Code-Comment Paired Instruction Tuned LLM for Code Generation

Aadil Tajani*
atajani@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

In the realm of LLM-driven code generation, this is a novel approach to enhance dataset quality through data augmentation. Focusing on instruction tuning Salesforce's CodeT5+ with OpenAI's GPT-3.5, the limitation of conventional methods is addressed that primarily filter or generate datasets but seldom explore improving quality through augmentation. This methodology involves infusing comments into the code dataset to enrich its contextual understanding. The augmentation technique employed is described here and demonstrates their application to an existing dataset. Experimental results reveal the efficacy of this approach, showcasing improved performance compared to traditional methods barring some experimentation limitations. The findings not only contribute to the advancement of code generation tasks but also underscore the significance of quality-focused dataset augmentation in refining the capabilities of language models. This research provides valuable insights into the untapped potential of leveraging comments for enhanced instruction tuning, paving the way for future advancements in natural language understanding for code generation.

1 INTRODUCTION

The rapid evolution of natural language processing (NLP) and code generation has witnessed the emergence of powerful open-source Language Model (LLM) architectures, such as StarCoder, CodeGen, LLaMa and CodeT5+. These models undergo pre-training on extensive corpora of natural language data and GitHub repositories, followed by fine-tuning on code-specific datasets to enhance their understanding, generation, and retrieval capabilities. Despite these advancements, open-source encoder-decoder LLMs still grapple with performance gaps when compared to their closed-source counterparts or specialized encoder-only and decoder-only models [3].

Recent strides in aligning language models with explicit instructions, as exemplified by Alpaca models [8], have demonstrated notable improvements. Inspired by this, Salesforce introduced CodeT5+, a model fine-tuned with explicit code-related instructions, leading to significant performance gains across various code-related benchmarks [11] Instruction tuning on a 16B CodeT5+ model, following the Code Alpaca dataset, achieved a new state-of-the-art (SoTA) pass rate at 1 (pass@1) of 35.0%, surpassing other open-source code LLMs and closing the gap with formidable closed-source models [11].

While these developments mark substantial progress, a critical question arises: Can we further enhance the performance of these instruction-tuned models by refining the quality of the finetuning dataset itself? The present work explores the augmentation of code-specific datasets with meaningful comments to deepen the contextual understanding of natural language instructions within the code. By infusing comments into the dataset generated by large language models like GPT-3.5, the aim is to investigate if this novel approach can bridge the performance divide and bring open-source LLMs closer to the capabilities of closed-source models.

2 RELATED WORK

The field of language model-driven code generation has witnessed significant advancements, with open-source Language Models (LLMs) playing pivotal roles. However, despite their prowess, existing open-source encoder-decoder LLMs struggle to match the performance levels achieved by closed-source models or specialized decoder-only architectures for code generation as the focused downstream task here [3].

Efforts to enhance the performance of open-source LLMs have led to the exploration of instruction tuning methodologies. Aligning language models with explicit instructions, as exemplified by Alpaca models [8], has demonstrated notable improvements. This concept was further explored in the development of CodeT5+ by Salesforce, a model fine-tuned with explicit code-related instructions, showcasing substantial performance gains across various code-related benchmarks [11] and shown in Table 1.

2.1 CodeT5+ Instruction Tuning

The study conducted by Salesforce on instruction tuning CodeT5+ using the Code Alpaca dataset [1] revealed a new state-of-the-art (SoTA) pass rate at 1 (pass@1) of 35.0%. This significant performance jump, achieved through instruction tuning, positions CodeT5+ as a previous frontrunner among open-source code LLMs, challenging even larger closed-source models [11]. Notably, the study highlights the importance of leveraging curated datasets and explicit code-related instructions for enhancing the performance of code generation models.

2.2 OctoCoder: StarCoder Instruction Tuning

The literature also introduces OctoCoder, an instruction-tuned model based on StarCoder, achieving a pass@1 value of 46.2, surpassing other openly licensed models. However, despite these advancements, there remains a performance gap when compared to larger closed-source models like GPT-4 [5].

2.3 Code Llama Instruct: Llama 2 Instruction Tuning

Code Llama – Instruct enhances the utility of Large Language Models (LLMs) through instruction fine-tuning, offering improved safety and performance [7]. Llama 2 undergoes further refinement by fine-tuning on a mix of proprietary instruction data, focusing on

safety and helpfulness. The model leverages a unique family model-generated self-instruction dataset, augmenting its capabilities by generating unit tests and solutions for coding problems. The results demonstrate significant improvements in truthfulness, toxicity, and bias benchmarks, achieving enhanced performance at a moderate cost to code generation performance.

