A Comparative Study of PEFT Methods for Python Code Generation

Johanna Männistö Joseph Attieh Jörg Tiedemann
Department of Digital Humanities, University of Helsinki
{first.last}@helsinki.fi

Abstract

Fine-tuning language models incurs high costs in training, inference and storage. Parameter-efficient fine-tuning (PEFT) methods have emerged as a more cost-effective alternative to full fine-tuning. However, limited work has compared different PEFT approaches for tasks like code generation. In this study, we examine the effect of various PEFT training methods on model performance in the task of Python code generation. We fine-tune four model families, ranging from 124M to 7B parameters, using three PEFT approaches alongside standard full fine-tuning. Our findings reveal that the effectiveness of each PEFT method varies with the model size and the corpus used.

1 Introduction

Language models (LMs) have shown great capabilities across a variety of natural language processing (NLP) downstream tasks, including code generation tasks (Chen et al., 2021; Li et al., 2023; Nijkamp et al., 2023; Rozière et al., 2023a; Xu et al., 2022). Generally, larger LMs tend to perform better on downstream tasks (Kaplan et al., 2020), as evidenced by CodeLlama, which exhibits improved code completion and generation abilities as its size increases from 7 billion to 70 billion parameters (Rozière et al., 2023a). However, the training of these larger models is resource-intensive, requiring substantial computational power and high storage costs.

To address these challenges, Parameter-Efficient Fine-Tuning (PEFT) methods have emerged (Dettmers et al., 2023; Houlsby et al., 2019; Hu et al., 2022; Lester et al., 2021; Lialin et al., 2023; Liu et al., 2022). These approaches update a small subset of the model parameters during fine-tuning, while the rest remain frozen, significantly reducing both computational and storage costs for each downstream task.

While significant research has been conducted on both PEFT methods and code LMs individually, at the time of this study, there is only limited research evaluating PEFT approaches applied to code LMs for code generation tasks (Purnawansyah et al., 2024; Weyssow et al., 2023; Zhuo et al., 2024). Existing studies on this topic have notable shortcomings: many focus only on smaller models, ignoring those with 1B parameters or more (Ayupov and Chirkova, 2022; Zou et al., 2023), while others concentrate solely on tasks like code understanding or clone detection, which often outperform code generation tasks under similar PEFT training conditions (Liu et al., 2023; Wang et al., 2023; Zou et al., 2023). These limitations highlight a significant research gap, particularly as state-of-the-art models increasingly feature billions of parameters and are predominantly generative.

We aim to fill existing research gaps through two key questions: 1) Which PEFT method delivers the best performance across various model sizes for Python generation tasks? 2) How do these methods compare to full fine-tuning?

2 Parameter Efficient Fine-Tuning

Parameter efficient fine-tuning (PEFT) methods provide a more efficient alternative to full fine-tuning of large LMs (LLMs), significantly reducing both computational and storage costs (Lialin et al., 2023). Various PEFT methods achieve remarkable performance compared to full fine-tuning for models of different sizes (Ding et al., 2023; Lester et al., 2021; Wang et al., 2023), all while offering substantial computational savings. We present an overview of the three methods that are relevant to our work. These methods are illustrated in Figure 1.

Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025), pages 390–396
March 3-4, 2025 ©2025 University of Tartu Library

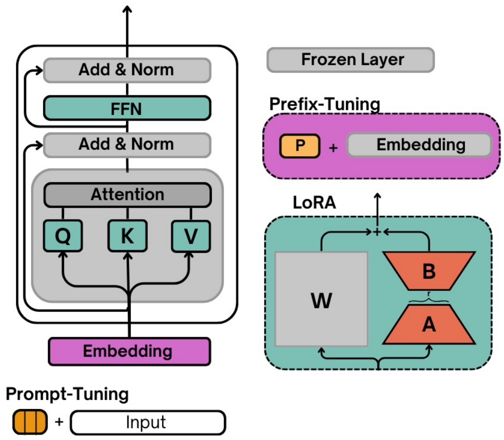

Figure 1: Diagram showing a decoder layer, as well the PEFT Techniques employed in the study.

