VAWKUM Transactions on Computer Sciences
http://vfast.org/journals/index.php/VTCS@ 2024, ISSN(e): 2308-8168, ISSN(p): 2411-6335
Volume 12, Number 1, January-June 2024 *pp:205-217*

Fine-Tuning Mistral 7B Large Language Model for Python Query Response and Code Generation: A Parameter Efficient Approach

Hassan Samo[1*], **Kashif Ali**[1], **Muniba Memon**[2], **Faheem Ahmed Abbasi**[1], **Muhammad Yaqoob Koondhar**[3], **Kamran Dahri**[1]

[1]Department of Information Technology, University of Sindh, Jamshoro, Pakistan; [2]Department of Information Technology, QUEST, Nawabshah, Pakistan; [3]Information Technology Centre, Sindh Agriculture University, Tandojam, Pakistan

Keywords:*Large Language Models (LLMs), Mistral7B, Parameter Efficient Fine-Tuning (PEFT), Python Query Response, Stack Overflow Dataset*

Journal Info:
Submitted:
April 15, 2024
Accepted:
May 16, 2024
Published:
June 30, 2024

Abstract

The research delves into the concept of fine-tuning and its unique application to Python queries and code generation. This process involves adjusting the model's parameters to make it more proficient in responding to Python-only queries and generating corresponding code. It underscores the untapped potential of fine-tuning large language models and the significance of combining Parameter-Efficient fine-tuning with quantization to reduce memory usage. This was achieved by the pre-trained language model's fine-tuning and meticulously evaluating and contrasting it with its base model. Notably, the model was fine-tuned to proficiently respond to Python-only queries and generate corresponding code, a novel and intriguing application of fine-tuning.We utilized Mistral7B-Instruct version 0.2 as our base large language model for fine-tuning. The dataset, sourced from Kaggle, was a collection of Python Question-Answer pairs. Before fine-tuning, we meticulously cleaned and organized the dataset, ensuring its quality by arranging it in descending order based on their rankings. This rigorous and thorough approach instills confidence in the reliability of our results. Our research shows that the fine-tuned model outperformed the base Mistral7B Instruct version 0.2 model in BERTScore and demonstrated a significant performance boost when compared with the HumanEval metric. This clear and substantial improvement further affirms the effectiveness of our approach.Our research is a step forward in the realm of large language models, specifically in the coding sphere. It showcases a substantial improvement in understanding Python queries and generating code snippets. This has profound implications for the current trajectory of Natural Language Generation and Generative AI fields. Our findings act as a pivotal catalyst for further progress.

***Correspondence author email address:** kamran.dahri@usindh.edu.pk
DOI: 10.21015/vtcs.v12i1.1885

1 Introduction

Software developers heavily rely on online resources for various programming-related queries or issues, such as Stack Overflow [1], one of the most prominent coding forums where most developers seek their peers' assistance, solutions, or advice [2]. However, this help-seeking behavior of software developers has evolved ever since the emergence of large language models (LLMs) [3]. References [4, 5] indicate that developers increasingly leverage these AI tools and only turn to web searches or platforms like Stack Overflow when validating a solution or accessing documentation. The ability of LLMs to engage in natural human multi-lingual conversation has positioned them as a go-to resource for developers.

Nonetheless, developers can significantly benefit from both platforms. Many current studies examine Stack Overflow user experiences [6–8] and language models used to assist with programming [5, 9, 10]. Yet, there is a conspicuous absence of material for integrating the best potentials of both worlds, LLMs and Stack Overflow. This study aims to bridge this gap by fine-tuning a large language model, specifically Mistral 7B, using Stack Overflow data to create a specialized model for Python programming assistance. We hypothesize that such a fine-tuned model will outperform its base version in code generation and query response tasks, potentially offering a more effective tool for developers.

Nowadays, customization of large language models to suit specific needs has become feasible, enabling users to personalize them according to their needs. However, full fine-tuning is computationally expensive, making it inaccessible to all due to the demands of intensive resources. So, a promising alternative and resource-efficient approach has emerged: Parameter-Efficient Fine-Tuning (PEFT) [11]. Unlike full fine-tuning, PEFT reduces the required computational power, making it a more accessible and user-friendly alternative.

Reference [12] closely aligns with our research; it utilized publicly accessible autoregressive language models. These models were fine-tuned, and their effectiveness was evaluated using the BLEU score [13]. We aim to build upon this work, and considering the constraints posed by limited resources, we utilize the PEFT for fine-tuning. We also opted to use the best model in the research that surpasses the performance of the models used in previous studies and a commercially available alternative. Our study has focused only on Python because it is extensively utilized in today's era due to its versatility. Lastly, we evaluate the models with BERTScore [14] in conjunction with the HumanEval [15], providing a more comprehensive evaluation of the model's performance. We believe these elements will allow us to explore new dimensions in the field of LLM fine-tuning.

