.e., small output length), shrinking the decoder model size has little impact on saving the computational costs, hence we retain the whole decoder under this setting; for other three tasks (SQuAD2.0, ReCoRD and XSum), the output length on decoder is much longer and we conduct partial PLM construction on both the encoder and decoder. We calculate FLOPs for each experiment using the *ptflops* tool [1], and report the average FLOPs of 5 tasks in Table 1 and Table 2.

[1]https://github.com/sovrasov/flops-counter.pytorch

Partial PLMs		Layer Dropping L_{enc}/L_{dec}	d'	FFN Reduction L_{enc}/L_{dec}	d'	Compound Reduction L_{enc}/L_{dec}	d'	Training Steps
T5$_{\text{LARGE}}$	$\mathcal{M}_1$	6 / 6	2,816	24 / 24	704	6 / 6	704	6,000
	$\mathcal{M}_2$	12 / 12	2,816	24 / 24	1,408	12 / 12	1,408	6,000
	$\mathcal{M}_3$	18 / 18	2,816	24 / 24	2,112	18 / 18	2,112	6,000
	$\mathcal{M}_4$	24 / 24	2,816	24 / 24	2,816	24 / 24	2,816	12,000
T5$_{\text{XL}}$	$\mathcal{M}_1$	18 / 18	5,120	24 / 24	1,280	18 / 18	1,280	3,000
	$\mathcal{M}_2$	18 / 18	5,120	24 / 24	2,560	18 / 18	2,560	3,000
	$\mathcal{M}_3$	18 / 18	5,120	24 / 24	3,840	18 / 18	3,840	3,000
	$\mathcal{M}_4$	24 / 24	5,120	24 / 24	5,120	24 / 24	5,120	6,000

Table 3: Architecture details of the partial PLMs on the three construction methods for both T5$_{\text{LARGE}}$ and T5$_{\text{XL}}$. L_{enc} and L_{dec} denote the number of layers of the encoder and decoder of the partial model $\mathcal{M}_i$, respectively. We also list the training steps for each stage in the last column.

Partial PLM Prompt Tuning. We use T5$_{\text{LARGE}}$ for our experiments of partial PLM PT. Following Lester et al. (2021), we leverage the LM-adapted version of T5 checkpoints, which are additionally trained for 100k steps. The adapted version of T5 has been demonstrated to achieve stable and better PT performance. For the implementation of PT, we set the prompt length to 20 and randomly initialize the soft prompts. The optimizer is chosen as Adafactor (Shazeer and Stern, 2018) and the learning rate is set to 0.3. We choose a batch size of 32 and greedy decoding to generate the predictions. The training steps are set to 30k to ensure that PT will not get stuck in a local optimum. We run all the experiments 3 times with different random seeds and report the average results.

Fast Prompt Tuning. For the implementations of FPT, we train T5$_{\text{LARGE}}$ / T5$_{\text{XL}}$ with a total step of 30k / 15k. The number of training steps of T5$_{\text{XL}}$ is chosen smaller than T5$_{\text{LARGE}}$ since we find empirically that T5$_{\text{XL}}$ converges faster than T5$_{\text{LARGE}}$. As mentioned in § 3.2, unless otherwise specified, we split the whole training process into 4 stages. Each of the first three stages takes 20% of the training steps, while the last stage (full-model PT) takes 40% training stages. Except for layer dropping on T5$_{\text{XL}}$, we find that a partial PLM, with fewer than 12 layers in either the encoder or decoder, achieves poor PT performance. Therefore, we only use two training stages where the first stage takes 60% training steps and the second stage takes 40% training steps. More detailed settings about the partial model construction are shown in Table 3. The experiments with T5$_{\text{LARGE}}$ are run three times with different random seeds and the average results are reported while experiments with T5$_{\text{XL}}$ are conducted once due to their huge computation consumption.

C Prompt Embedding Visualization

In Figure 3, we visualize the soft prompts of different partial PLMs and tasks in Table 1. The embedding used for visualization is derived by averaging soft prompt along the token length dimension. As described in § 2.3, we run each experiment three times with different random seeds to get stable results. Therefore, we plot 30 points (3 runs × (9 partial PLM + 1 full-size PLM)) for each task in Figure 3. And the size of the marker in the figure denotes the performance of the soft prompts on corresponding partial PLMs. Larger size indicates better performance. We can observe that soft prompts with better performance will be easier to form a compact cluster.

