Evaluating the Effectiveness of ChatGPT in Improving Code Quality

Shanal Divyansh*, Pranjal Apoorva*, Suraj Sanjay Singh*, Anita Ershadi, Hiral Makwana, Eman Abdullah AlOmar
Stevens Institute of Technology, Hoboken, New Jersey, USA
{sdivyans,papoorva,singh71,aershadi,hmakwan3,ealomar}@stevens.edu

***Abstract*—Code refactoring is a crucial process in software development that helps improve the quality and maintainability of code without changing its functionality. Although code refactoring is widely recognized as an essential practice, measuring its impact on code quality is challenging. This paper investigates the impact of ChatGPT, on code quality. The study focuses on four key metrics: cyclomatic complexity, cognitive complexity, code smells, and time debt, using `SonarQube` to assess code quality and identify potential issues. The original dataset of Python code is compared with the refactored dataset to evaluate the effectiveness of ChatGPT in improving code quality. The results demonstrate that ChatGPT's refactoring efforts have led to improvements in the quality of the codebase. The refactored code exhibited lower complexity values, fewer code smells, and reduced time debt, highlighting ChatGPT's success in addressing significant issues that can cause system failures and performance issues. The study emphasizes the potential benefits of using ChatGPT for code refactoring, which can significantly benefit software development efforts by improving code quality and reducing development time.**

***Index Terms*—quality, ChatGPT, LLMs**

I. Introduction

Software development is a complex and ever-evolving field, where code quality is critical to software success. Code refactoring is essential for improving code quality by restructuring and optimizing existing code without changing its external behavior [1], [2]. In recent years, there has been a surge of interest in automated code refactoring tools, powered by advances in machine learning and natural language processing [3], [4], [5], [6]. ChatGPT is one such tool, which is a significant language model trained on the GPT-3.5 architecture and has shown effectiveness in generating high-quality code.

ChatGPT can contribute to improving code quality by aiding in the refactoring process. By leveraging its natural language processing capabilities, ChatGPT can identify code smells or areas in the codebase that need improvement. It can then suggest or implement refactoring techniques to enhance the quality and maintainability of the software.

Measuring the impact of ChatGPT-driven refactoring can be done using metrics such as cyclomatic complexity, cognitive complexity, and code smells. Cyclomatic complexity allows developers to determine the complexity of a software component and identify areas with high defect rates. By utilizing ChatGPT to refactor these high cyclomatic complexity value components, developers can decrease their complexity, leading to fewer defects and an overall improvement in code quality. Using cyclomatic complexity as a metric for software testing offers several benefits. Firstly, it can help identify high-risk components likely to have a high defect rate or be challenging to test and maintain. Secondly, by identifying modules with high cyclomatic complexity values, developers can focus on refactoring or redesigning those modules to reduce their complexity and improve maintainability. This leads to improved code quality and reduced likelihood of defects. Additionally, reducing the complexity of software components makes them easier to understand and maintain over time, enhancing their maintainability. Finally, cyclomatic complexity is widely used in criticality prediction and static code analysis tools, making it a valuable metric for assessing the overall quality of software components [7]. Cognitive complexity is another metric that measures the ease of understanding a code. Refactoring assisted by ChatGPT can help simplify the code, reduce cognitive complexity, and result in more maintainable and easier-to-understand code. While cyclomatic complexity focuses on the code's structure, cognitive complexity considers both internal structures and external inputs/outputs. Measuring cognitive functional size provides a stable and practical software complexity measurement that helps explain the fundamental nature of software complexity during the design, implementation, or maintenance phases of software engineering. Unlike cyclomatic complexity, cognitive functional size is more robust and independent of language or implementation [8]. Additionally, cognitive complexity is closely related to code maintainability, as it measures software's cognitive and psychological complexity as an intelligent human artifact. Therefore, measuring cognitive functional size can help identify complex parts of the code that may be difficult to maintain, enabling developers to improve their designs and make them more maintainable.