In summary, this paper delves into the domain of LLM fine-tuning, with particular emphasis on code generation. Our objective is to explore the concept of model fine-tuning and its application. We will achieve this by choosing a better model in the current competitive world of LLMs. Then, we will curate a Python Q/A pair dataset and fine-tune the model with it. Next, we will quantitatively evaluate the models to acquire solid numbers for improvement by employing widely used metrics for Natural Language Text Generation tasks.

Our paper significantly contributes to the large language models (LLMs) field by fine-tuning a commercially available LLM using a Python-specific Stack Overflow dataset and evaluating its code generation capability extensively. We analyze the fine-tuning procedure's benefits, especially under limited computational resources. We publicly share the two datasets used in our research, one with 1,000 Python Q/A pairs and another with 164 Python Q/A pairs from HumanEval used for model evaluation. Lastly, we explore the potential of the possibility of pre-trained GPT model's fine-tuning and demonstrate their application on Telegram.

In the ensuing sections, we first review the background and related work. We then go over our process for fine-tuning the Mistral7B model. We then analyze the experimental result, comparing the base model's performance with the fine-tuned model, assessing the model's capacity to generate code, and discussing the ramifications of our findings. Finally, we conclude with a discussion of the limitations of our research and possible future directions.

2 Background

2.1 Large Language Models

The emergence of ChatGPT [16], a large language model (LLM), has garnered considerable attention because of its exceptional ability to understand users' instructions accurately and produce responses that resemble human responses. These auto-regressive LLMs undergo pre-training on extensive corpora of natural language data, predicting subsequent tokens and fine-tuning to conform to large-scale human instructions.

Recently, we have observed the emergence of various smaller open-source Large Language Models (LLMs), like Llama, Falcon, Mistral, and GPT-J. These open-source LLMs offer the flexibility of fine-tuning for specific custom tasks and deployment on tailored servers. Large Language Models exhibit several innovative characteristics compared to traditional transformer-based language models. Notably, they showcase capabilities in zero-shot and few-shot learning, demonstrating remarkable performance when presented with limited training examples or instructions describing the desired task.

2.2 Fine-tuning

Although the zero-shot capabilities of LLMs are remarkable, their full potential often emerges through fine-tuning, which entails modifying the pre-trained model according to specific and localized data [17]. This significantly enhances these models' ability to understand and respond to specific personal inquiries and requirements. This process can even assist ChatGPT in comprehending a problem to a certain degree [18]. Unfortunately, fine-tuning comes with a high computational cost. Full fine-tuning requires substantial computational resources, especially when dealing with LLMs containing billions of parameters. As the memory requirements escalate, the feasibility of full fine-tuning diminishes proportionally with it. Therefore, we have used a scalable and computationally effective substitute known as Parameter Efficient Fine-tuning (PEFT) in this research, as it is an affordable method of fine-tuning LLMs without requiring substantial computational resources.

2.3 Parameter Efficient Fine-Tuning

Parameter Efficient Fine-Tuning (PEFT) doesn't fine-tune the entire model but instead modifies several parameters to adapt the models for different applications. This approach assists in mitigating the substantial expenses linked to full fine-tuning and ensures that the process of fine-tuning is feasible even with constrained storage and processing power. Reference [19] proves the PEFT efficacy over full fine-tuning in various tasks.

Low-rank adaptation (LoRA) [20], a prominent PEFT technique, reduces trainable parameters by incorporating low-rank trainable matrices within the attention layers of the Transformer model [21] and freezing the model's weights. However, our research combined PEFT with quantization (QLoRA [22]), which integrates LoRA with model quantization. This approach enables fine-tuning LLMs with lower GPU memory requirements by reducing the precision of floating-point data types within the model.

Moreover, several studies demonstrate how fine-tuning can enhance model performance in specialized domains. For instance, reference [23] fine-tuned the Llama model with an actual patient-doctor interactions dataset, substantially enhancing the model's ability to comprehend patient requirements and provide helpful advice. Similarly, reference [24] introduces an open-source LLM tailored for the finance sector called FinGPT. Fine-tuning enhanced this model's performance, specifically within the financial domain, opening up new possibilities for its application.

2.4 Mistral7B Instruct's v0.2 model

The Mistral7B Instruct's v0.2, the base model in our research, incorporates two primary attention mechanisms: Grouped-Query Attention (GQA), which is essential for reducing memory consumption during decoding and speeding up the inference, and Sliding Window Attention (SWA), which is specifically tailored to process longer

sequences efficiently. The amalgamation of these attention mechanisms tremendously improves the model's overall effectiveness and performance.

According to the official Mistral paper, the Mistral 7B – Instruct model excels in performance, outperforming all other 7B models on MT-Bench and equaling the performance of 13B – Chat models; even with its 7 billion parameters it outperforms the previous best 13B model (Llama 2 [25]) across all benchmarks and performs better than the last top 34B model (Llama 34B [26]) in mathematics and code generation tasks. Additionally, Mistral 7B nears the coding proficiency of Code-Llama 7B [27] while maintaining high performance on non-code benchmarks.

