PLEX: Adaptive Parameter-Efficient Fine-Tuning for Code LLMs using Lottery-Tickets

Jaeseong Lee[1*] **Hojae Han**[1*] **Jongyoon Kim**[2] **Seung-won Hwang**[12†]
Naun Kang[3] **Kyungjun An**[3] **Sungho Jang**[3]
{[1]CSE, [2]IPAI} Seoul National University
[3]Samsung SDS
{tbvj5914,stovecat,john.jongyoon.kim,seungwonh}@snu.ac.kr
{naun.kang,kyungjun.an,sh119.jang}@samsung.com

Abstract

Fine-tuning large language models (LLMs) for code generation is challenging due to computational costs and the underrepresentation of some programming languages (PLs) in pre-training. We propose PLEX, a lottery-ticket based parameter-efficient fine-tuning (PEFT) method that adapts LLMs to either well-supported and underrepresented PLs. During lottery ticket selection, PLEX employs a dual strategy: for well-represented PLs, it leverages the LLM's full parametric knowledge by selecting from full layers, while for underrepresented PLs, it narrows the selection scope to dense layers, prioritizing the most influential parameters. Additionally, PLEX-E, a low-rank extension of PLEX, further reduces computational costs by limiting the scope of fine-tuning. On MultiPL-E benchmarks, PLEX achieves state-of-the-art performance among PEFT methods, while PLEX-E maintains competitive results with reduced computational overhead. Both variants demonstrate effective adaptation across diverse programming languages, particularly for those underrepresented in pre-training.

1 Introduction

Code generation is a critical task in software development, and large language models (LLMs) have shown great promise in this domain (Chen et al., 2021; Allal et al., 2023).

In industrial settings, serving code LLMs often requires optimized generation for a target PL. Moreover, the target language can be proprietary, such as those used in chip design, which are typically absent from pretraining corpora.

Fine-tuning LLMs for specific PLs faces a computational bottleneck: While scaling laws suggest that larger models yield better performance, fully adapting the model for each target PL is prohibitively resource-intensive. This underscores

*Equal contribution
† Corresponding author.

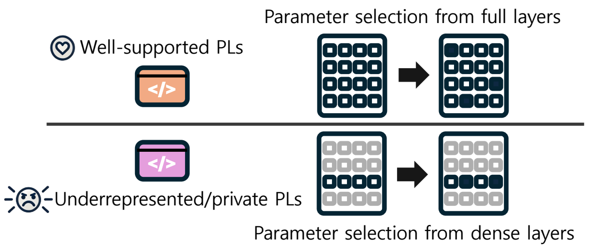

Parameter selection from full layers
Well-supported PLs
</>
</>
Underrepresented/private PLs
Parameter selection from dense layers

Figure 1: PLEX adaptively finetunes code LLMs by selectively updating parameters. For well-supported PLs, it uses a full parameter space, while for underrepresented or private PLs, it focuses on dense layers.

the importance of parameter-efficient fine-tuning (PEFT; Hu et al. 2022; Ansell et al. 2022), which addresses this issue by adapting only a subset of parameters, making fine-tuning more feasible and efficient.

An additional challenge arises when the target PL is underrepresented. We specifically use the term 'underrepresented', distinct from 'low-resourced', as low-resourced is defined with respect to an amount of public training resources per language. Meanwhile, underrepresentation is defined with respect to a specific PL-model pair. For instance, one language that is abundantly observed in pretraining one model, can be underrepresented in another.

Needs for adapting to an underrepresented PL are common in industrial setting, when supporting a rare PL or a private or proprietary language unavailable during pre-training. However, in our preliminary experiments, we observe that no existing PEFT method is one-size-fits-all for both well-supported and underrepresented PLs.

