# AI-Generated Python Code Documentation: A Case Study

Levi Shafter, Ryan Martel, Kevin Zhu

December 6, 2024

## Abstract

¡++¿

## 1 Introduction

In an era of LLM-based revolution, many are exploring the endless possibilities of using artificial intelligence to aid in software development. This comes as no surprise considering the astounding need for software in nearly every aspect of modern industry. [5] With a dramatic increase in the need for software, some of the hottest topics being explored are the potential of enhancing the software development process, creating new tools as well as enhancing the old, and even automating certain aspects of writing code (See Section 3: Related Work).

In this paper, we detail our experience with training a model using machine learning for the purpose of generating Python documentation. We discuss other similar works, compare our results, and discuss applications and potential improvements as well as viable future work. With so much recent progress in the field of AI [8], models like ours might drastically change the way we think about and approach software development.

## 2 Related Work

Several recent studies and developments in the field of artificial intelligence and software development have highlighted significant advancements. These include the use of ML in resolving code review comments [7], the introduction of IDE plugins for AI-assisted development [1], innovations in AI-powered fuzzing [3, 6], research into the out-of-distribution generalization of pre-trained language models [4], efforts in automated documentation with ChatGPT [2], and the enhancement of large language models for multi-modal research synthesis [9].

## 3 Method

For building our model, we used the following Python modules:

- datasets
- torch
- transformers
- accelerate
- evaluate

For our dataset, we used the google/code_x_glue_ct_code_to_text dataset available here: https://huggingface.co We made use of the RobertaTokenizer for preprocessing; specifically we began with the pretrained Salesforce/codet5-small model for some starting weights. Due to time constraints, we decided to skip the examples in the dataset so as to make our training time feasible on our hardware. To pad tokens, we used a value of -100 for our labels so as to avoid reduced model performance.

We selected a batch size of 16 for our training dataset, and a batch size of 8 for both our validation and testing datasets so as to avoid running out of memory on our selected hardware. We then used the T5 model from the Hugging Face Transformers library to train our model on Colorado State University's computer science department machines. We decided to go with Roberta Tokenizer since it creates byte-level Byte-Pair-Encoding, suitable for our base model. Additionally, we used the accelerate library to speed up training. Finally, we used the evaluate library to evaluate our model.

We unfortunately found that higher values used as hyper parameters, particularly those used in the number of epochs, drastically increased our training time. With our limited time, we decided to move forward with a proof-of-concept using a number of epochs equal to one with zero warmup steps being used. Using CUDA, we trained our model primarily on our available GPUs using the AdamW optimizer.

# 4 Results

¡++¿

# 5 Discussion

¡++¿

# 6 Conclusion

¡++¿

# 7 Future Work

Our greatest challenges with this experiment were definitely a lack of computational resources and time. We would like to explore the potential of using a larger dataset, as well as training for more epochs. We would also like to explore the potential of using a larger model, such as the Salesforce/codet5-large model. We would also like to explore the potential of using a different optimizers, such as the SGD optimizer. With more time and computational power, we might be able to find a convergence on a more optimal solution using SGD. We might also try SGD with momentum if we still find ourselves limited on resources in the future to help reach a balance between optimal convergence and resource usage.

We would also like to perform more tests and analysis on our model to better understand its performance and potential improvements. This would help us compare our model to other works and evaluate where our model needs improvement, as well as where it really excels. Perhaps performing this analysis would help us and other researchers develop better and better models for the purpose of intelligent Python documentation generation.

# References


[1] Qodo AI. Ide plugin for ai-assisted development, 2024. Accessed: 2024-12-06.

[2] Awekrx. Autodoc-chatgpt: documentation generation using chatgpt, 2024. Accessed: 2024-12-06.

[3] Google Security Blog. Ai-powered fuzzing: breaking bug hunting barriers, August 2023. Accessed: 2024-12-06.

[4] Tianle Chen, Mingjie Li, and Shengyu Zha. Towards understanding out-of-distribution generalization of pre-trained language models. *arXiv*, May 2023.

[5] Emerald Group Publishing Limited. Corporate social responsibility and firm value: the moderating role of governance mechanisms. *Advances in Corporate Governance Research*, 16, 2019. Accessed: 2024-12-06.

[6] CSO Online. What is ai fuzzing and why it may be the next big cybersecurity threat, 2024. Accessed: 2024-12-06.

[7] Google Research. Resolving code review comments with ml, 2023. Accessed: 2024-12-06.

[8] Feifei Shi, Huansheng Ning, Wei Huangfu, Fan Zhang, Dawei Wei, Tao Hong, and Mahmoud Daneshmand. Recent progress on the convergence of the internet of things and artificial intelligence. *IEEE Network*, 34(5):8–15, 2020.

[9] Kyle M. Wiggers, James Smith, and Xinyi Zhao. Enhanced large language models for multimodal research synthesis. *arXiv*, July 2023.