Write me this Code: An Analysis of ChatGPT Quality for Producing Source Code

Konstantinos Moratis, Themistoklis Diamantopoulos, Dimitrios-Nikitas Nastos, and Andreas Symeonidis
kmoratis@ece.auth.gr,thdiaman@issel.ee.auth.gr,diminast@ece.auth.gr,symeonid@ece.auth.gr
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Thessaloniki, Greece

ABSTRACT

Developers nowadays are increasingly turning to large language models (LLMs) like ChatGPT to assist them with coding tasks, inspired by the promise of efficiency and the advanced capabilities they offer. However, this raises important questions about the ease of integration and the safety of incorporating these tools into the development process. To investigate these questions, this paper examines a set of ChatGPT conversations. Upon annotating the conversations according to the intent of the developer, we focus on two critical aspects: firstly, the ease with which developers can produce suitable source code using ChatGPT, and, secondly, the quality aspects of the generated source code, determined by the compliance to standards and best practices. We research both the quality of the generated code itself and its impact on the project of the developer. Our results indicate that ChatGPT can be a useful tool for software development when used with discretion.

CCS CONCEPTS

• **Software and its engineering** → **Reusability**; **Open source model**; *Software defect analysis*; Software libraries and repositories.

KEYWORDS

Code Generation, Code Quality, ChatGPT, Large Language Models

ACM Reference Format:
Konstantinos Moratis, Themistoklis Diamantopoulos, Dimitrios-Nikitas Nastos, and Andreas Symeonidis. 2024. Write me this Code: An Analysis of ChatGPT Quality for Producing Source Code. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643991.3645070

1 INTRODUCTION

Lately, large language models (LLMs) have significantly influenced the software development landscape. LLMs are pretrained on vast corpora of text and source code from diverse sources, enabling them to respond effectively to a wide range of software development-related queries [6, 9]. As a result, more and more developers are turning to tools like ChatGPT for various code-related challenges, including e.g. writing specific code snippets [2, 3], identifying and resolving bugs [8], even generating tests [1].

In this context, developers often request ChatGPT to write code, which is then integrated into their projects. Although ChatGPT can streamline certain tasks, it is important to consider whether this approach is practical and safe in terms of quality. Considering practicality, developers might find themselves making extended dialogues with ChatGPT, refining multiple prompts to achieve the desired code output. This process can be time-consuming and may not always lead to ready-to-use code. Code quality is equally (if not more) important; there's a risk that the generated code may not always adhere to best practices or fit seamlessly into existing projects. Thus, developers may have to assess and even alter the generated code before committing it to their project.

In this paper, we employ DevGPT [7], a dataset that includes developer interactions with ChatGPT as well as links to commits made after each conversation. Upon annotating the conversations according to the intent of the developers (e.g. writing new code, fixing a bug, etc.), we focus on the following research questions:

RQ1: How easy is it to direct ChatGPT towards writing the code for a development scenario?

This research question is meant to assess the practicality of using ChatGPT for generating source code. To do so, we employ useful statistics, such as the number of prompts required to guide the model towards producing the expected result.

RQ2: Is the code proposed by ChatGPT of high quality?

This research question investigates the quality of the code proposed by ChatGPT, assessed through violations' analysis to ensure adherence to coding principles.

RQ3: Does adding code written by ChatGPT improve the existing code in terms of quality?

This research question further assesses the impact of integrating ChatGPT-generated code on the quality of the project. To perform this analysis, we extract the number of code violations before and after introducing the relevant commit proposed by ChatGPT.