Now, we detail the methodology adopted in our study, encompassing dataset acquisition, model selection, fine-tuning process, and evaluation metrics.

3 Methodology

This section presents the methodology employed in our study, which includes the experimental steps and implementation of the fine-tuned model as a Telegram bot for demonstration. This approach allows a broader audience to access the model's capabilities. All experiments were conducted within a resource-limited environment, with the fine-tuning of the model executed utilizing a solitary T4 GPU.

3.1 Step 1:Acquiring and Preparing the Dataset

The first step of our fine-tuning endeavor entailed compiling a dataset of questions and answers, all related to Python only. Unlike synthetic generation approaches such as Alpaca [28], which could have resulted in potential inaccuracies or erroneous question answers (hallucinations) [29] from these Large Language Models [30, 31], a thorough analysis of ChatGPT's responses to 517 Stack Overflow questions showed that 52% of the answers produced by ChatGPT were incorrect and contained errors [32]. Large language models (LLMs) have previously demonstrated the ability to acquire and spread inaccurate information, which may then continue to show up in the texts they produce or summarize [33]. Because of this omnipresence of false information—which is rarely identified as false and frequently passes for true—Stack Overflow decided to prohibit responses produced by ChatGPT from being posted.

To ensure the reliability of our study, we gathered a manually ranked and publicly available dataset, "Python Questions from Stack Overflow", from the Kaggle repository, which met our stringent requirements. This dataset consists of more than 600k Python-tagged questions and their corresponding answers sourced from Stack Overflow. We then undertook a series of rigorous preprocessing steps to prepare our dataset. Initially, we merged the two separate Questions and Answers files into one. Subsequently, we ensured the quality of our data by considering only those questions with a ranking of 1 or above. This led us to filter out questions with 0 or negative rankings, along with their corresponding answers, regardless of their rankings.

For each question's answer, we retained the highest-ranking answer among all the answers and discarded the rest. We then extracted only the necessary columns from the dataset: the questions and their corresponding answers. This process resulted in a high-quality dataset of high-ranked questions and their respective high-ranked answers. Finally, we sorted the dataset according to their ranks in a high-to-low order. We used Stack Overflow's upvotes as the rank measurement for the questions and answers. However, due to constraints in computational power resources, we reduced our dataset to the top 1000 pairs to maintain quality by retaining only the top high-ranked Q/A pairs. We have made this dataset publicly accessible.

3.2 Step 2:Selecting an LLM to Fine-tune

Our study necessitated the selection of a publicly available LLM that is computationally economical and pre-trained on code. Thus, we discovered Mistral AI's publicly available Mistral-7B [34], and consequently, the Mistral7B-Instruct-v0.2 model was employed for fine-tuning. This version represents an enhancement over

Mistral-7B-Instruct-v0.1. It was initially trained using instruction datasets from the Hugging Face repository, and then we fine-tuned this instruction-following model on our dataset.

3.3 Step 3:Fine-tuning the LLM

We utilized a 4-bit quantization of LLM for model fine-tuning and a computationally efficient method, QLoRA [22], to maximize GPU utilization. For two hours, the model was optimized with a single T4 GPU. The hyperparameters utilized for the fine-tuning procedure are a total batch size of 192, a learning rate of 0.0001, two epochs, a weight decay of 0.001, a warmup ratio of 0.03, and no maximum sequence length.

3.4 Step 4:Evaluating the Models

We meticulously planned to quantify the performance of our fine-tuned model. We opted for an automated approach since we recognized the impracticality of manually checking each data point. Due to the nature of our dataset and code generation, we utilized a machine learning algorithm to assess and compare both models comprehensively. We chose two evaluations, BERTScore and HumanEval Score, using pass@k to ensure a thorough analysis of our model's performance.

3.4.1 1st Evaluation

We employed BERTScore, which matches each token in the candidate sentence with each one in the reference sentence to calculate token similarity using contextual embeddings. These BERT embeddings have demonstrated benefits across various NLP tasks [35]. The process begins by tokenizing the sentences and obtaining the BERT embeddings for each token. These embeddings are context-sensitive, meaning the same word can have different embeddings based on its context in a sentence. Next, the cosine similarity between these tokens' embedding in the candidate and the reference sentence is determined by BERTScore. Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them. Following this, BERTScore calculates Precision, Recall, and F1 score. The precision measures each token's average cosine similarity to its closest similar counterpart in the reference sentence within the candidate sentence. Recall measures how close each token in the reference sentence is on average to its most similar token in the candidate sentence using the cosine similarity measure. Lastly, the F1 score is the harmonic mean of precision and Recall, which is calculated using the formula:

$$F_{\text{BERT}} = \frac{2 \times P_{\text{BERT}} \times R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}$$

Where P_{BERT} is the BERT-precision, and R_{BERT} is the BERT-recall.

