AN EFFICIENT FINE-TUNING APPROACH FOR A CODE GENERATION AND DEBUGGING ASSISTANT

Mayank Vyas, Savankumar Pethani, Mridul Tailor
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ

Vacha Patel, Charu Sneha Laguduva Ravi
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ

ABSTRACT

In modern software development, programmers spend a significant amount of time on repetitive coding tasks and debugging complex errors. This project aims to build an intelligent assistant that helps streamline these processes. By leveraging a powerful pre-trained language model, we will create a tool that can understand a developer's request—written in plain English—and automatically generate relevant code snippets or suggest fixes for bugs. The core idea is to train the model on a massive library of existing code, teaching it the patterns, syntax, and logic of programming. Our focus will be on making this advanced technology accessible and efficient, even without access to supercomputers, by using specialized training techniques. The final result will be a smart, responsive coding partner designed to boost developer productivity and reduce frustration.

***K*eywords** Code Generation · Debugging · Large Language Models · Fine-Tuning · LoRA · Python

1 Introduction and Problem Statement

1.1 Problem and Motivation

The demand for complex software continues to grow exponentially, placing immense pressure on developers to write high-quality code faster. While Integrated Development Environments (IDEs) offer basic autocompletion, they often lack the contextual understanding to assist with complex logic or abstract problems. Large Language Models (LLMs) like those in the GPT family [1] have shown remarkable code generation capabilities [2, 3, 4, 5, 6]. However, these models are often general-purpose and require significant computational resources and task-specific fine-tuning to become truly reliable developer tools.

This project is motivated by the need to bridge this gap. We aim to explore how powerful pre-trained models can be adapted into specialized, high-performing assistants that are both accurate and efficient. By focusing on resource-efficient techniques like Low-Rank Adaptation (LoRA) [7], we are tackling the critical challenge of democratizing access to state-of-the-art AI, making it feasible for individuals or small teams with limited computational budgets to build custom solutions.

1.2 Technical Background and Related Work

Our project's core involves fine-tuning a pre-trained transformer model (GPT-2) [1]. Standard fine-tuning is compute-intensive, so we are comparing it against Parameter-Efficient Fine-Tuning (PEFT) methods. Our primary technical approach, LoRA [7], freezes the base model and injects small, trainable matrices, drastically reducing resource needs compared to other methods like Adapters [8]. Our work integrates this efficiency technique with foundational code models [3, 6] and modern evaluation methods [2], with the goal of creating a practical tool for both code generation and debugging [9].

2 Progress Report

Our execution plan is divided into four main phases. We have successfully completed Phase 1 and have established a concrete baseline for Phase 2, yielding promising initial results.

2.1 Phase 1: Dataset Curation and Environment Setup (Completed)

This phase is 100% complete. (Led by: Mayank and Vacha). We established a Python/PyTorch/Hugging Face environment and processed the `google/code_x_glue_ct_code_to_text` (Python) dataset. This dataset provides over 251,000 training samples. Our pipeline formats this data into a `Prompt: [docstring] Code: [code]` structure for training.

2.2 Phase 2: Model Fine-Tuning (Baseline Established)

We have established our baseline model (Led by: Savankumar and Mridul) by performing a standard (full) fine-tuning of GPT-2 (124M parameters). The training was conducted on a 10,000-sample subset of the Python dataset for 3 epochs. The validation loss steadily improved, decreasing from 1.28 at epoch 1 to 1.24 at epoch 3. The LoRA implementation (‘USE_LORA = False‘) was not enabled for this run, providing a solid performance baseline to compare against.

2.3 Preliminary Results and Achievements

Our primary achievement is the successful implementation of a new, robust evaluation pipeline that iterates through each saved checkpoint. Crucially, this script feeds *only* the prompt to the model and evaluates the *newly generated* code, giving a true measure of its generative capabilities.

The results from our 3-epoch baseline run are detailed in Table 1 and illustrated in Figure 1 and Figure 2. We are highly encouraged by these findings. Most notably, we achieved a **39.58% execution pass rate** by the third epoch. This confirms the model is generating functionally correct Python code. The text-based metrics (BLEU, ROUGE-L) also show slight improvement with more training.

Table 1: Baseline Model Evaluation Over 3 Epochs (GPT-2 Full-Tune, 10k Python samples)

Epoch	BLEU Score	ROUGE-L (F1)	Execution Pass Rate
1	15.97	0.443	35.42%
2	16.20	0.438	31.25%
3	17.03	0.444	39.58%