D Prompt Embedding Similarity

To further gain insights on the transferability of the soft prompts learned by T5$_{\text{LARGE}}$'s different partial PLMs defined in Table 3, in addition to the visualization conducted in § 2.3, we calculate the average cosine similarity of the soft prompts corresponding to different tasks in Table 4. Specifically, for different partial PLMs $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_{\text{N}-1}$ and the full-size PLM $\mathcal{M}_{\text{N}}$, we conduct PT with each model $\mathcal{M}_i$ on the task $\mathcal{T}_j$ and obtain the corresponding soft prompts $\mathbf{P}_i^j \in \mathbb{R}^{l \times d}$. Then we average $\mathbf{P}_i^j$ along the token length dimension and get a vector $\overline{\mathbf{P}}_i^j \in \mathbb{R}^d$. After that, we calculate $\mathcal{S}(\mathcal{T}_j^P, \mathcal{T}_k^F)$ (average cosine similarity between (1) task j's partial PLMs' prompts and (2) task k's full-size PLM's prompts) using the following metric:

$$\mathcal{S}(\mathcal{T}_j^P, \mathcal{T}_k^F) = \frac{1}{\text{N}-1} \sum_{i=1}^{\text{N}-1} \frac{\overline{\mathbf{P}}_i^j \cdot \overline{\mathbf{P}}_{\text{N}}^k}{\|\overline{\mathbf{P}}_i^j\| \|\overline{\mathbf{P}}_{\text{N}}^k\|} \quad (1)$$

$\mathcal{T}^P$ \ $\mathcal{T}^F$	MNLI	QQP	SQuAD2.0	ReCoRD	XSum
MNLI	**0.249**	0.131	0.175	0.109	0.139
QQP	0.145	**0.177**	0.135	0.103	0.126
SQuAD2.0	0.202	0.143	**0.286**	0.190	0.202
ReCoRD	0.167	0.119	0.219	**0.224**	0.195
XSum	0.164	0.128	0.237	0.188	**0.301**

Table 4: Average prompt similarity ($\mathcal{S}(\mathcal{T}_j^P, \mathcal{T}_k^F)$) among different tasks. The highest score in each row is **highlighted**.

Selection Method	Performance	Relative FLOPs
Layer Dropping		
Uniform	**70.92**	72%
Last	69.91	
FFN Reduction		
Activation	**71.50**	84%
Random	66.80	

Table 5: Average performance on 5 investigated tasks using different strategies of layer dropping and FFN reduction on T5$_{\text{LARGE}}$.

From the results in Table 4, we observe that the highest similarity is achieved when $j = k$, showing that the prompts of the partial PLMs are closer to the same task's prompts of the full-size model. This phenomenon is aligned with the observation in Figure 3, implying that on the same task, the soft prompts learned by partial PLMs could be potentially transferred to the full-size PLM.

E Effect of Partial Model Construction Designs for FPT

We construct a partial PLM by dropping a few layers or masking some neurons. As mentioned in § 2.2, for layer dropping, we retain the layers uniformly; for FFN reduction, we mask the neurons that are less likely to be activated. How to select the retained parameters is essential to the performance of FPT. To demonstrate this, in Table 5, we experiment FPT with another strategy for layer dropping and FFN reduction, respectively.

For layer dropping, we compare our strategy of dropping layers uniformly (denoted as *Uniform*) with dropping the last few layers (denoted as *Last*). For both methods, we retain the same number of layers. For example, in order to select 3 layers from a 24-layer PLM, the *Uniform* strategy will retain the layer $\{1, 12, 24\}$, and the *Last* strategy will retain the layer $\{1, 2, 3\}$. From Table 5, we can derive that the *Uniform* strategy is slightly better than the *Last* strategy. We hypothesize the reason is that the overall functionalities of a PLM are uniformly distributed among different layers, and adjacent layers tend to have similar functionalities. Therefore, retaining layers uniformly tends to reserve more functionalities than only retaining the first few layers.