To address these issues, we propose PLEX, a novel PEFT method designed to efficiently adapt LLMs to both well-supported major PLs and underrepresented or private PLs. Our method employs an adaptive parameter selection using lottery-ticket (Ansell et al., 2022; Frankle and Carbin, 2019; Chen et al., 2020), adjusting the parameter groups based on whether the target PL is supported by the LLM. For well-supported PLs, PLEX leverages the full parameter space for ticket selection, maximizing the use of the model's pretrained knowledge. For underrepresented or private PLs, PLEX narrows the focus to dense layers, ensuring that the most influential parameters are prioritized during fine-tuning (Meng et al., 2022).

To further improve computational efficiency, we introduce PLEX-E, an extension of PLEX that replaces full fine-tuning with low-rank LoRA tuning (Hu et al., 2022) for parameter selection. This reduces the computational burden while minimizing performance drops, making it feasible to apply to larger models.

Our experimental results on the MultiPL-E HumanEval and MBPP benchmarks (Cassano et al., 2023) demonstrate the effectiveness of PLEX. The method performs well not only on well-supported languages like Java, PHP, C++, and Swift for StarCoder-7B (Li et al., 2023) and Java for SantaCoder-1.1B (Allal et al., 2023), but also on underrepresented languages like PHP and C++ for SantaCoder-1.1B. These results validate the adaptability of PLEX across a diverse range of programming languages, including those underrepresented during pre-training. Additionally, in the StarCoder-7B experiments, PLEX-E, the computationally efficient version of PLEX, generally outperforms existing PEFT baselines while remaining competitive with PLEX. The code and dataset are publicly available.[1]

2 Related Work

2.1 Multilingual LLMs

LLMs trained on large programming-related corpora, such as GitHub or The Stack, inherently support code generation across diverse PLs. However, their performance often declines when focusing on a specific PL, due to the curse of multilinguality (Conneau et al., 2020).

This degradation is more pronounced for an underrepresented PL. An immediate solution is rebalancing the pretraining corpus by adding substantial data for the target PL. However, this incurs pretraining costs.

A widely deployed solution is finetuning a pretrained multilingual model specifically on the target PL (Chen et al., 2021; Nijkamp et al., 2023; Guo et al., 2024).

[1]https://github.com/thnkinbtfly/PLEX

Our distinction We question a widely adopted approach of training an LLM with a large number of PLs, or fine-tuning the whole LLM. Our distinction is employing parameter-efficient fine-tuning (PEFT) to adapt to a target PL. This is different from works utilizing PEFT to code LLMs (Zhuo et al., 2024) for a different purpose of adapting to other tasks, not PLs.

2.2 PEFT: Parameter-Efficient Fine-Tuning

To adapt pretrained LMs to specific PLs for code generation, PEFT methods, which aim to add a handful number of parameters for fine-tuning, were a popular solution. A common PEFT employed for code LMs was LoRA (Zhuo et al., 2024), which fine-tuned a low-rank subspace of each weight matrix. However, LoRA often missed important information outside of low-rank space (Chen et al., 2023).

A promising alternative was LT-SFT (Ansell et al., 2022), which aimed to find lottery tickets (Frankle and Carbin, 2019; Chen et al., 2020), which is a subnetwork whose performance is similar or better when fine-tuned. However, it required full fine-tuning, which was not practical for ever-enlarging LMs. Moreover, we noticed sometimes it underperforms LoRA significantly.

Our Distinction We find that both LoRA and LT-SFT are suboptimal for PL adaptation, and propose PLEX, a best practice for parameter-efficient adaptation to both under- and well-represented PL as a target PL. We observe when LT-SFT underperforms despite a higher cost, to aim at reducing such cases. Moreover, we devise PLEX-E, a computation-efficient version of PLEX, that is more suitable for large LMs.

3 Proposed Method

3.1 Both LoRA and LT-SFT are Suboptimal In PL Adaptation

Our first finding is that both LoRA and LT-SFT are suboptimal when adapting pretrained LMs to diverse PLs.

While LoRA is a popular PEFT method (Zhuo et al., 2024), it may miss important information outside of low-rank space (Yu et al., 2017). A promising alternative would be the lottery-ticket-based PEFT method, LT-SFT (Ansell et al., 2022).