2 METHODOLOGY

Our methodology is shown in Figure 1. We use the commits of the dataset provided by DevGPT [7] as they are capable of capturing all three of our research questions. The dataset is offered in MongoDB format and includes full ChatGPT conversations, along with linked commits. An example conversation is shown in Figure 2, where the user wants to build a specific component and thus converses with

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04
https://doi.org/10.1145/3643991.3645070

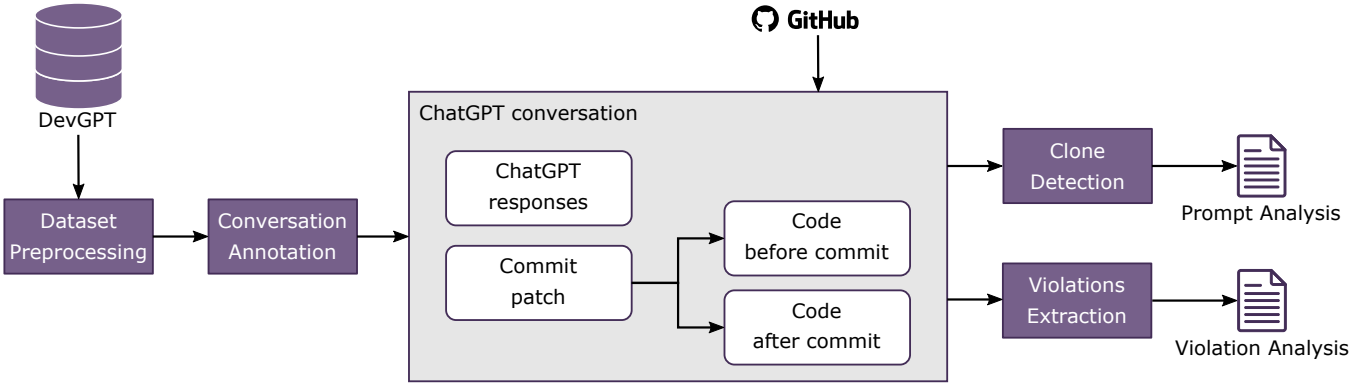

Figure 1: Architecture Overview

ChatGPT in a series of prompts. When ChatGPT provides the final code, the developer copies it and commits it in the project.

The dataset is initially preprocessed to fix any inconsistencies, duplicate entries, etc. The next step is to annotate the conversations according to their purpose (i.e. what the developer asks from ChatGPT). After that, the data are enriched by retrieving their source code contents from GitHub. Finally, we apply two kinds of analysis, one including code clone detection to extract useful statistics, and one including the extraction of code violations. The former is used to answer RQ1, while the latter is used to answer RQ2 and RQ3. These steps are analyzed further in the following paragraphs[1].

2.1 Dataset Preprocessing

Upon loading the DevGPT dataset (snapshot 20230914, which was the latest when the analysis was performed), we filter the entries to keep only those with status code equal to 200. Moreover, we keep only entries that contain at least one prompt and one response that includes a ChatGPT generated code blocks. To ensure data consistency, any entries containing non-UTF-8 characters in a conversation's prompt or answer or in a source text (e.g. commit message) are removed. Duplicate entries and links were also removed, reducing the initial 571 conversations to 453.

Finally, another necessary step is the identification of the primary programming language of each conversation. To obtain this information, the types of generated code blocks were analyzed and the language with the highest frequency within each conversation was considered the primary language. We chose JavaScript as a proof of concept for our analysis, given its majority presence in the dataset (58%), while other popular languages were underrepresented (e.g. Java - 4%) or were relevant to few repos (e.g. Python).

2.2 Conversation Annotation

The next step is to annotate each conversation according to the user's intent. To do so, we reviewed all conversations and classified them into five categories, shown in Table 1, along with their number of conversations (7 conversations were discarded, as they were not relevant to code, e.g. asking for help when writing a CV).

[1]The steps required to reproduce our methodology can be found online in the GitHub repository https://github.com/AuthEceSoftEng/msr-devgpt

Table 1: Conversation Categories of the Dataset

Category	Description	# Convs.
Write me this code	ChatGPT is asked to produce a piece of code (text instructions usually provided)	47
Improve this code	ChatGPT is given a piece of code snippet and is asked to improve it (code review)	334
Fix this issue	ChatGPT is given a piece of code and an error trace and is asked to fix it	56
Example usage	ChatGPT is asked to provide an example of using a specific API function/object	6
Explain this code	ChatGPT is asked to explain the functionality of a specific piece of code	3

We use the first two categories, as these offer clear examples of ChatGPT producing or altering source code that is integrated by the developer. E.g., Figure 2 depicts a "Write me this code" scenario.