Further, reducing code smells indicates the effectiveness of the refactoring process [9]. By addressing various code smells identified by ChatGPT, developers can significantly improve the software's quality while making future changes more manageable. Code smells indicate potential problems in the source code that can make it harder to maintain over time. For example, duplicated code can make it harder to update the code in the future [10], while long methods can make it harder to understand and modify the code [11]. By addressing code

*These authors contributed equally to this work.

smells early on, developers can improve the maintainability of their software. Refactoring the source code to remove code smells can make it easier to understand, modify, and extend over time. This can help reduce the maintenance cost and improve the software's overall quality. In addition, identifying and addressing code smells can help to prevent bugs and other issues from arising in the future by improving the overall design of the source code. This can lead to more stable and reliable software that is easier to maintain over time [12].

Time debt is the accumulation of technical debt over time, which can occur when software is developed quickly without proper planning, design, or testing. Many factors can contribute to technical debt, including taking shortcuts, using outdated technologies or libraries, or failing to maintain and update code properly. Time debt is crucial in restructuring code since it can lead to various issues harmful to software development initiatives. Time debt can lead to increased maintenance costs, reduced developer productivity, and decreased software quality, impacting the ability to deliver software products on time and within budget. Refactoring reduces time debt by improving code quality, which makes it easier to maintain and update over time. This can lead to cost savings, increased productivity, and superior software quality, resulting in a better overall development process and better results [13], [6].

In this study, we investigate the impact of ChatGPT on code quality by experimenting with how code quality is affected when code is refactored using this tool. Specifically, we focus on four key code quality metrics: cyclomatic complexity, cognitive complexity, code smells, and time debt. To measure these metrics, we use `SonarQube`, a popular static analysis tool used to assess code quality and identify potential issues [14]. To do so, we compare the original dataset of Python code with the refactored dataset to evaluate the effectiveness of ChatGPT in improving code quality across these metrics. The results of this study can help inform software developers and researchers about the potential benefits and challenges associated with using ChatGPT for code refactoring and its impact on code quality, as measured by `SonarQube`.

II. Problem Statement

Code refactoring is a crucial process in software development that helps improve the quality and maintainability of code without changing its functionality. Although code refactoring is widely recognized as an essential practice, measuring its impact on code quality is challenging [15]. This study aims to investigate the effectiveness of code refactoring in improving code quality using `SonarQube` on Cyclomatic Complexity, Cognitive Complexity, Code Smell, and Time Debt metrics. Specifically, the study will focus on how ChatGPT, a language model trained on the GPT-3.5 architecture, can improve code quality after refactoring a dataset of Python codes. The study aims to answer the following Research Questions (RQs):

- **RQ1.** How does ChatGPT affect the Cyclomatic Complexity after refactoring a dataset of Python code?
- **RQ2.** How does ChatGPT affect the Cognitive Complexity after refactoring a dataset of Python code?
- **RQ3.** How does ChatGPT affect the Code Smells after refactoring a dataset of Python code?
- **RQ4.** How does ChatGPT affect the Time Debt after refactoring a dataset of Python code?

By answering these questions, this study will provide insights into the effectiveness of code refactoring in improving code quality and reducing technical debt. The findings of this study can be helpful for software developers, managers, and researchers interested in improving software quality through code refactoring.

III. Study Design

This research aims to evaluate the impact of ChatGPT on code quality metrics, specifically Cyclomatic Complexity, Cognitive Complexity, Code Smell, and Time Debt, by comparing the metrics of "Original Files" and "Refactored Files." This provides valuable insights into the impact of ChatGPT on code quality metrics, which can inform the development of AI-assisted software development tools and improve code quality in software development. Our study methodology includes the following steps:

- **Phase 1:** We utilized an existing dataset containing around 10,000 Python files [16]. This dataset is used as the basis for our analysis.
- **Phase 2:** Using a JavaScript script we created, we have extracted each file from the above CSV file into separate Python files called the "Original Files." These files are used as the baseline for our analysis.
- **Phase 3:** We used the ChatGPT API in the JavaScript script to send each previously generated Original Files to be refactored and stored as independent Python files called "Refactored Files." Then we uploaded the "Original Files" and "Refactored Files" to `SonarQube`, which provided us with metrics for the Cyclomatic Complexity, Cognitive Complexity, Code Smell, and Time Debt for all of the Python files. Then, we compared the "Original Files" metrics and "Refactored Files" to see how ChatGPT affected code quality. We examined how ChatGPT affects Cyclomatic Complexity, Cognitive Complexity, Code Smell, and Time Debt. We also analyzed to see if there were any significant differences in the metrics for "Original Files" and "Refactored Files." We also used visualizations, such as charts, to present the findings clearly and concisely.