Although we observe it usually outperforms LoRA, we observe sometimes it significantly underperforms LoRA (Table 1). To investigate why, we analyze the updated parameter distribution over layers, and find that the updated parameters in dense layers are too small (Figure 3 blue in subsubsection 4.3.2). This can be a problem, since most knowledge is believed to reside in dense layers (Meng et al., 2022).

3.2 Proposed: PLEX

To address this, we propose PLEX, which moves beyond the limitations of the low-rank assumption (Hu et al., 2022), known to overlook critical information (Yu et al., 2017). Instead, we focus on finding lottery tickets within the network and ensure the selected parameters reside in dense layers, where most knowledge is concentrated (Meng et al., 2022), overcoming the shortcomings of existing methods. Last but not least, for large language models, we make our version to be computation-efficient, avoiding the expensive full fine-tuning of LT-SFT.

One promising alternative of LoRA, LT-SFT (Ansell et al., 2022), the lottery-ticket-based PEFT method, first finds lottery-ticket– a subnetwork whose fine-tuned performance is comparable to fine-tuning the full model. Formally, given a neural function with pretrained weight $\theta \in \mathbb{R}^N$, finding a ticket corresponds to finding a mask $m \in \{0,1\}^N$, by choosing the parameter with the largest movement (Sanh et al., 2020) after fully fine-tuning the model (Ansell et al., 2022). Then it restricts the training updates to be $\Delta\theta \odot m$, converting the given parameters as follows:

$$f_{LT}(\theta) = \theta + \Delta\theta \odot m \tag{1}$$

where $\odot$ is element-wise multiplication. $\epsilon = \frac{\|m\|_0}{N}$ is naturally referred to as the density, tuned as a hyper-parameter, but expected to be $\ll 1$ for sparsity, where $\|\cdot\|_0$ counts the number of non-zero values.

However, LT-SFT underperforms in underrepresented PLs, likely due to the low proportion of parameters in dense layers (Figure 3 blue in subsubsection 4.3.2). Therefore, PLEX focuses the updates to dense layers only, for efficient adaptation to PLs. Formally, we update the given parameter as follows:

$$f_{PLEX}(\theta) = \theta + \Delta\theta \odot m \odot (1 - \mathbb{1}(l \in \text{UR})m_D) \tag{2}$$

where $\mathbb{1}(l \in \text{UR})$ is an indicator function conditioning whether given PL l is underrepresented[2] or not, and $m_D \in \{0,1\}^N$ is 1 where the index does not correspond to any dense layer. Here, density is defined as $d = \frac{\|m \odot (1-\mathbb{1}(l \in \text{UR})m_{\overline{D}})\|_0}{N}$.

3.3 PLEX-E: Computation-Efficient Variant for Large LMs

The proposed PLEX could overcome the downside of LT-SFT, but it would not be practical to apply to large LMs, since it requires fully fine-tuning the dense layers.

Inspired by LoRA, given a dense layer weight $W \in \mathbb{R}^{a \times b}$, instead of directly optimizing the weight, we reduce the computational cost by applying low-rank updates as follows:

$$\Delta W = W_u W_d \tag{3}$$

where $W_u \in \mathbb{R}^{a \times r}, W_d \in \mathbb{R}^{r \times b}$ are the optimization target. The computational cost is controlled by reducing r.

4 Experiments

In this section, our goal is answering to the following research questions:

- RQ1: How do existing PEFT methods (LoRA, LT-SFT) exhibit complementary strengths and weaknesses across different PLs?
- RQ2: Can we design a PEFT method that combines the advantages of both LoRA and LT-SFT while mitigating their limitations?
- RQ3: How can we maintain the benefits of our approach while achieving computational efficiency for LLMs?

4.1 Experimental Setup

Model Selection We evaluate the effectiveness of PLEX on code generation with pretrained LMs. We strategically select SantaCoder-1.1B (Allal et al., 2023)[4] for our main experiments due to its focused pretraining on only three PLs (Python, Java, and JavaScript). This focused pretraining provides a controlled setting for simulating underrepresented scenarios with PLs absent from pretraining data (see Figure 2). To investigate the scalability and

[2] $l \in \text{UR}$ can be empirically decided based on zero-shot performance (Section 4).