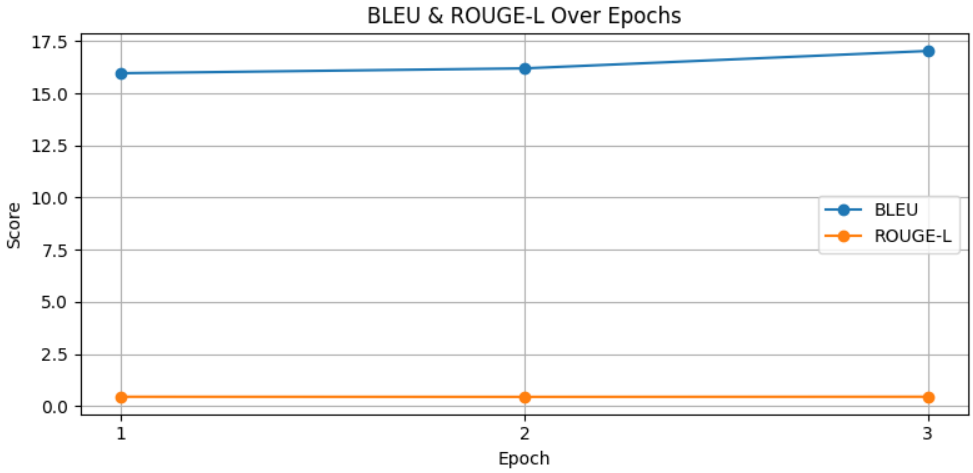

Figure 1: BLEU and ROUGE-L scores per training epoch.

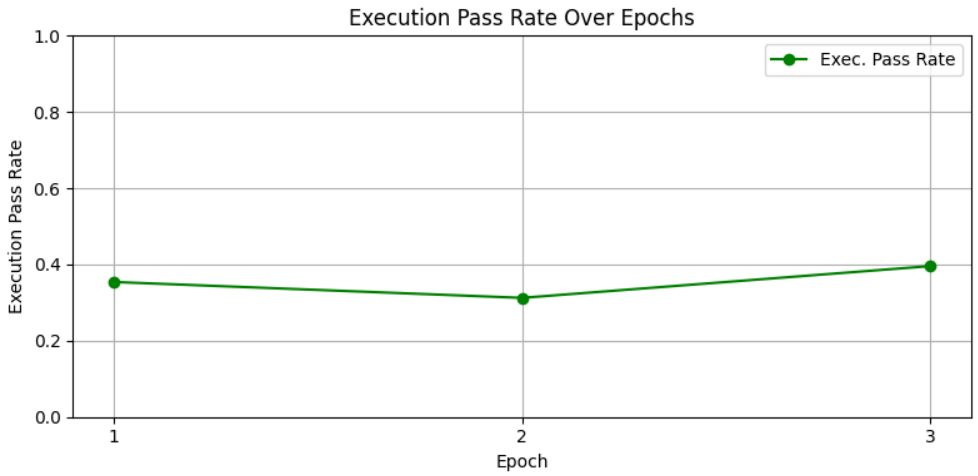

Figure 2: Execution pass rate per training epoch.

2.4 Challenges and Risk Mitigation

Our preliminary results have clarified our main challenge: while we have achieved a promising 39.58% execution pass rate (Figure 2), our goal is to improve this functional correctness metric significantly. The results also suggest that textual similarity (Figure 1) and functional correctness are not perfectly correlated, validating our plan to use a hybrid evaluation.

Our mitigation plan is two-fold:

1. Implement LoRA as planned to get a direct comparison of efficiency and functional performance against our new baseline.
2. Scale up our training from the 10,000-sample subset to the full 251,000-sample dataset, which we expect will substantially improve all metrics.

3 Plan for Remainder of Semester

Our plan remains on track, with priorities now refined based on our successful baseline results.

Phase 2 (Continued): LoRA Comparison (Weeks 8-9) (Led by: Savankumar and Mridul) We will immediately implement and train a LoRA-enabled GPT-2 model on the same 10,000-sample subset for 3 epochs. This will provide a direct, fair comparison of resource usage (VRAM, time) and metric performance (BLEU, ROUGE, Exec. Rate) against the baseline results in Table 1.

Phase 3: Scale Up & Iterate (Weeks 10-11) (Led by: Charu and Vacha) This is our new priority. We will take the most promising model (baseline vs. LoRA) and begin training it on the **full 251,820-sample training dataset**. This scaling is the most critical step to significantly boost our functional correctness. We will also begin developing the core assistant interface (CLI or basic web page) in parallel.

Phase 4: Final Evaluation and Report (Week 12) (Led by: Full Team) The full team will conduct a final, large-scale evaluation of our best scaled-up model on the test set. We will compile the final "Performance Analysis Report," with a deep analysis of the trade-off between textual similarity and functional correctness, and prepare the final project presentation.

Acknowledgments

We thank our professor for the valuable guidance and resources provided, including the "Project Idea 3" guide which helped shape our technical approach.

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8), 2019.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, H. P. de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[3] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

[4] Yujia Li, David Choi, Junyoung Chung, Quoc V Le, Lu Han, Finbarr Hsieh, Paul St-Jean, Sergii Tsvyashchenko, Maja Kunc, H-Chris Soyer, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[5] Raymond Li, Loubna Allal, Yang Zi, Guillaume Lample, T Call, R, W Lee, A Kocetkov, C Mou, B., C Akiki, et al. Starcoder: a state-of-the-art llm for code. *arXiv preprint arXiv:2305.06161*, 2023.

[6] Yue Wang, H-A Le, and T Nguyen. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[7] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv preprint arXiv:2106.09685.

[8] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Atta, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning (ICML)*, pages 2790–2799. PMLR, 2019.

[9] Chun S Xia and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 2635–2647, 2023.