We obtained a dataset of 10,000 Python submissions from Stepik and Hyperskill platforms, described in [16]. The dataset included essential items such as date, user, and the raw Python code. We only kept the raw data during preprocessing and discarded other details. It is important to note that we have manually tested the files with large classes during this phase. We found that when feeding large programs broken into multiple parts, ChatGPT sometimes exhibited a phenomenon known as "hallucination." By "hallucination," we mean that ChatGPT

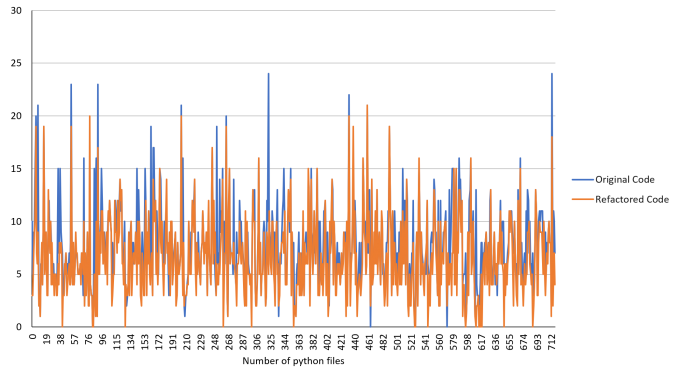


Figure 1: Cyclomatic Complexity by SonarQube - before and after refactoring.

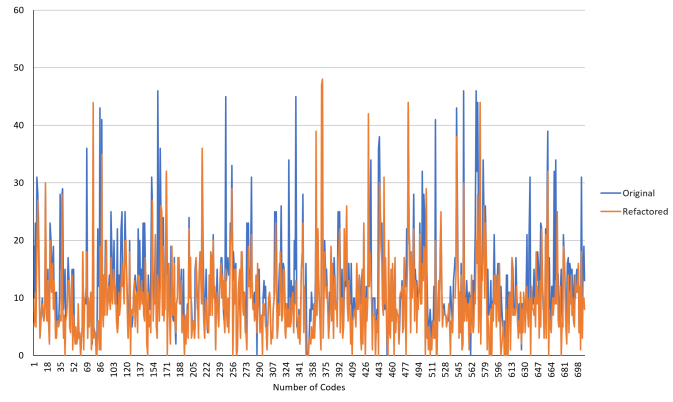


Figure 2: Cognitive Complexity by SonarQube - before and after refactoring.

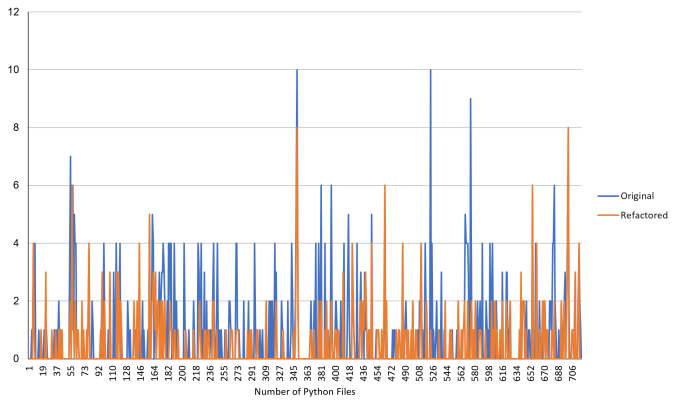


Figure 3: Code Smell by SonarQube - before and after refactoring.