[3] https://huggingface.co/datasets/bigcode/the-stack

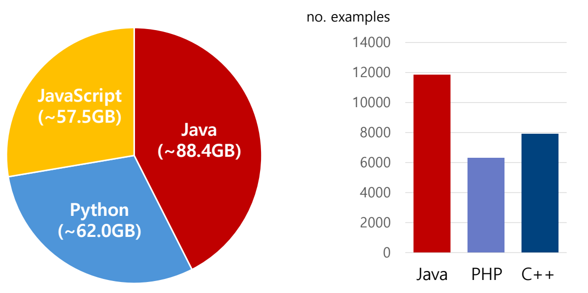
[4] https://huggingface.co/bigcode/gpt_bigcode-santacoder

Method	Δ param. size (↓)	Java $\mathcal{H}$	Java $\mathcal{M}$	PHP $\mathcal{H}$	PHP $\mathcal{M}$	C++ $\mathcal{H}$	C++ $\mathcal{M}$	Swift $\mathcal{H}$	Swift $\mathcal{M}$
No tune	0GB	15.0	28.1	1.5	3.1	6.2	15.7	0.7	3.0
Full FT	4.2GB	17.9	26.4	11.3	17.5	10.4	22.0	$\underline{7.1}$	$\underline{13.9}$
LoRA	154-205MB	16.7	28.6	**11.1**	16.0	9.4	20.3	2.1	7.3
SoRA	158-199MB	17.8	24.0	10.0	**19.1**	$\underline{11.4}$	$\underline{21.1}$	**7.4**	**17.1**
LT-SFT	130-194MB	**18.2**	**29.7**	3.4	7.0	8.2	18.3	4.8	12.4
PLEX	130-194MB	**18.2**	**29.7**	$\underline{10.1}$	$\underline{17.1}$	**12.3**	**21.3**	4.5	12.1

Table 1: **SantaCoder-1.1B Pass@1 scores for various PEFT methods on the HumanEval ($\mathcal{H}$) and MBPP ($\mathcal{M}$) benchmarks in MultiPL-E.** Δ param. denotes the size of trainable parameters. Java is included in the pretraining corpus, while PHP, C++ and Swift (gray highlighted) are not. Best scores are in bold; second-best are underlined.

Method	Computational Efficiency	Java $\mathcal{H}$	Java $\mathcal{M}$	PHP $\mathcal{H}$	PHP $\mathcal{M}$	C++ $\mathcal{H}$	C++ $\mathcal{M}$	Swift $\mathcal{H}$	Swift $\mathcal{M}$
No tune		24.4	37.7	22.1	35.1	23.3	42.0	15.1	30.1
LoRA	$r \times (M + N)$	28.5	38.7	$\underline{29.2}$	41.7	$\underline{26.0}$	38.7	20.0	32.7
SoRA	$r \times (M + N)$	**30.5**	39.5	28.3	$\underline{44.5}$	22.6	38.3	$\underline{21.5}$	33.0
P-Tuning	l	27.3	36.2	0.0	0.0	0.6	0.0	18.1	28.2
PLEX-E	$r \times (M + N)$	$\underline{29.1}$	$\underline{39.7}$	28.4	44.2	25.6	$\underline{40.2}$	20.9	**34.2**
PLEX	$M \times N$	29.0	**40.4**	**29.4**	**45.3**	**26.7**	**40.6**	**22.3**	$\underline{33.9}$

Table 2: **StarCoder-7B Pass@1 scores of various PEFT methods across diverse PLs on the HumanEval ($\mathcal{H}$) and MBPP ($\mathcal{M}$) benchmarks in MultiPL-E.** PLEX-E is a computationally efficient version of PLEX. LT-SFT is omitted as it is equivalent to PLEX since all Java, PHP, C++, and Swift are well-supported PLs for StarCoder-7B. Best scores are in bold; second-best are underlined.