Model	Pass@1 value	Change
OctoCoder	46.2 (Base value 33.6)	+12.6
Code Llama - Instruct	41.5 (Base value 29.9)	+11.6
InstructCodeT5+16B	35.0 (Base value 30.9)	+4.1

Table 1: Performance gain on HumanEval after Instruction Tuning compared to Base on Open Source Models

3 DATASET

The foundation of language model-driven code generation hinges upon the careful selection of datasets and the strategic application of instruction-tuning methodologies. In prior research endeavors, a range of datasets for instruction-tuning approaches have been employed to enhance the performance of language models in the domain of code downstream tasks.

The quality of the dataset utilized for instruction tuning forms the cornerstone of robust and reliable model development. The CodeT5+ instruction tuning paper underscores the significance of the Code Alpaca dataset, demonstrating substantial performance gains after training on this meticulously curated dataset. Dataset creation through the generation of novel tasks using pre-trained language models contributes to the improved alignment of models with natural language instructions.

Code Alpaca Dataset: Renowned for its pivotal role in instruction tuning, the Code Alpaca dataset, curated by Chaudhary [1], has been instrumental in achieving state-of-the-art results for models like CodeT5+. With over 20,000 instruction data points generated by letting pre-trained LLMs i.e. text-davinci-003, generate novel tasks, including task instructions, inputs (if any), and expected output., this dataset contributes significantly to improving alignment with natural language instructions [11].

COMMITPACK: Introducing a novel 4TB dataset sourced from Git commits, COMMITPACK [5] serves as a rich reservoir of data for instruction tuning. The dataset's expansive scope and real-world coding examples are essential for enhancing models' generalization abilities.

Other Sources: Datasets such as **xP3x** [6], **Self-Instruct** [10], and **OASST** [4] have played crucial roles in benchmarking and training, each offering unique characteristics and challenges.

Code-Comment Augmented Dataset: This research, introduces a novel approach to dataset augmentation—comment-infused data augmentation. Acknowledging the importance of contextual information in code generation tasks, this method leverages large language models such as GPT-3.5 to generate meaningful comments strategically placed within the code. Going beyond conventional methods, the augmentation approach more crucially, enriches the dataset's quality by establishing a nuanced association between natural language tokens and specific code segments.

The Python code version of the Alpaca Set [9] is taken from Huggingface with 18.6k rows of instruction, input(if any), and an output consisting of the Python code Figure 1. OpenAI's API was then used with a prompt that instructs gpt3.5 to carefully and strategically infuse concise comments without affecting the executable code output Figure 2. The augmentation was applied to 11k rows due to cost limitations and obtained 10.3k rows because of rate-limit issues in an automated multithreaded process.

Instruction and Input for Figures 1 and 2:

Instruction: "Generate a Python code for crawling a website for a specific type of data."

Input: "website: www.example.com data to crawl: phone numbers"

Original Output

```
import requests
import re
def crawl website for phone numbers(website):
       response = requests.get(website)
       phone numbers = re.findall('\d{3}-\d{3}-\d{4}',
       response.text)
       return phone numbers
if __name__ == '__main__':
       print(crawl_website_for_phone_numbers('www.example.com
       '))
```

Figure 1: original 'output' code in alpaca dataset

Augmented Output

```
import requests
import re
def crawl website for phone numbers(website):
       # Send a request to the website
       response = requests.get(website)
       # Use regular expressions to find phone numbers in the
       response text
       phone_numbers = re.findall('\\d{3}-\\d{3}-\\d{4}',
       response.text)
       # Return the phone numbers found
       return phone numbers
if __name   == '  main  ':
       # Print the phone numbers found on the website
       print(crawl_website_for_phone_numbers('www.example.com'
       ))
```

Figure 2: comment infused 'output' code in alpaca dataset

This innovative augmentation approach is grounded in the hypothesis that supplementing the dataset with comments provides models with a more nuanced understanding of the relationship between natural language instructions and code tokens. The smaller nature of the dataset is still considerable as instruction Tuning doesn't necessarily require so much data [12].

4 EXPERIMENT

The methodology employed in this research centered around the utilization of CodeT5+ models for instruction tuning, with a deliberate

focus on the smallest model in the family, codet5p-220m-py. This selection was driven by the consideration of open-source availability and resource constraints. Codet5p-220m-py represents a manageable entry point for exploration with limited resources, ensuring feasibility in the experimental setup. The task at hand involved instruction tuning, specifically fine-tuning codet5p-220m-py using a seq2seq approach. To execute this, a Colab notebook equipped with a V100 GPU was employed. This architectural choice balanced computational efficiency with the limited resources available for the study.