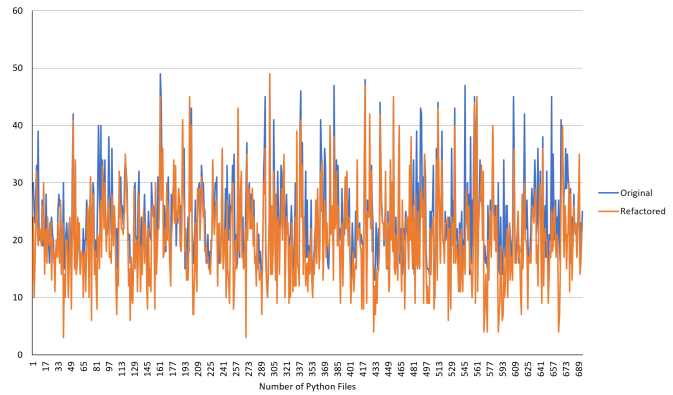


Figure 4: Lines of code by SonarQube - before and after refactoring

generated unrelated or inconsistent responses with the given context. This observation is in line with a recent study that mentioned ChatGPT is susceptible to "hallucination" when interpreting code semantic structures and fabricating non-existent facts [17]. This limitation prompted us to restrict the length of the programs we fed into ChatGPT. Thus, we selected files with lines of code (LOC) more than 20, and ended up with 1032 files. Our next step was to split the large combined CSV file containing all the Python codes into individual files. We first added the keyword "End-" after each code in the merged file to achieve this. We then imported the file into our node environment and split the code using the "End-" keyword, writing each resulting code into an individual file.

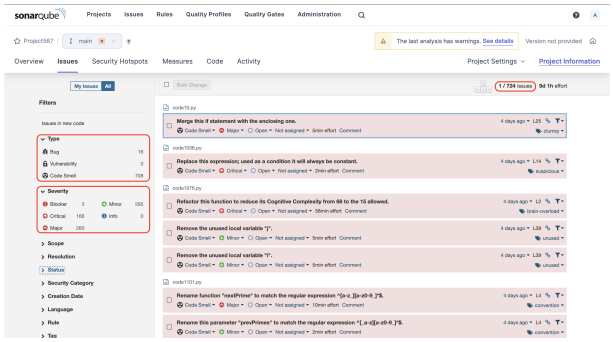
We then developed a program using Node and JavaScript, which read the individual files and used ChatGPT's API to feed each code to refactor. We provided a code fragment to ChatGPT and asked it to perform the refactoring. We saved the output from ChatGPT into each file, allowing us to have the original and refactored files. We then utilized `SonarQube` to run the analysis on both the original code files and the refactored code files. This gave us quantifiable metrics such as cognitive complexity, cyclomatic complexity, and code smells. These metrics were used in our final analysis to compare the before-and-after effects of the code refactoring to determine if ChatGPT had improved the code quality.

To summarize, we obtained a dataset of Python submissions. We used a combination of Node, JavaScript, ChatGPT, and `SonarQube` to refactor the code and assess the impact of the refactoring on the code quality.

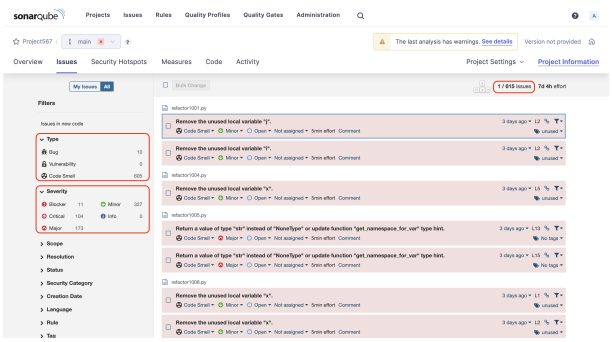
IV. Experimental Results

A. RQ1: How does ChatGPT affect the Cyclomatic Complexity after refactoring a dataset of Python code?

Figures 1 and 7a show the Cycolmatic Complexity before and after refactoring. Upon conducting our analysis, it was observed that the initial code exhibited a median cyclomatic complexity of 7, whereas the refactored code demonstrated a value of 6. This indicates that the refactored code is less intricate and more manageable, making it simpler to comprehend and maintain than the original code. The reduction in cyclomatic complexity indicates that the refactoring procedure has been successful in simplifying the code and enhancing its overall quality, resulting in an improved control flow and readability, and we can make the following observations: (1) Generally, the cyclomatic complexity of the refactored code is lower than the original code. This indicates that the refactored