(a) Pretraining Corpus. (b) Finetuning set (MegaCode-Training).

Figure 2: SantaCoder 1.1B was pre-trained on the Java, Python, and JavaScript subset of the Stack-v1.1.[3] Accordingly, Java is a well-supported programming language, whereas PHP and C++ are underrepresented.

computational efficiency of PLEX, particularly PLEX and PLEX-E, we extend our evaluation to StarCoder-7B (Li et al., 2023).[5]

Evaluation Metrics Consistent with existing works (Chen et al., 2021; Li et al., 2022), we use Pass@k $:= \mathbb{E}_{\text{Problems}}[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}]$ (Chen et al., 2021) as the main metric to evaluate the code generation abilities of pretrained LMs. Note that, for an unbiased evaluation, Pass@k calculates the average probability of selecting at least one of c correct code snippets from every combination of k samples chosen from n given samples.

Datasets and Languages for Evaluation We evaluate PLEX on MultiPL-E (Cassano et al., 2022), which expands the Python-only benchmarks HumanEval ($\mathcal{H}$) and MBPP ($\mathcal{M}$) to support diverse PLs.

For adaptation to specific PL, we utilize MegaCodeTraining corpus,[6] filtered to contain the specific PL of our target. As PEFT performance is generally bounded by full finetuning performance, we first verify that full finetuning shows clear improvements over the pretrained model for each candidate PL. This ensures our evaluation meaningfully assesses PEFT effectiveness rather than dataset limitations. Based on these criteria, we use Java, C++, PHP, and Swift.

Baselines We compare PLEX with the following approaches: 1) *LoRA* (Hu et al., 2022): a popu-

[5]https://huggingface.co/bigcode/starcoderbase-7b

[6]huggingface.co/datasets/rombodawg/MegaCodeTraining

lar parameter-efficient fine-tuning method, which assumes a low-rank update of parameters. We focus on the Q,K,V attention matrices.[7] 2) *LT-SFT* (Ansell et al., 2022): an alternative parameter-efficient fine-tuning method, which is supported by the lottery-ticket hypothesis. 3) *SoRA* (Ding et al., 2023): an efficient variant of LoRA which reduces the rank adaptively. 4) *P-Tuning* (Liu et al., 2021): prepending trainable prefix vectors to inputs.

Implementation Details For RQ1-2, we use ϵ=3%,[8] batch size of 8, learning rate of 2e-5, and train for 3 epochs. For LoRA, we use batch size of 8, learning rate of 5e-5, train for 3 epochs. Specifically, to use the comparable number of PEFT parameters, for LoRA, we set r=α=768 for Swift, and r=α=1024 for other PLs. For SoRA, the training setting is similar to LoRA, while we set learning rate as 1.5e-4, r=α=128 for SoRA on C++ and PHP, r=α=192 for SoRA on Java, and r=α=96 for SoRA on Swift.[9] For RQ3, the hyperparameters are mostly similar. We use ϵ=1%, and r=α=420 for LoRA. For SoRA, we use r=α=96 for SoRA on C++, PHP, Swift, and r=α=160 for SoRA on Java. We use r=α=1024 for PLEX. We use l=256 for P-tuning. We generate 200 samples per problem, with temperature 0.2, and max length of 650. We regard a PL as underrepresented if the average Pass@1 performance without any tuning is under 15%.

4.2 Results

4.2.1 RQ1: Both LoRA and LT-SFT are Suboptimal for PL Adaptation

The rows for LoRA (or SoRA) and LT-SFT in Table 1 highlight their suboptimal performance in adapting pretrained LMs to diverse PLs. For instance, when adapting to Java, LoRA's Pass@1 is 1.5%p lower than LT-SFT on Humaneval ($\mathcal{H}$). Conversely, for out-of-domain PLs like C++ or PHP, LT-SFT's Pass@1 is up to 9%p lower than LoRA on MBPP ($\mathcal{M}$).[10] In Section 4.3.2, we analyze why LT-SFT struggles in these scenarios.