Low Rank Adaptation (LoRA) LoRA (Hu et al., 2022) approximates model weight matrices through low-rank decomposition into a smaller set of parameters. The pretrained weights are frozen, and the approximation is fine-tuned during training. LoRA can be applied to any weight matrix, and Dettmers et al. (2023) shows that applying it to all linear layers enhances performance compared to limiting it to query and value matrices as done in Hu et al. (2022). The efficiency of LoRA is determined by the rank of the decomposed matrices and the scaling factor, alpha. Alpha is often set to be twice the size of the rank (Zhuo et al., 2024; Weyssow et al., 2023) or equivalent to the rank (Lee et al., 2023).

Prefix-tuning Inspired by in-context learning, this method (Li and Liang, 2021) prepends trainable tensors called ”soft prompts” to the input of each transformer block. These task-specific prefixes are updated during training while the original model parameters are frozen.

Prompt-tuning Similar to prefix-tuning, prompt-tuning (Lester et al., 2021) adds trainable parameters to the input layer only, leading to a further reduction in the number of parameters that need updating compared to prefix-tuning.

3 Experimental Approach

In this section, we describe the methodology used to investigate how models of different sizes adapt to the Python code generation task using PEFT.

The experimental approach is illustrated in Figure 2, which outlines the models, dataset, data processing methods, training setup, and evaluation strategy. These elements will be described in detail in the following section.

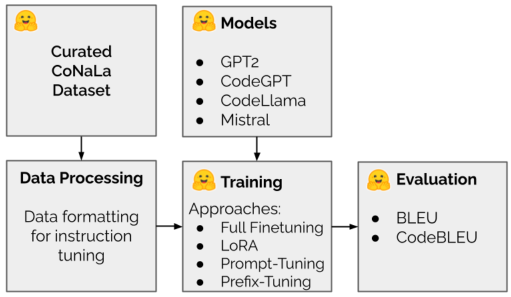

Figure 2: Diagram describing the experimental approach adopted in this study.

3.1 Models

In this study, we strategically select four distinct model families, mainly GPT-2, CodeGPT, CodeLlama, and Mistral v0.1[1].

We selected the models with sizes ranging from 124M to 7B parameters and trained them on either text, code, or both. This enable us to explore model sizes that have been overlooked in similar studies.

GPT-2 (Radford et al., 2019) Autoregressive models ranging from 124M to 1.5B. The study employs GPT-2, GPT-2 M, L, and XL.

CodeGPT (Lu et al., 2021) is initialized from GPT-2 and fine-tuned on code corpora. The study focuses on the Python variants of the models, using both adapted and small versions[2].

CodeLlama (Rozière et al., 2023b) Available in three sizes (7B, 13B, and 34B) and three variants. Only the 7 billion parameter base model was fine-tuned for this study.

Mistral Mistral v.01 (Jiang et al., 2023) A 7B autoregressive model trained on open-source text and code data, with no training datasets listed. At the time of this study, Mistral did not support prefix-tuning.

[1]This model was the latest release at the time of the study. It was selected as it is trained on both text and code.

[2]The adapted is trained using the same tokenizer as GPT-2 and the small uses another newly trained BPE tokenizer.

3.2 Datasets

The study utilizes the CoNaLa dataset (Yin et al., 2018), consisting of 2,379 natural language-code pairs for training and 500 pairs for testing. This dataset is derived from the larger CoNaLa-mined dataset, initially sourced from Stack Overflow. For training, we use the `rewritten_intent` field, which contains the natural language instruction (i.e., Python problem), and the `snippet` field, which provides the corresponding Python code solution. As the dataset was already curated for quality by annotators, no additional filtering was conducted prior to training.

We formatted the data for model input by adding indicator prompts `### Instruction:` before the `rewritten_intent` and `### Response:` before the `snippet`, followed by a newline separator[3]. An example from the processed dataset can be seen in Table 1.

Rewritten Intent	### Instruction: How can I send a signal from a Python program?
Snippet	### Response: os.kill(os.getpid(), signal.SIGUSR1)

Table 1: Example from the CoNaLa dataset showing the structure of processed training data.