2.3 Clone Detection

To identify cases where users have copied code from ChatGPT into their source code, we detect similarities between the committed files and ChatGPT-generated code blocks. Since the files of commits are not included in the dataset, for each commit we retrieved this information from the GitHub API. After that, we applied code clone detection to detect copy-pasted ChatGPT responses.

Although there are several tools for code clone detection, they usually rely on compilable code. Instead, the code blocks generated by ChatGPT, while often insightful, may not always be syntactically correct or compilable. This is a common issue of LLMs like ChatGPT, as they prioritize coherence over strict language-specific syntax. Furthermore, the generated code blocks may only represent code fragments, which are not compilable without the rest of the file. As a result, for our analysis, we employed the Simian tool [5], using its text similarity option that compares texts line by line and considers two texts as similar when they have at least 2 identical lines.

To determine whether any of the generated blocks were copied, we iterated all blocks in reverse order (i.e. starting from the last one)

Basho Object for Sumo

Anonymous
I am building a JavaScript application to simulate a game based on sumo wrestling. The game includes multiple instances called 'waves', where each wave starts at a different point in time. Within each wave, players select a wrestler for each basho (tournament). I need to build a 'Basho' object that represents a basho. Each Basho should contain a dictionary mapping from player names to their picks for this basho.

ChatGPT
This is a basic object-oriented design that might suit your needs. Based on the complexity of your game, you might want to consider additional functionalities, like removing or changing a pick, adding more data to the picks, etc. You could add methods to handle such functionalities.

javascript Copy code
```
class Basho {
    constructor(id) {
        this.id = id;
        this.picks = {};
    }
...
```

Anonymous
you didn't incorporate the wave concept which feels important

ChatGPT
Apologies for the oversight. You could represent this with a "Wave" class that contains multiple Bashos. Here's an example:

javascript Copy code
```
class Wave {
    constructor(id) {
        this.id = id;
        this.bashos = [];
    }
...
```

Anonymous
no you didn't tell the basho its associated wave

ChatGPT
In this case, we can add a waveId property to the Basho class to represent the wave that the basho belongs to.

javascript Copy code
```
class Basho {
    constructor(id, ...  ) {
        this.id = id;
        this.waveId = waveId;
        this.picks = {};
    }
...
```

Figure 2: Example Conversation

and compared each block to the committed code until a match was detected. After that, we extract statistics like the number of prompts before the generation of the copied code for each conversation.

2.4 Violations Extraction

To examine the potential impact on code quality from integrating ChatGPT-generated code, we used PMD [4] to detect three categories of violations: (a) violations relevant to standard practice, such as avoiding the use of globals (Best Practices), (b) violations relevant to code readability, such as not using braces for if statements (Code Style), and (c) violations that can lead to bugs, such as using trailing comma when declaring an array (Error Prone).

Concerning the "Write me this code" scenario, our analysis is performed on all code blocks generated by ChatGPT, to evaluate the quality of its code. Apart from that, we further assess the "Improve this code" scenario by executing PMD twice for each conversation, one for the code before the commit (and thus before ChatGPT assistance) and one for the code after the commit. This way, we are able to better determine whether using ChatGPT to change/improve one's code may have pitfalls relevant to code quality.

3 ANALYSIS

Upon having built our methodology, we are now going to investigate the research questions posed in the introduction.

RQ1: How easy is it to direct ChatGPT towards writing the code for a development scenario?