4.2.2 RQ2: PLEX, the Best Practice

Overall, PLEX outperforms all the baselines including LoRA or LT-SFT. For instance, even in C++ or PHP adaptation, where LT-SFT fails, Pass@1 increases by up to 10.1%p in MBPP ($\mathcal{M}$) compared with LT-SFT. PLEX even outperforms the recently proposed LoRA variant (SoRA). The score of PLEX is up to 5.7% higher in Java MBPP ($\mathcal{M}$) compared to SoRA. In overall, PLEX usually outperforms SoRA in the benchmark (wins 5/8 times), outperforms LT-SFT (wins 6/8 times), and LoRA (wins 7/8 times).

4.2.3 RQ3: The Computation-Efficient Version, PLEX-E

Table 2 shows that PLEX-E outperforms other computation-efficient PEFT methods, such as SoRA (wins 5/8 times), LoRA (wins 6/8 times), and P-Tuning (wins 8/8 times). Note we do not compare with LT-SFT, which is computationally inefficient.[11]

4.3 Analyses

4.3.1 Efficiency Analysis of PEFTs

We analyze the relative computational cost of comparisons (Table 2 2nd column). PLEX-E requires $\min(r \times (M+N), 3\epsilon MN)$, which reduces to $r \times (M+N)$ if $\epsilon < r\frac{M+N}{3MN}$, the similar computational cost to LoRA, when given the target dense layer $W \in \mathbb{R}^{M\times N}$, the rank of LoRA $r < \min(M,N)$, the density $\epsilon \ll 1$. Note that P-Tuning depends on different dimension l, the length of trainable prompt, but we omit empirical comparison due to its suboptimal performance.

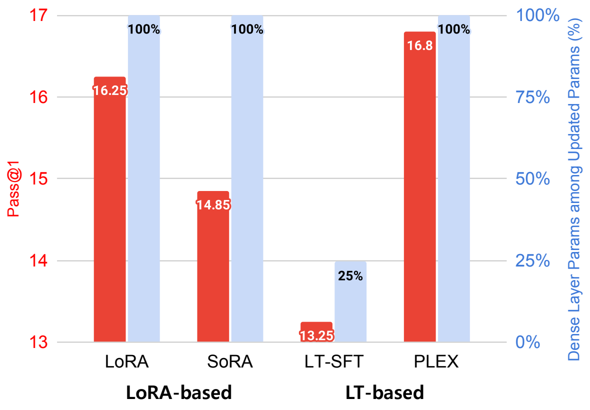


Figure 3: SantaCoder-1.1B Pass@1 scores on a failure case (C++) of LT-SFT. We report the averaged score of HumanEval and MBPP (red), along with the ratio of dense layer parameters in the updated parameters (blue).