3.3 Training Setup

The implementation relies on the following libraries: HuggingFace transformers (Wolf et al., 2020), TRL (Werra et al., 2020) and PEFT (Mangrulkar et al., 2022). We perform the training using HuggingFace's SFTTrainer. The training arguments were selected to be the same as the reported hyperparameters for each model whenever feasible; otherwise, we pick hyperparameters and empirically validate them to ensure a reliable baseline for our experiments.

The models were given packed[4] input sequences of length 1024, which included any additional prefix or prompt tokens when needed, and were separated by an EOS (end-of-sequence) token. This value was selected due to GPU memory limitations. As done by Shi et al. (2024), we include the entire instruction-response set in the loss calculation rather than masking the instructions, as this approach can enhance performance with smaller datasets.

We apply LoRA to all linear layers of the model following Dettmers et al. (2023), and we set the rank to 16 and alpha to 32. Experiments by Lester et al. (2021) on prompt length demonstrated that only marginal gains were achieved when prompts exceeded 20 tokens, motivating the use of just 20 tokens for prompt-tuning and prefix-tuning.

3.4 Evaluation

We evaluate the models on the CoNaLa dataset using BLEU-4 (Papineni et al., 2002) and CodeBLEU (Ren et al., 2020). For both metrics, 1.0 is the highest score. To generate the predictions, we use a temperature of 0.2 and nucleus sampling (Holtzman et al., 2020) with top p = 0.95. All models are loaded using BF16 for inference.

4 Discussion

Table 2 summarizes the BLEU and CodeBLEU[5] scores of the different models on the CoNaLa dataset.

Best PEFT Approach We observe that smaller models tend to achieve higher CodeBLEU scores when utilizing prompt-based techniques, while larger models show improved performance with LoRA. Prompt-tuning, which tunes the fewest parameters, demonstrates enhanced effectiveness as model size increases, consistent with the findings of Lester et al. (2021). In terms of BLEU scores, LoRA consistently outperforms other PEFT techniques. It seems that LoRA tries to learn the exact n-gram matches from the Python solution, succeeding to do so for larger models. Conversely, prefix-tuning appears to degrade performance across all models, aligning with the results reported by Zou et al. (2023).

Full Fine-tuning versus PEFT Table 3 displays the number of parameters trained for each PEFT method across the models in addition to the peak GPU memory consumption, reported by HuggingFace's Trainer. Full fine-tuning often outperforms PEFT methods. Although PEFT approaches offer greater efficiency, they still effectively compete with full fine-tuning despite the significant reduction in trained parameters. Additionally, memory savings from utilizing PEFT methods increase as

[3] This structure follows the Stanford Alpaca.

[4] Packed input sentences combine multiple sequences into a single one separated by end-of-sequence token, to maximize training efficiency.

[5] Unlike BLEU, CodeBLEU captures semantically equivalent code snippets that may differ in syntax.

	BLEU				CodeBLEU			
	FT	**LoRA**	**Prefix**	**Prompt**	**FT**	**LoRA**	**Prefix**	**Prompt**
GPT2	**0.06025**	*0.05043*	0.00035	0	*0.113*	0.09006	**0.25**	0
CodeGPT-Small	**0.12152**	*0.04647*	0.00093	0.00328	*0.1096*	0.08588	**0.13349**	0.08974
CodeGPT-Adapt	**0.20204**	*0.05877*	0.00050	0.00470	**0.14476**	*0.14085*	0.07047	0.13243
GPT2-M	**0.17327**	*0.06364*	0	0	*0.16641*	0.10781	0	**0.25**
GPT2-L	**0.24957**	*0.12984*	0.04929	0.03104	**0.18253**	*0.17777*	0.13185	0.13504
GPT2-XL	**0.27059**	*0.221*	0.00340	0.02419	*0.1771*	**0.18665**	0.0296	0.12399
CodeLlama	**0.44735**	*0.43625*	0.0001	0.33996	**0.29512**	*0.27793*	0.13267	0.20798
Mistral	0.00019	**0.43533**	0	*0.39626*	0.25132	**0.29378**	0	*0.25466*

Table 2: Performance comparison of models using BLEU and CodeBLEU metrics. Scores highlighted in bold and italic represent the maximum and second-highest scores for each metric per row, respectively. Rows shaded in gray indicate models that are pre-trained on code data.

model size grows. Unexpectedly, Mistral experienced a significant decline in BLEU after fine-tuning, but not on CodeBLEU. This indicates that fine-tuning impacted Mistral's ability to generate exact n-gram matches with the reference, but did not compromise its performance in code-related tasks, highlighting a key distinction between these evaluation metrics.