The dataset, denoted as python_code_instructions_18k_alpaca [9], encompassed crucial attributes essential for the instruction tuning task. These attributes included instruction, input, and output, providing a comprehensive foundation for evaluating model performance in code generation tasks. It was strategically partitioned, with instructions and inputs serving as the source and outputs (code) as the target. Fine-tuning was executed for three epochs using mixed-precision training (fp16), a methodology aligned with the practices outlined for the InstructCodeT5+ [11].

For a comprehensive evaluation, OpenAI's HumanEval benchmark was used which is tailored for code generation by Humans. The benchmark comprises prompts and test cases for 164 executable Python problems. Evaluation metrics include pass rates at 1, 10, and 100 (pass@1, pass@10, pass@100), with pass@1 regarded as the key metric for zero-shot code generation scenarios. To assess model performance, the finetuned model was employed to generate 200 examples for each problem at a set temperature value. The base model underwent fine-tuning on both the original and augmented datasets, allowing a comparative analysis of the two augmentation methods. The study faced notable limitations, primarily rooted in resource constraints. Larger models within the CodeT5+ family were not feasible due to computational limitations. Additionally, the dataset size was reduced by 7,000 rows due to cost constraints, emphasizing the need for careful consideration of available resources in experimental design.

$$pass@k := \mathbb{E}_{\text{problems}} \left[1 - \frac{C(n-c,k)}{C(n,k)} \right]$$

Figure 3: How Pass@k chooses datapoints

In summation, the methodology crafted for this research aimed to strike a balance between model complexity, resource availability, and comprehensive evaluation metrics, providing a structured approach to investigate instruction tuning in the context of code generation.

5 RESULTS

The introduction of comments through the augmentation approach showcased an improvement in pass rates compared to the traditional non-augmented method. Pass rates at 1, 10, and 100 demonstrated a noticeable increase for different temperature values shown in Figure 4, suggesting that the contextual information provided by comments contributed to a better understanding and synthesis of natural language instructions. The comparisons are laid out for different temperature values of T0.8 in Table 2, T0.6 in Table 3, and T0.2 in Table 4 where codet5p220 is the base model, codet5p220-alpaca is the instruct tuned model on the original alpaca dataset and codet5p220-alpacacomments is the instruct tuned model on the comment augmented alpaca dataset.

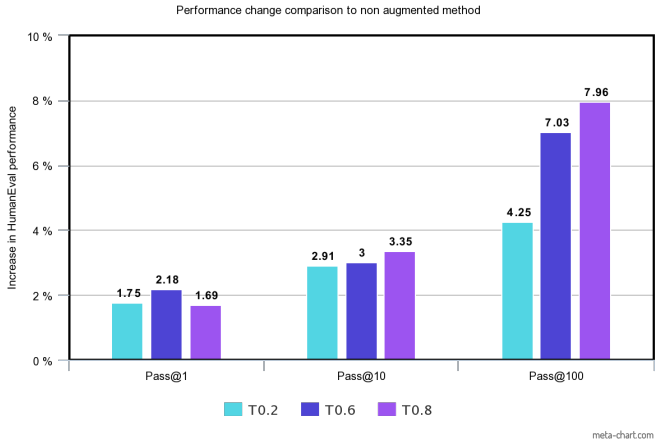

Figure 4: Performance change after Instruction Tuning on Comment Augmented Dataset compared to Original Dataset

Model	Pass@1	Pass@10	Pass@100
codet5p220	8.28	19.08	30.47
codet5p220-alpaca	4.59	13.80	22.80
codet5p220-alpacacomments	6.28	17.15	30.76

Table 2: HumanEval Performance comparison at Temperature 0.8

Model	Pass@1	Pass@10	Pass@100
codet5p220-alpaca	4.88	12.81	18.85
codet5p220-alpacacomments	7.06	16.68	25.88

Table 3: HumanEval Performance comparison at Temperature 0.6

Model	Pass@1	Pass@10	Pass@100
codet5p220-alpaca	5.64	10.24	11.56
codet5p220-alpacacomments	7.39	13.15	15.81

Table 4: HumanEval Performance comparison at Temperature 0.2

But, as seen in Table 2 there was a negative impact on the performance of both instruction-tuned models compared to the base except Pass@100 for comment-paired, exposing a key issue. Increasing epochs for finetuning was also explored bumping it from

Model	Pas@1	Pas@10	Pas@100
codet5p220-alpacacomments-3ep	6.28	17.15	30.76
codet5p220-alpacacomments-6ep	6.18	17.23	29.65

Table 5: Performace comparison for 3 and 6 epochs at T0.8 for the code-comment finetuned model

3epochs to 6epochs but no significant change was observed as seen in Table 5.