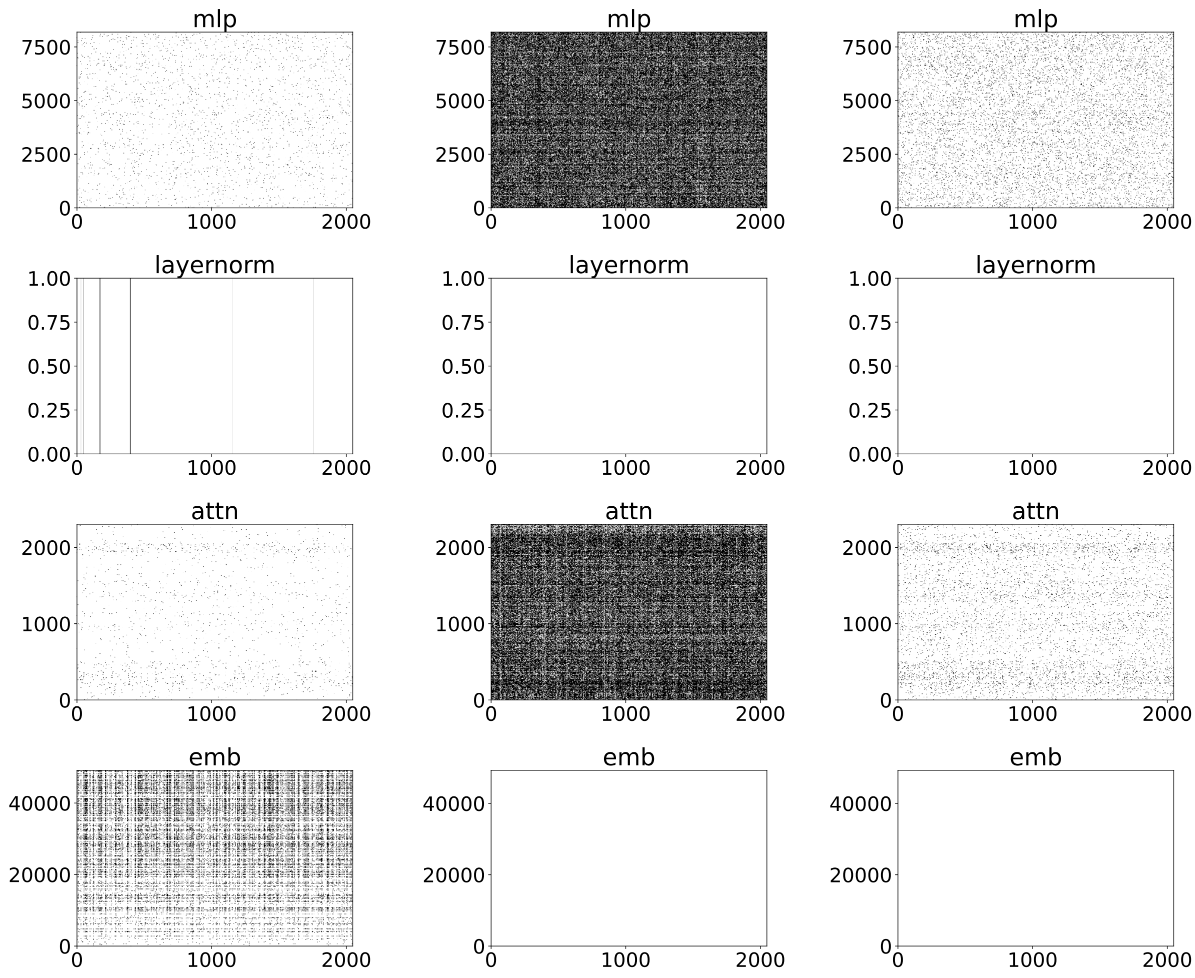
[7]Adding attention output matrix or feed forward networks as the target underperformed this base setting.

[8]We selected among 1%, 3%, 10%, based on Java Pass@1 performance.

[9]They scale learning rate about 3x than LoRA, and use all the dense layers as their target.

[10]We consider Swift on SantaCoder as an outlier, which tends to underperform with any PEFT methods.

[11]Refer to Appendix A for the application of PLEX-E in SantaCoder-1.1B.



(a) LT-SFT (b) SoRA (c) PLEX

Figure 4: Heatmap of affected parameters of layers in LT-SFT, LoRA-variant (SoRA), and PLEX. We investigate a SantaCoder-1.1B case study on C++, where LoRA outperforms LT-SFT. The analysis covers the embedding layer (*emb*), attention layer (*attn*), layernorm layer (*layernorm*), and multilayer perceptron (*mlp*).

4.3.2 Visualization of Our Distinction

In this section, we visually analyze why PLEX is superior to LT-SFT or LoRA. Specifically, we investigate the case when LT-SFT underperforms, such as in C++ where it falls short of LoRA (see Table 1).

First, LT-SFT updates too few parameters in the dense layers, where most of the knowledge resides (Meng et al., 2022). To delve deeper, we examine the heatmap of affected parameters across layers—embedding, attention (attn), layernorm, and MLP— on the 22nd layer for comparison.