Model	**# Par.**	**Method**	**% Par. Trained**	**Avg. GPU Use (GB)**
GPT-2 CodeGPT-Small CodeGPT-Adapted	124M	FT LoRA Prefix Prompt	100.00% 0.47% 0.30% 0.01%	7.15 6.91 5.85 6.17
GPT2-M	355M	FT LoRA Prefix Prompt	100.00% 2.99% 0.28% 0.01%	17.19 16.49 13.59 14.37
GPT2-L	774M	FT LoRA Prefix Prompt	100.00% 1.50% 0.003% 1.50%	31.56 29.82 24.33 25.83
GPT2-XL	1.6B	FT LoRA Prefix Prompt	100.00% 1.25% 0.20% 0.002%	52.85 48.94 39.72 42.19
CodeLlama	6.7B	FT LoRA Prefix Prompt	100.00% 0.59% 0.08% 0.001%	50.23 37.33 22.91 23.39
Mistral	7.2B	FT LoRA Prompt	100.00% 0.58% 0.0011%	54.98 41.80 26.52

Table 3: Percentage of Parameters Trained and Average GPU Use Across Model Families and Training Methods.

Code vs No-code models We compare GPT-2 to the CodeGPT models, as they share the same architecture. Fine-tuning consistently leads to the best BLEU performance, with CodeGPT-Adapt achieving the top BLEU and CodeBLEU scores, indicating the effectiveness of fine-tuning when a model is pretrained on code (without adapting the tokenizer). In addition, prefix-tuning on GPT-2 achieves the highest CodeBLEU scores. This motivates further use of such PEFT methods on general-purpose models, like GPT-2, where prefix-tuning can achieve competitive or even superior performance without the need for extensive fine-tuning. Interestingly, Code-Llama and Mistral, pretrained on both code and text, achieve the best overall performance when paired with LoRA, highlighting that large models pretrained on both types of data combined with efficient PEFT methods offer strong performance gains, especially for computationally efficient code generation.

5 Related Work

Most research combining software engineering tasks with PEFT methods has focused on small models (under 1B parameters), often comparing only a few PEFT techniques or excluding code generation tasks. Ayupov and Chirkova (2022) evaluated LoRA (Hu et al., 2022) and Adapters (Houlsby et al., 2019) on PLBART (Ahmad et al., 2021) and CodeT5 (Wang et al., 2021), finding that for complex tasks like code generation, these PEFT methods underperformed compared to full fine-tuning. Wang et al. (2023) showed that PEFT approaches can mitigate catastrophic forgetting in code summarization and search tasks but did not explore code generation. Recent studies have begun to address larger models. Weyssow et al. (2023) trained models up to 7B parameters using PEFT techniques and full fine-tuning, finding that LoRA provides improvements over in-context learning while offering significant memory savings compared to full fine-tuning. Zhuo et al.

(2024) instruction-tuned 28 models ranging from 1B to 16B parameters across 7 different methods for code generation tasks, concluding that while full fine-tuning generally yields the best performance, LoRA can achieve comparable results.

6 Conclusion

We investigated the effect of PEFT approaches on code generation tasks by training four model families with four fine-tuning methods on the curated CoNaLa dataset. Our findings suggest that LoRA is an efficient and effective PEFT method, one which rivals full fine-tuning once the model size is sufficiently large. Notably, smaller models excel with prompt-based techniques, achieving higher CodeBLEU scores, while larger models benefits more from LoRA, which focuses on fitting the exact n-gram matches from the reference code. This dual performance is reflected in the differing results of BLEU and CodeBLEU, giving us insights in how these technique work. Overall, techniques like LoRA and prompt-tuning are promising for enhancing efficiency and maintaining performance in code generation tasks, particularly in models pretrained on both code and text.