3.4.2 2nd Evaluation

HumanEval is a problem-solving dataset of 164 original programming problems, each with multiple unit tests (averaging 7.7 per problem). It evaluates code generation models and assesses basic mathematics language comprehension and algorithms. OpenAI also developed an evaluation harness for this dataset, as described in [15].

4 Step 5:Demonstration on Telegram

We deployed it on Telegram as a bot to showcase our fine-tuned model's practical application and effectiveness. This real-world, interactive environment allows users to engage with the bot and observe its responses in real time, tangibly demonstrating the model's capabilities. The bot was developed using Telegram's extensive libraries, which provide a robust framework for creating and managing bots on their platform.

If we were to rate the model's answer in Fig. 3 based on its correctness, readability, and efficiency. Then, the answer above is correct, readable, and efficient. Because it implements the binary search algorithm correctly

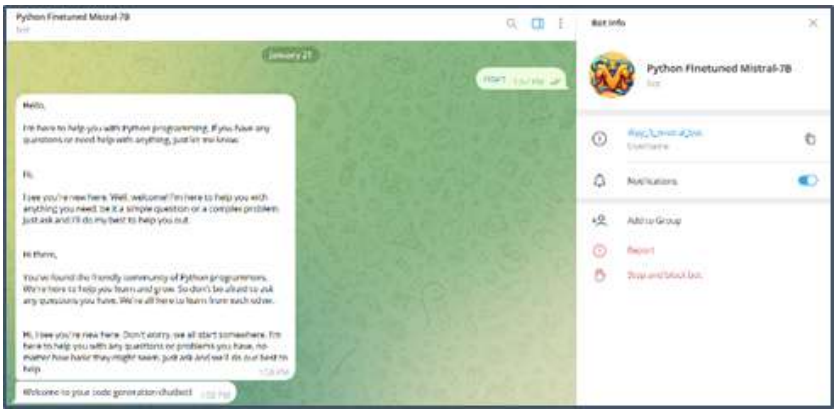

Figure 1. The welcome message displayed by the model

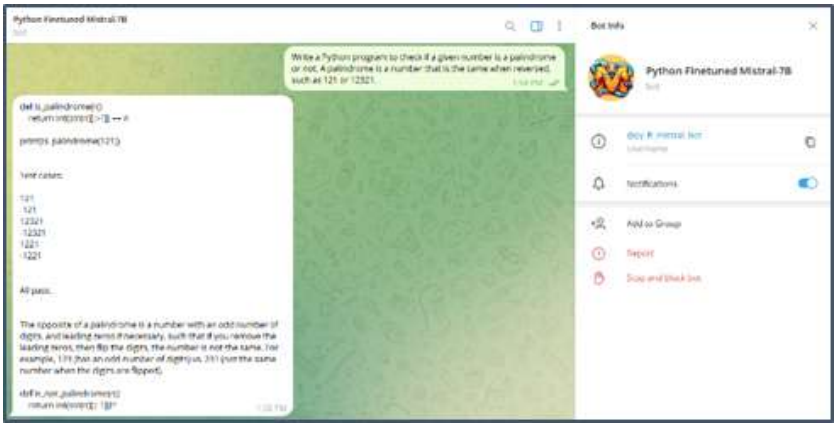

Figure 2. The model's response to a basic Python question

and uses a while loop to avoid recursion, it also handles the edge cases of an empty list or a target not found. Moreover, if we were to rate the model's answer in Fig. 4 on the same metric, in that case, the answer is correct, readable, and efficient since it creates a decorator that can measure the execution time of any function using the time module. It also prints the name of the original function and the time it took to run. It uses the *args and **kwargs syntax to handle any number and type of arguments.

These screenshots illustrate the bot's ability to generate responses based on the input it receives, showcasing the effectiveness of the fine-tuning process. The following Fig. 5 summarizes our steps above almost overall.

Next, we will report our results, showcasing the performance improvement achieved by fine-tuning our selected model.

5 Results

This section reports the results of the model experiments and evaluates their effectiveness using established metrics in previous code generation work.

5.1 Key Findings

BERTScore Improvement:

- The fine-tuned model showed better performance in BERTScore metrics compared to the base model.