Figure 4 illustrates that LT-SFT (Figure 4a) updates fewer parameters in the attention and MLP layers (where dense layers are concentrated) and instead updates other layers, like the embedding layer. In contrast, PLEX (Figure 4c) prioritizes updates in dense layers, effectively targeting the knowledge stored in the language model (Meng et al., 2022).

Second, LoRA-variants densely affect the parameters, but they require low-rank assumption to do it in a parameter-efficient way, which is known to overlook critical information (Yu et al., 2017). In contrast, PLEX is free of low-rank assumption, by achieving parameter-efficiency with the lottery-ticket hypothesis.

5 Conclusion

We studied an adaptive PEFT method using lottery tickets. We propose PLEX, which effectively adapts PEFT to any PL, whether well- or under-represented. We also introduce PLEX-E, a computation-efficient version of PLEX, which reduces the full fine-tuning cost during ticket selection, making our method applicable to large LMs.

Acknowledgments

This work was partly supported by Samsung SDS. This work was also partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) [NO.RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)]

References

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. 2023. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable Sparse Fine-Tuning for Cross-Lingual Transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796, Dublin, Ireland. Association for Computational Linguistics.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2022. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2023. Multipl-e: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

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*, volume 33, pages 15834–15846. Curran Associates, Inc.

Xuxi Chen, Tianlong Chen, Weizhu Chen, Ahmed Hassan Awadallah, Zhangyang Wang, and Yu Cheng. 2023. DSEE: Dually Sparsity-embedded Efficient Tuning of Pre-trained Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8208–8222, Toronto, Canada. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse Low-rank Adaptation of Pre-trained Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4133–4145, Singapore. Association for Computational Linguistics.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. 2023. StarCoder: May the source be with you! *Transactions on Machine Learning Research*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT Understands, Too.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *International Conference on Learning Representations*.

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33, pages 20378–20389. Curran Associates, Inc.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7370–7379.

Terry Yue Zhuo, Armel Zebaze, Nitchakarn Suppattarachai, Leandro von Werra, Harm de Vries, Qian Liu, and Niklas Muennighoff. 2024. Astraios: Parameter-Efficient Instruction Tuning Code Large Language Models.

A PLEX-E on SantaCoder-1.1B

Table 3 shows that PLEX-E shows comparable Pass@1 scores to LoRA and SoRA when applied to SantaCoder-1.1B. Note that PLEX-E still significantly improves in adapting to underrepresented PLs (PHP and C++) over the existing lottery-ticket based PEFT approach LT-SFT, though PLEX is computationally more efficient.

Method	Δ param. size (↓)	Java $\mathcal{H}$	Java $\mathcal{M}$	PHP $\mathcal{H}$	PHP $\mathcal{M}$	C++ $\mathcal{H}$	C++ $\mathcal{M}$	**Avg. (↑)**
Full FT	4.2GB	17.9	26.4	**11.3**	<u>17.5</u>	10.4	**22.0**	<u>17.6</u>
LoRA	205MB	16.7	28.6	<u>11.1</u>	16.0	9.4	20.3	17.0
SoRA	162-199MB	17.8	24.0	10.0	**19.1**	<u>11.4</u>	21.1	17.2
LT-SFT	194MB	**18.2**	**29.7**	3.4	7.0	8.2	18.3	14.1
PLEX-E	194MB	17.1	29.2	9.5	15.6	8.9	<u>21.4</u>	16.9
PLEX	194MB	**18.2**	**29.7**	10.1	17.1	**12.3**	21.3	**18.1**

Table 3: **SantaCoder-1.1B Pass@1 scores of various PEFT methods across diverse PLs on the HumanEval ($\mathcal{H}$) and MBPP ($\mathcal{M}$) benchmarks in MultiPL-E.** Δ param. signifies the size of trainable parameters. PLEX-E is a computationally efficient version of PLEX. Java is included in the pretraining corpus, while PHP and C++ (gray highlighted) are not. Best scores are in bold; second-best are underlined.