This research question is relevant to the "Write me this code" scenario, where developers ask ChatGPT to generate pieces of code which are then committed to their project with minimal changes. To achieve that, they often have to provide instructions and feedback during consecutive conversation cycles until the code meets their requirements. Figure 2 depicts such a scenario where the developer asks for a 'Basho' object, which is subsequently updated using 2 more prompts to include a 'Wave' concept. As already mentioned, the number of prompts needed before the code is copied can be an indication of whether the communication between the developer and ChatGPT is clear enough to achieve high efficiency. According to the extracted statistics (Figure 3), in most cases the conversation required less than 5 prompts, which could be considered an effective interaction (of course, this is highly case specific, and it would be better to be evaluated by means of a survey). There are also conversations with greater length, however manual examination showed that these are usually due to the developers asking for multiple pieces of code in the same conversation.

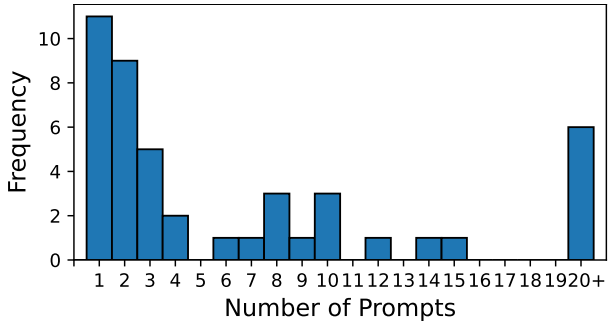

Figure 3: Histogram depicting the number of prompts before copying the code provided by ChatGPT

RQ2: Is the code proposed by ChatGPT of high quality?

The quality of the generated source code is assessed for the "Write me this code" scenario. In specific, for any code block proposed by ChatGPT we measure the number of violations detected. In total 59 violations were found in 144 code blocks analyzed. The resulting frequencies are shown in Figure 4. Although most code blocks contain no violations, one may notice that there is a significant number of code blocks (approximately 1 out of 4) that have one or more violations. However, upon further investigation, we found that 50.8% of these violations belong to the Best Practices category and 37.3% of them belong to the Code Style category. This means that only 11.9% of the violations may pose significant harm if copied by the user (as they belong to the Error Prone category).

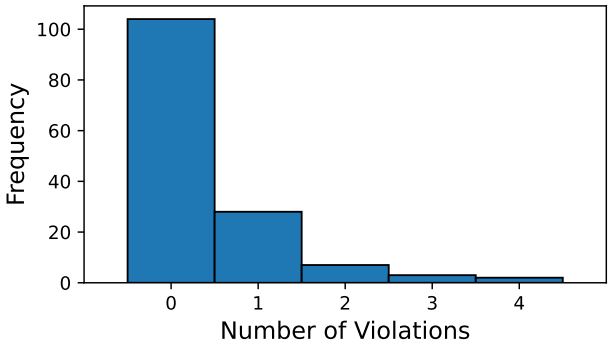

Figure 4: Histogram depicting the number of violations found in JavaScript code proposed by ChatGPT

RQ3: Does adding code written by ChatGPT improve the existing code in terms of quality?

Apart from generating high-quality code, an important question is whether ChatGPT actually improves existing code provided by developers. To assess each "Improve this code" conversation, we have computed the difference between the number of violations in the final (ChatGPT-improved) code minus the number of violations in the original (human-written) code. The results for conversations that led to change in number of violations are shown in Figure 5.

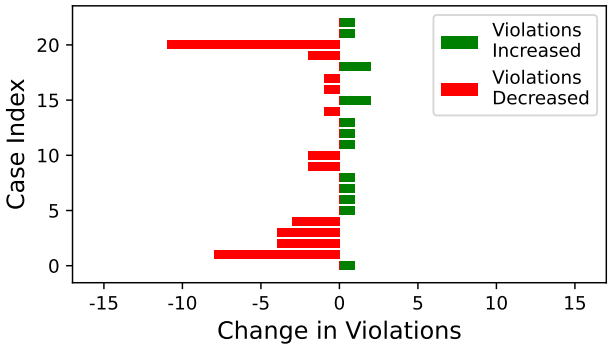

Figure 5: Bar chart depicting the impact of adding code generated by ChatGPT on the quality of the source code

In the majority of cases, there was a decrease in the total violations after copying the generated code. Although there are cases where ChatGPT introduced previously non-existing violations, these were limited to one or two violations that were almost always in Best Practices and Code Style categories. On the other hand, certain cases of ChatGPT improving existing code resulted in greatly decreasing the number of violations. This suggests that using ChatGPT can indeed improve the quality of the existing code.