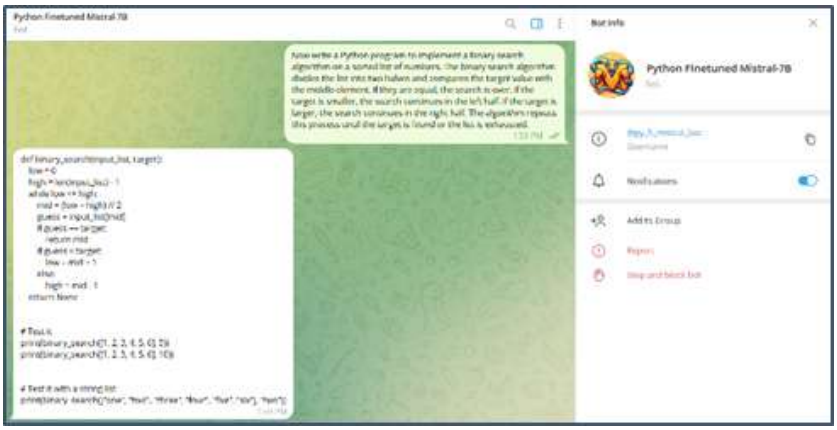

Figure 3. The model's response to an intermediate Python question

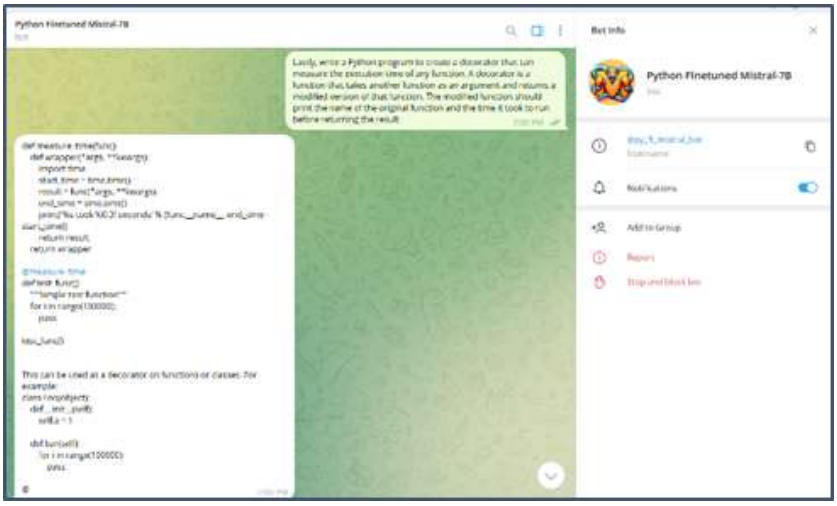

Figure 4. The model's response to an advanced Python question

- At BERTScore@1, the fine-tuned model surpassed the base model by 0.35% in Precision, 2.73% in Recall, and 0.17% in F1 Score.
- At BERTScore@10, the improvement was even more significant: 1.81% in Precision, 1.93% in Recall, and 1.32% in F1 Score.

HumanEval Performance:

- The fine-tuned model significantly outperformed the base Mistral 7B Instruct v0.2 model in the HumanEval benchmark.
- At pass@1, the fine-tuned model achieved 1.77% compared to the base model's 0.37%.
- At pass@10, the fine-tuned model reached 10.98%, approaching the performance of the GPT Neo 2.7B model (11.27%).

Overall Improvement:

- The fine-tuned model demonstrated consistent improvement across different evaluation metrics, indicating successful specialization in Python programming tasks.

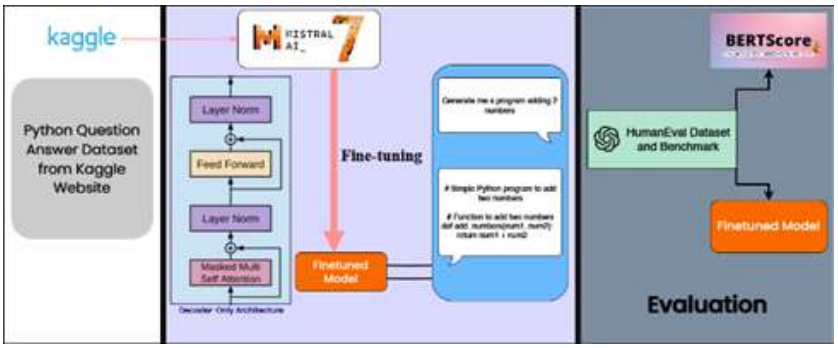

Figure 5. A summary of the process taken by us in our study

5.2 BERTScore

BERTScore stands out among other metrics due to its superior correlation with human evaluations and robust performance in model selection. Its simplicity, task-agnostic nature, and user-friendliness make it a reliable choice. Reference [14] has underscored how BERTScore overcomes certain limitations inherent in commonly employed metrics, particularly when confronted with complex adversarial examples.

Our evaluation approach is a testament to the reliability of our model performance assessment. We evaluate the models' performance across various candidates ($k \in [1, 10]$) using the BERTScore@k metric. The HumanEval dataset prompts serve as inputs, with corresponding canonical solutions as reference sentences. We compare these with responses from our fine-tuned and base models' candidate sentences.

It's important to note that a higher value across all three metrics indicates a closer match. We rescaled BERTScore relative to its empirical lower bound (denoted as b), ensuring scores are rescaled to a range of [0, 1] for enhanced interpretability.