For FFN reduction, we compare our strategy of masking neurons based on the activation score (denoted as *Activation*) with randomly masking neurons (denoted as *Random*). For the *Activation* strategy, we feed 1000 samples prepended by randomly initialized soft prompts into the PLM, and then record the activation score of neurons along each dimension. The results in Table 5 show that the *Activation* strategy significantly outperforms the *Random* strategy, demonstrating the effectiveness of our method. Randomly masking neurons may abandon those highly activated (most informative) ones, which hinders PT's convergence. We also find empirically the activation score of each neuron in FFN layer may vary a lot across different tasks, which means different neurons may respond differently to the input. This phenomenon also implies that there may exist some "functional partitions" in the FFN layers of PLMs.

F Effect of Duration for Each Training Stage

To show the effects of the duration of each training stage, following Gong et al. (2019), we conduct experiments on MNLI using T5$_{\text{LARGE}}$ with three proposed variants of FPT, and evaluate the effects of training duration for the last two stages.

Specifically, for layer dropping of FPT, we conduct PT on the 18-layer partial PLM for 15k steps, and save the learned soft prompts every 3k steps to get $15/3 = 5$ sets of soft prompts. Then using each of these 5 soft prompts as the initialization, we conduct PT with the full-size PLM for 3k steps. We report the validation performance and compare

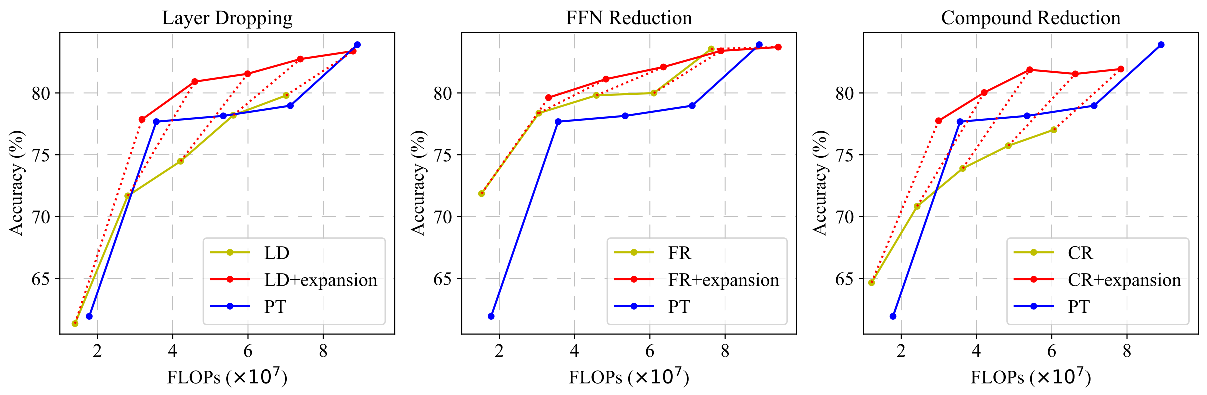

Figure 4: The validation performance on MNLI with different training duration for the last two stages. We conduct this ablation study for each of the three variants of FPT. We compare our FPT with different expanding time (**red** line) with vanilla PT (**blue** line) and PT without model expansion (**yellow** line). Each red dot is connected with a yellow dot by a dashed line, indicating it is initialized by the yellow dot and optimized by conducting PT on full-size PLM.

FPT with vanilla PT. For FFN reduction and compound reduction of FPT, we conduct similar experiments except that we start from a partial PLM using different construction methods.

The results are shown in Figure 4, from which we can see that expanding the partial PLM's size and then conducting PT (i.e., the red line) performs better than only conducting PT on the partial PLM (i.e., the yellow line). In addition, comparing our FPT (i.e., the red line) with vanilla PT (i.e., the blue line), there is a specific threshold s' of training steps, if we expand the partial PLM before s', the training efficiency can be improved compared with vanilla PT; however, after s', expanding the partial PLM and continuing PT on it does not bring consistent improvement over vanilla PT. In general, expanding the partial PLM between 3k steps and 12k steps works well for all three variants of FPT, indicating that within a reasonably broad range, the performance improvement of FPT is not sensitive to the duration of each training stage. We aim to explore how to decide the optimal duration for each training stage in future to make our FPT more practical.