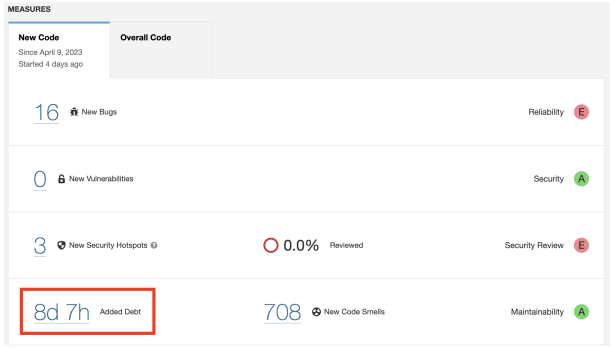


(a) Code Smell - Before

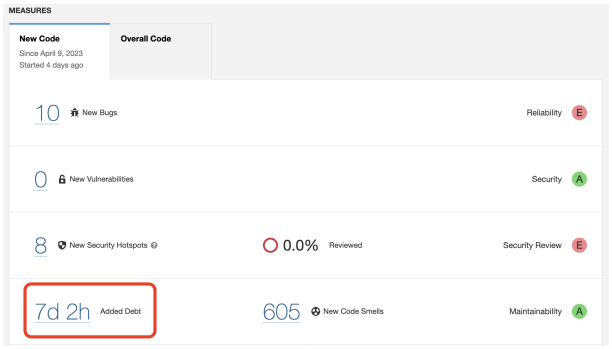


(b) Code Smell - After

Figure 5: Code smell in the original and refactored code.



(a) Time debt - Before



(b) Time debt- After

Figure 6: Time debt in the original and refactored code.

code is simpler, easier to maintain, and potentially less error-prone. Lower complexity scores are usually desirable, as they often signify that the code is more straightforward to understand. (2) However, there are some instances where the refactored code has a higher cyclomatic complexity than the original code. This could be due to an increase in code branching or changes in the control flow of the refactored code. In such cases, it is essential to review the refactored code to ensure that it still meets the desired quality and maintainability standards. (3) There are also instances where the cyclomatic complexity of the original and refactored codes is the same. This might mean the refactoring process did not significantly improve code simplicity or maintainability. However, it is still possible that the refactored code offers other benefits, such as enhanced readability or better adherence to coding standards. Overall, the refactored code generated by ChatGPT generally demonstrates lower cyclomatic complexity than the original code. This positive outcome suggests that the refactored code is easier to maintain and understand. However, it is important to carefully review each refactored code snippet to ensure it meets the desired quality standards and doesn't introduce new issues.

B. RQ2: How does ChatGPT affect the Cognitive Complexity after refactoring a dataset of Python code?

The Cognitive Complexity before and after refactoring is depicted in Figures 2 and 7b. Our analysis shows that the median cognitive complexity of the refactored code is 8, which is lower than the original code's value of 11. This indicates that the refactored code is comparatively simpler and easier to comprehend than the original code. The result suggests that the refactoring process has likely enhanced the code's quality, making it more maintainable and reducing the chances of errors or bugs in the future. ChatGPT's refactored code displays a reduced average cognitive complexity compared to the original code, implying that it is more comprehensible, maintainable, and modifiable, leading to improved software quality and decreased development time. The refactored code adheres to best practices such as code decomposition, clear variable naming, and eliminating redundancies, promoting improved readability and maintainability, making it more efficient for developers to work with. However, in some cases, the cognitive complexity of the refactored code may increase. For example, including more complex algorithms or data structures can increase the cognitive complexity of the code. Furthermore, if the code is refactored to include more functions or methods, the cognitive complexity may also increase due to the necessity of comprehending the different operations and interactions between the functions or methods.

C. RQ3: How does ChatGPT affect the Code Smells after refactoring a dataset of Python code?

Figures 3, 5, and 7c provide an overview of the metrics before and after the applied refactoring. The refactored code