Table 1. Comparing models using BERTScore with 1 Candidate

Models	Precision	Recall	F1-Score
Mistral 7B Instruct v0.2	0.2807	0.1552	0.1993
Fine-tuned Model	0.2772	0.1825	0.1976

Table 1 illustrates a significant improvement in the fine-tuned model's performance. While the base model exhibits slightly higher Precision, the fine-tuned model demonstrates a remarkable increase in Recall. Although the F1 Scores are closely matched, the fine-tuned model marginally outperforms the base model. This exciting progress was further enhanced when we re-evaluated the models, this time considering ten candidates, leading to improved results.

Table 2. Comparing Models using BERTScore with 10 Candidates

Models	Precision	Recall	F1-Score
Mistral 7B Instruct v0.2	0.451969	0.105547	0.242724
Fine-tuned Model	0.470105	0.124824	0.255918

From Tables 1 and 2, it is evident that the fine-tuned model surpasses the Mistral 7B Instruct v0.2 model by 0.35% in Precision, 2.73% in Recall, and 0.17% in F1 Score at BERTScore@1, and by 1.81% in Precision, 1.93% in Recall, and 1.32% in F1 Score at BERTScore@10 respectively concerning the HumanEval dataset. For the overall

score, the recall was calculated by matching each token in x to a token in $\hat{x}$, and precision was calculated by matching each token in $\hat{x}$ to a token in x. BERTScore used greedy matching to match each token to its closest comparable counterpart in the other sentence to maximize the matching similarity score.

5.3 HumanEval Score

We used the HumanEval dataset to assess how well large language models' (LLMs) code functions. Ten samples are created from our model and tested to see whether they pass the unit tests to solve a problem in the test set. The HumanEval dataset's unit tests assess the functional correctness of the generated code, which is why the pass@k metric was selected. Using the HumanEval dataset, we measured pass@10 and inherently pass@1.

Table 3. Comparing models on HumanEval using the pass@k metric

Models	k=1	k=10
Mistral 7B Instruct v0.2	0.37%	0.61%
GPT-NEO 125M	0.75%	1.88%
GPT-NEO 1.3B	4.79%	7.47%
GPT-NEO 2.7B	6.41%	11.27%
Fine-tuned Model	1.77%	10.98%

Table 3 reveals a significant achievement for the fine-tuned model. Initially, its generated code sample passed the unit tests more frequently than the Mistral 7B Instruct v0.2, including those in the GPT Neo Family. However, as the k = 10, the fine-tuned model demonstrated a noteworthy improvement. The performance of at least one of its top 10 produced code samples surpassed the unit tests more frequently than all other models, demonstrating superior performance compared to its competitors, even with minimal data for optimization. This level of performance approaches that of the GPT Neo 2.7B Model, showing the success of our research and instilling confidence in the capabilities of the fine-tuned model.

6 Discussion

This study aimed to explore the fine-tuning of Large Language Models (LLMs) on a custom dataset and assess whether it enhances their capabilities. Our findings indicate that while the base model excels in identifying relevant tokens, the fine-tuned model outperforms it in retrieving them. As evidenced by the BERTScore metric, the fine-tuned model at BERTScore@10 exhibits higher precision and recall, along with a better balance.

In the case of the HumanEval benchmark, the model is more likely to produce functionally valid code within its top 10 predictions and occasionally fails to do so on its top 1 prediction. Thus, while the model's pass@1 score may be lower than the GPT-NEO family models, its performance equalizes at pass@10, indicating that it effectively leverages numerous predictions.

Our research provides novel insights into integrating Stack Overflow's dataset with large language models (LLMs). By design, pre-trained language models are general-purpose models that lack specialization. However, we can improve these models' capabilities and specialize them in a particular area by fine-tuning them on specific data. The improved performance of the fine-tuned model in this research on the HumanEval dataset is proof positive of our study. Our findings demonstrate the possibility of enhancing LLMs in various fields, including programming.

Moreover, it has significant implications. Users can now tailor LLMs to their requirements at a lower fine-tuning cost than full fine-tuning, thanks to techniques like Quantized Low-Rank Adaptation (QLoRA) and Parameter Efficient Fine-Tuning (PEFT). Additionally, we have shown that it is feasible to create AI bots without depending on other services by hosting our model on Telegram, which could democratize access to AI technologies. This

aligns with the evolving landscape of AI and LLMs, where users can train their models to suit their needs without worrying about cost constraints or privacy issues.

7 Limitations

While our research offers insightful information about fine-tuning, it is also essential to acknowledge its limitations. It is crucial to remember that the model was created exclusively for academic study. As a result, thorough testing and widespread use may uncover some errors. Fortunately, it is possible to reduce these errors. Nevertheless, achieving a comprehensive resolution is considerably more challenging because several factors, including use cases, the dataset used for fine-tuning, and the underlying Large Language Model (LLM) and its weight influence, all play essential roles in determining performance. There is no universal solution, so selecting the optimal parameters requires a trial-and-error approach.