Limitations

We acknowledge several limitations of this work. Firstly, no hyperparameter search has been conducted on the PEFT approaches. Many studies (Zhuo et al., 2024; Weyssow et al., 2023), including ours, rely on previously reported fine-tuning or pre-training hyperparameters as an expedient solution and do not run the experiments with different seeds, due to the computation restrictions that incentifies the use of PEFT approaches. However, we note that Zhang et al. (2024) found that scaling up LoRA and Prompt-Tuning parameters does not significantly impact downstream task performance, though they also indicate that this effect may be highly task-dependent. Secondly, this study was limited to decoder-only models, despite encoder-decoder models also being applied to code generation tasks (Li et al., 2022). Additionally, we focused specifically on addition-based and re-parameterization-based PEFT methods. As new approaches are developed, further research should explore their impact on code generation tasks. Lastly, as model sizes increased during experimentation, we did not proportionally increase the amount of data used in training, as recommended by Kaplan et al. (2020) and Hoffmann et al. (2022). Future work should investigate this aspect further.

Acknowledgments

This work was supported by the GreenNLP project, which is funded by the Research Council of Finland.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Shamil Ayupov and Nadezhda Chirkova. 2022. Parameter-Efficient Finetuning of Transformers for Source Code.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri 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, Josh 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.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric

Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. 2022. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. ArXiv:1904.09751 [cs].

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.

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*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. ArXiv:2310.06825 [cs].

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. ArXiv:2001.08361 [cs, stat].

Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. 2023. Platypus: Quick, cheap, and powerful refinement of llms.

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.

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, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, 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!

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.

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.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

J. Liu, C. Sha, and X. Peng. 2023. An Empirical Study of Parameter-Efficient Fine-Tuning Methods for Pre-Trained Code Models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 397–408, Los Alamitos, CA, USA. IEEE Computer Society.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the neural information processing systems track on datasets and benchmarks*, volume 1. Curran.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art Parameter-Efficient Fine-Tuning Methods. `https://github.com/huggingface/peft`.

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.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the association for computational linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Purnawansyah, Zahrizhal Ali, Herdianti Darwis, Lutfi Budi Ilmawan, Sitti Rahmah Jabir, and Abdul Rachman Manga. 2024. Memory Efficient with Parameter Efficient Fine-Tuning for Code Generation Using Quantization. In *2024 18th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, pages 1–6.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a method for automatic evaluation of code synthesis. ArXiv: 2009.10297 [cs.SE].

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023a. Code Llama: Open Foundation Models for Code. ArXiv:2308.12950 [cs].

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023b. Code llama: Open foundation models for code.

Zhengyan Shi, Adam X. Yang, Bin Wu, Laurence Aitchison, Emine Yilmaz, and Aldo Lipani. 2024. Instruction tuning with loss over instructions.

Deze Wang, Boxing Chen, Shanshan Li, Wei Luo, Shaoliang Peng, Wei Dong, and Xiangke Liao. 2023. One Adapter for All Programming Languages? Adapter Tuning for Code Search and Summarization. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, page 5–16. IEEE Press.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. TRL: Transformer Reinforcement Learning. Publication Title: GitHub repository.

Martin Weyssow, Xin Zhou, Kisub Kim, David Lo, and Houari Sahraoui. 2023. Exploring parameter-efficient fine-tuning techniques for code generation with large language models.

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.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS 2022, page 1–10, New York, NY, USA. Association for Computing Machinery.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *2018 IEEE/ACM 15th international conference on mining software repositories (MSR)*, pages 476–486. IEEE.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. 2024. When Scaling Meets LLM Finetuning: The Effect of Data, Model and Finetuning Method. In *The Twelfth International Conference on Learning Representations*.

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. ArXiv:2401.00788 [cs].

Wentao Zou, Qi Li, Jidong Ge, Chuanyi Li, Xiaoyu Shen, Liguo Huang, and Bin Luo. 2023. A Comprehensive Evaluation of Parameter-Efficient Fine-Tuning on Software Engineering Tasks. ArXiv:2312.15614 [cs].