Another noteworthy observation is the decrease in Pass@1 performance on increasing the temperature value which is opposite to the increase in Pass@10 and Pass@100 values. More comparisons can be carried out for this experiment but was not possible to add here due to resource limitations. Moreover, fine-tuning challenges were observed for smaller LLMs, impacting performance as the models shown in Table 1 are comprising Billions of parameters compared to 220 million used here and smaller models are notorious for instruction tuning[2]. Thus, codet5-220m-py model demonstrated sensitivity to instruction tuning, highlighting potential limitations for models with fewer parameters. Another key issue might be the lengthy code outputs or instructions in prompts constrained space for essential context, impacting the models' ability to grasp nuanced instructions.

6 CONCLUSION

In conclusion, the exploration of comment-infused data augmentation as a novel approach to enhance instruction-tuned code generation models has yielded promising results. The introduction of meaningful comments into the dataset demonstrated a positive impact on model performance, as evident by improved pass rates at 1, 10, and 100 compared to the traditional non-augmentation method as seen in Figure 4. The findings suggest that the contextual information provided by comments contributes to a more nuanced understanding of natural language instructions, enhancing the synthesis of code.

However, several considerations and challenges emerged during the experimentation. Smaller models exhibited sensitivity to instruction tuning, indicating a need for exploration with larger models. Constraints on prompt length highlighted the importance of accommodating essential context in instruction-tuning setups. Additionally, the resource demands for full fine-tuning underscored the need for efficient approaches in resource-limited environments.

7 FUTURE WORK

This study opens avenues for future research and exploration in several directions:

Model Size and Architecture Investigate the impact of comment-infused augmentation on larger models, such as CodeT5+ 2B/6B/16B, and models with varying architectures (e.g., Llama, StarCoder). Understanding the scalability and adaptability of this approach is essential for broader applicability.

Optimizing Resource Usage Address the challenges posed by resource demands in fine-tuning by exploring more efficient approaches. This could involve optimizing memory and computation usage or exploring transfer learning strategies.

Open-Source Data and Models Explore the integration of comment-infused augmentation with open-source data and models (e.g. OctoPack and Llama generated Dataset), ensuring ethical and legal compliance. This step is crucial for the broader adoption and applicability of the proposed method.

Broader Evaluation Metrics Expand the evaluation metrics to include a more comprehensive assessment of model performance other than HumanEval, considering aspects such as code readability, coherence, and adherence to coding conventions.

ACKNOWLEDGMENTS

I extend my heartfelt gratitude to Dr. Bowen Xu for guiding and imparting invaluable knowledge throughout the course on Generative AI for Software Engineering. Dr. Xu's insightful discussions on various tasks under Software Engineering improved by Large Language Models (LLMs) have been instrumental in broadening my understanding of this dynamic field.

I would like to express my sincere appreciation to the class members for their thoughtful suggestions and constructive feedback, particularly during the initial presentation of the idea. The collaborative and intellectually stimulating environment fostered in the class has been crucial in refining and shaping the trajectory of this research.

This work has benefited greatly from the mentorship and expertise provided by Dr. Bowen Xu, and I am grateful for the opportunity to explore and contribute to the intersection of Generative AI and Software Engineering under such guidance.

REFERENCES

[1] Sahil Chaudhary. 2023. CodeAlpaca: An Instruction-Following Llama Model for Code Generation. https://github.com/sahil280114/codealpaca.

[2] Akash Desai. 2023. *Optimizing LLMs: A Step-by-Step Guide to Fine-Tuning with PEFT and QLORA*. https://blog.lancedb.com/optimizing-llms-a-step-by-step-guide-to-fine-tuning-with-peft-and-qlora-22eddd13d25b

[3] S. Guo, N. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. In *ACL (1)*. Association for Computational Linguistics, 7212–7225.

[4] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, and et al. 2023. OpenAssistant Conversations – Democratizing Large Language Model Alignment. *arXiv preprint arXiv:2304.07327* (2023).

[5] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. 2023. OCTOPACK: Instruction Tuning for Large Language Models. https://github.com/bigcode-project/octopack. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*. The BigCode Project. Contact: n.muennighoff@gmail.com.

[6] Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M. Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, and et al. 2022. Crosslingual Generalization through Multitask Finetuning. *arXiv preprint arXiv:2211.01786* (2022).

[7] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askel, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

[8] I. Taori, T. Gulrajani, Y. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T.B. Hashimoto. 2023. Stanfordalpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

[9] Hugging Face Tarun. 2023. *Python Code Instructions 18k Alpaca Dataset.* https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

[10] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-Instruct: Aligning Language Model with Self-Generated Instructions. *arXiv preprint arXiv:2212.10560* (2022).

[11] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. 2023. CodeT5: Unified pre-trained models for programming

language understanding and generation. https://github.com/salesforce/CodeT5/tree/main/CodeT5+. Equal contribution.

[12] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, and et al. 2023. Lima: Less is More for Alignment. *arXiv preprint arXiv:2305.11206* (2023).