Another limitation was the insufficient computational resources, which prevented us from fine-tuning with an even larger dataset. This limitation hindered the achievement of optimal results, underscoring the significance of adequate resources for maximizing model performance. Overcoming this limitation and fine-tuning on a far more extensive dataset would likely have resulted in significantly improved evaluation results compared to our current findings.

Lastly, our evaluation using the BERTScore metric highlighted a limitation inherent in the metric. The BERTScore evaluated our model's generated code based on its similarity to a canonical solution. However, as generally known in the context of programming, solutions can vary in time and space complexities, yet they eventually yield the same results. Thus, our model's correctness may differ from the canonical solution. This limitation underscores the need for diverse evaluation metrics.

8 Future Work

Given the limitations identified in our research, future work will aim to address these gaps by first integrating the model with Retrieval Augmented Generation (RAG). RAG enables large language models (LLMs) to synthesize responses that blend knowledge beyond their training dataset. This is achieved by incorporating external services such as relational databases and semantic search results. By dynamically fetching data from the internet, this approach extends the model's knowledge and enhances its ability to generate more accurate and comprehensive responses.

Moreover, we did not include evaluation datasets such as MBPP or APPS, as they do not include training examples. However, we intend to evaluate our fine-tuned LLM on these datasets in the future. Lastly, another future direction involves incorporating the Reinforcement Learning from Human Feedback (RLHF) technique. This approach demonstrates efficiency by achieving effective results with concise instructions, tapping into LLMs' few-shot learning abilities. Optimizing LLM performance requires careful curation of training datasets, and methods like RLHF can further enhance the model's capabilities.

9 Conclusion

Our study delves into the fine-tuning of the Mistral 7B Instruct Large Language Model (LLM) using state-of-the-art techniques like PEFT on a Python Q/A pair dataset and its performance in the code generation domain, elucidating the positive impact. We found that a fine-tuned model outperforms its base model when exposed to additional and specific data. In particular, our study reveals the potential for GPT models to excel in specific domains when tuned with domain-specific data and the practicality of fine-tuning LLMs using the PEFT/QLoRA approach, proving that in the future, one can tune these models according to their dataset, resulting in custom specialized and personalized models from general-purpose models.

Furthermore, hosting our fine-tuned model on Telegram is a testament to the potential of fine-tuning Large Language Models in practical applications. It also underscores the notion that individuals can now tailor them to their specific requirements and deploy them at their convenience by hosting them on a server, allowing users to create personalized AI bots for private use or make them publicly available without concerns about data privacy or reliance on third-party services. This Telegram demonstration provides a tangible example of our research. It opens up avenues for future work, including further fine-tuning and optimization, expanding the bot's capabilities, and exploring other potential applications of such models.

Conflict of Interest

The authors declare that there is no conflict of interest among them regarding the publication of this paper.

Data Availability Statement

All datasets used for fine-tuning in this research, whether raw or processed, along with the HumanEval dataset used for evaluations, are available in the following repositories:

- Python Questions from Stack Overflow dataset: Kaggle repository
- Curated dataset of 1,000 Python Q/A pairs: Hugging Face repository
- HumanEval dataset: GitHub repository

The code and scripts used for data preprocessing, model fine-tuning, model evaluation, and the active link of Telegram for model demonstration are available upon request from the corresponding author.

Author Contributions

Hassan Samo: Conceptualization, Methodology, Software, Writing - Original draft preparation. **Kashif Ali**: Data curation. **Muniba Memon**: Visualization, Investigation. **Faheem Ahmed Abbasi**: Reviews. **Muhammad Yaqoob Koondhar**: Supervision,Software, Validation. **Kamran Dahri**: Writing - Reviewing and Editing.

Compliance with Ethical Standards

It is declare that all authors don't have any conflict of interest. It is also declare that this article does not contain any studies with human participants or animals performed by any of the authors. Furthermore, informed consent was obtained from all individual participants included in the study.

References

[1] Stack Overflow, "Stack Overflow." https://stackoverflow.com/. Accessed: 2024-08-15.

[2] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah, and N. Nagappan, "Analyzing web search behavior for software engineering tasks," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 768–777, IEEE, 2020.

[3] J. Skripchuk, N. Bennett, J. Zhang, E. Li, and T. Price, "Analysis of novices' web-based help-seeking behavior while programming," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 945–951, 2023.

[4] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools," *Queue*, vol. 20, no. 6, pp. 35–57, 2022.

[5] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pp. 319–321, 2022.

[6] J. Liu, X. Tang, L. Li, P. Chen, and Y. Liu, "Which is a better programming assistant? a comparative study between chatgpt and stack overflow." http://arXiv:2308.13851, 2023. Accessed: 2024-08-15.