4 DISCUSSION

Our findings suggest that integrating ChatGPT into software development, when done appropriately, can be quite effective. However, developers should exercise caution as the generated code might not always meet the required quality standards. Although these findings are supported by our analysis, we also recognize certain limitations and threats to validity, which we discuss in this section.

First of all, we must note that the dataset used in this analysis is inherently optimistic, as it exclusively contains instances of successful interactions between developers and ChatGPT. Consequently, it is not suitable for evaluating the overall effectiveness of ChatGPT as a software development assistant. This is why we focused on the length of the conversations, examining the ease of arriving at useful responses, instead of the responses themselves.

Concerning the quality of code blocks, we note that code quality assessment is a non-trivial exercise that may extend beyond the identification of the violations. To ensure a more comprehensive evaluation, a thorough quality analysis should incorporate metrics such as cyclomatic complexity and/or consider additional characteristics like reliability and security. In fact, such an analysis would truly be well-founded if it took into account entire projects, as a generated code block may influence several different code segments.

Moreover, expert examination could also be used to validate the claim of improved code quality. To further elaborate on this idea, it would be interesting to perform a comparative analysis between code written by developers and functionally equivalent code by ChatGPT. This way, we could truly determine the quality of ChatGPT-generated code and further support our findings.

Finally, it is worth noting that our analysis did not explicitly address instances where multiple topics were discussed within a single conversation. However, through manual investigation, we observed that such cases were not common in the dataset.

5 CONCLUSIONS

In this work, we investigated the efficiency and safety of using ChatGPT to assist software development. Our findings indicate that ChatGPT is easily directed towards writing useful code and generally produces code of high quality. Nevertheless, there are also cases where the generated code includes violations, indicating the need to examine it before copying it into one's own project.

Future work lies in several directions. We could explore additional scenarios, such as 'Fix this issue' and/or extend our analysis to other languages (e.g. Python) or even further analyze the text of the conversations. Another interesting idea would be to compare the code generated from ChatGPT with functionally equivalent code from Stack Overflow, with the aim of assessing the quality of each source towards assisting software development.

REFERENCES

[1] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel C. Desmarais. 2023. Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing. arXiv:2308.16557 [cs.SE]

[2] Zhenlan Ji, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2023. Benchmarking and Explaining Large Language Model-based Code Generation: A Causality-Centric Approach. arXiv:2310.06680 [cs.SE]

[3] Vijayaraghavan Murali, Chandra Maddila, Imad Ahmad, Michael Bolin, Daniel Cheng, Negar Ghorbani, Renuka Fernandez, and Nachiappan Nagappan. 2023. CodeCompose: A Large-Scale Industrial Deployment of AI-assisted Code Authoring. arXiv:2305.12050 [cs.SE]

[4] PMD. 2021. PMD Source Code Analyzer. https://pmd.github.io/ [last accessed November, 2023].

[5] Harris S. 2015. Simian – similarity analyser. http://www.harukizaemon.com/simian [last accessed November, 2023].

[6] Ensheng Shi, Fengji Zhang, Yanlin Wang, Bei Chen, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2023. SoTaNa: The Open-Source Software Development Assistant. arXiv:2308.13416 [cs.SE]

[7] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

[8] Yuwei Zhang, Zhi Jin, Ying Xing, and Ge Li. 2023. STEAM: Simulating the InTeractive BEhavior of ProgrAMmers for Automatic Bug Fixing. arXiv:2308.14460 [cs.SE]

[9] Li Zhong and Zilong Wang. 2023. Can ChatGPT replace StackOverflow? A Study on Robustness and Reliability of Large Language Model Code Generation. arXiv:2308.10335 [cs.CL]