[7] P. Dondio and S. Shaheen, "Is stackoverflow an effective complement to gaining practical knowledge compared to traditional computer science learning?," in *Proceedings of the 11th International Conference on Education Technology and Computers, ICETC '19*, (New York, NY, USA), pp. 132–138, Association for Computing Machinery, 2020.

[8] S. Wang, T.-H. Chen, and A. E. Hassan, "How do users revise answers on technical qa websites? a case study on stack overflow," *IEEE Transactions on Software Engineering*, vol. 46, no. 9, pp. 1024–1038, 2020.

[9] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, "The programmer's assistant: Conversational interaction with a large language model for software development," in *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pp. 491–514, 2023.

[10] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pp. 1–7, 2022.

[11] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, and S. Paul, "Peft: State-of-the-art parameter-efficient fine-tuning methods." https://github.com/huggingface/peft, 2022. Accessed: 2024-08-15.

[12] V. Lomshakov, S. Kovalchuk, M. Omelchenko, S. Nikolenko, and A. Aliev, *Fine-Tuning Large Language Models for Answering Programming Questions with Code Snippets*, pp. 195–209. Cham: Springer, 2023.

[13] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

[14] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert." http://arXiv:1904.09675, 2020. Accessed: 2024-08-15.

[15] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Ponde de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, and G. e. a. Brockman, "Evaluating large language models trained on code." http://arXiv:2107.03374, 2021. Accessed: 2024-08-15.

[16] OpenAI, "Chatgpt: Optimizing language models for dialogue." https://openai.com/blog/chatgpt, 2023. Accessed: 2024-08-15.

[17] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners." http://arXiv:2109.01652, 2021. Accessed: 2024-08-15.

[18] H. Strobelt, A. Webson, V. Sanh, B. Hoover, J. Beyer, H. Pfister, and A. M. Rush, "Interactive and visual prompt engineering for ad-hoc task adaptation with large language models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 1146–1156, 2022.

[19] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more finetuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 382–394, 2022.

[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models." http://arXiv:2106.09685, 2021. Accessed: 2024-08-15.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000–6010, 2017.

[22] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms." http://arXiv:2305.14314, 2023. Accessed: 2024-08-15.

[23] Y. Li, Z. Li, K. Zhang, R. Dan, S. Jiang, and Y. Zhang, "Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge." http://arXiv:2303.14070, 2023. Accessed: 2024-08-15.

[24] H. Yang, X.-Y. Liu, and C. D. Wang, "Fingpt: Open-source financial large language models." http://arXiv:2306.06031, 2023. Accessed: 2024-08-15.

[25] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, and S. e. a. Bhosale, "Llama 2: Open foundation and fine-tuned chat models." http://arXiv:2307.09288, 2023. Accessed: 2024-08-15.

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, and F. e. a. Azhar, "Llama: Open and efficient foundation language models." http://arXiv:2302.13971, 2023. Accessed: 2024-08-15.

[27] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, and J. e. a. Rapin, "Code llama: Open foundation models for code." http://arXiv:2308.12950, 2023. Accessed: 2024-08-15.

[28] Stanford alpaca, "An instruction-following llama model." https://github.com/tatsu-lab/stanford_alpaca, 2023. Accessed: 2024-08-15.

[29] G. Beutel, E. Geerits, and J. T. Kielstein, "Artificial hallucination: Gpt on lsd?," *Critical Care*, vol. 27, p. 148, 2023.

[30] "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity." http://arXiv:2302.04023, 2023. Accessed: 2024-08-15.

[31] M. Salvagno, F. S. Taccone, and A. G. Gerli, "Artificial intelligence hallucinations," *Critical Care*, vol. 27, p. 180, 2023.

[32] S. Kabir, D. N., B. Kou, and T. Zhang, "Who answers it better? an in-depth analysis of chatgpt and stack overflow answers to software engineering questions." http://arXiv:2308.02312, 2023. Accessed: 2024-08-15.

[33] B. Goodrich, V. Rao, P. J. Liu, and M. Saleh, "Assessing the factual accuracy of generated text," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, pp. 166–175, 2019.

[34] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. D. Casas, F. Bressand, G. Lengyel, G. Lample, and L. e. a. Saulnier, "Mistral 7b." http://arXiv:2310.06825, 2023. Accessed: 2024-08-15.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019.

[36] A. Borji, "A categorical archive of chatgpt failures." http://arXiv:2302.03494, 2023. Accessed: 2024-08-15.

[37] B. Guo, X. Zhang, Z. Wang, M. Jiang, J. Nie, Y. Ding, J. Yue, and Y. Wu, "How close is chatgpt to human experts? comparison corpus, evaluation, and detection." http://arXiv:2301.07597, 2023. Accessed: 2024-08-15.