has significantly improved code quality, as evidenced by the reduced code smells identified by SonarQube. During the analysis, the original code was found to have 724 code smells, out of which 3 were blockers, 295 were minor, 166 were critical, and 260 were major code smells. In contrast, the refactored code exhibited only 11 blockers, 327 minor, 104 critical, and 173 major code smells. It is noteworthy to mention that a blocker code smell refers to an issue that can obstruct the execution of a function or application, causing significant performance and functionality issues, and requiring immediate attention. On the other hand, minor code smells are less critical and may not immediately impact the system's performance. However, they can accumulate over time and affect code maintainability. Similarly, major code smells can cause significant issues if left unresolved and may affect the system's performance or functionality. Critical code smells are the most severe and require immediate attention as they can lead to major failures in the system. SonarQube executes rules on source code to generate issues. We also explored some of the major and critical code smell issues and observed the changes in the following areas: - The number of instances of the "add missing 'self' parameters" code smell was reduced from 91 to 12.
- The number of instances of the "remove unused function parameter" code smell was reduced from 56 to 20.
- The number of instances of the "rename function" code smell was reduced from 84 to 61.
- The number of instances of the "remove commented-out code" code smell was reduced from 19 to 4.
- The number of instances of the "remove duplicates in this character class" code smell was reduced from 19 to 14.

Example with Description: Although it is noticeable the code smells that ChatGPT can identify and help remove, it is important to note that ChatGPT cannot eliminate all code smells, and some may still require manual intervention. In the original code, there were 7 code smell issues that were reduced to 2 after refactoring using ChatGPT. While ChatGPT can be a useful tool in improving software quality, it should be used in conjunction with other methods to ensure comprehensive code smell detection and removal. Therefore, it is recommended to use ChatGPT as a complementary tool alongside other manual and automated techniques for detecting and removing code smells.

D. RQ4: How does ChatGPT affect the Time Debt after refactoring a dataset of Python code?

ChatGPT's refactoring efforts have yielded noteworthy enhancements in the time debt of the majority of the code files. By refactoring the code, ChatGPT has managed to decrease the time debt by nearly 24 hours, thereby producing code of superior quality. The refactored code can potentially diminish the time required for bug fixes and code maintenance by addressing code smells and simplifying complexity (see Figures 6 and 7d).

In software development, the metric of lines of code (LOC) is commonly used to estimate the size of a software project. However, it is important to note that this metric alone does not always accurately indicate the quality of the code. It is a misconception that more lines of code necessarily mean better software. In fact, sometimes, fewer lines of code can be more efficient, readable, and maintainable than code with more lines. Therefore, relying solely on LOC to assess code quality can be misleading. Instead, it is crucial to consider other metrics such as complexity to evaluate the quality of the code.

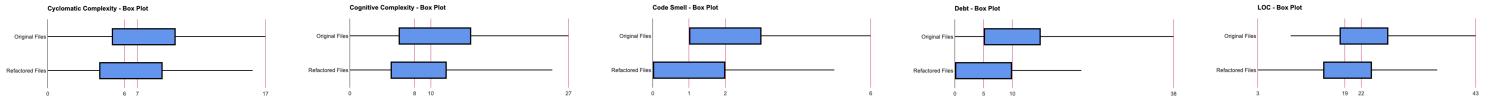
Despite the aforementioned concerns, we also looked into the metric of lines of code and traditional metrics such as cognitive complexity, technical debt, cyclometric complexity, and code smell. Our analysis showed that the median lines of code for the original code were 22, while the refactored code had a median of 19 lines of code. This suggests that code refactoring can result in a reduction in code size, potentially leading to simpler and more maintainable software. So, while it is important not to rely solely on LOC, it is still a useful metric to consider in conjunction with other metrics for a more comprehensive evaluation of the code (see Figures 4 and 7e).

V. TAKEAWAYS

Refactoring code by ChatGPT has successfully improved the code quality in multiple ways. The median values of the refactored code's cyclomatic complexity and cognitive complexity are lower than those of the original code, indicating that the refactored code is less intricate and simpler to comprehend and maintain. This demonstrates that the refactoring has made the code more manageable and easier to work with. Additionally, ChatGPT's refactoring efforts have yielded significant improvements in the time debt of the code files, reducing it by almost 24 hours. This implies that the code is now more efficient, faster, and of superior quality, making it more productive and valuable to the project. However, it is essential to note that while the refactoring has successfully reduced major/critical issues, minor issues have increased in some cases. Therefore, monitoring minor problems and refining the automated tools is essential to ensure that they effectively address these issues. Although ChatGPT can identify and eliminate certain code smells, it cannot eradicate them. Hence, it is imperative to have developers examine the code to ensure its quality and rectify any issues that may arise. Below, we list the implications for developers, researchers, tool builders, and educators.

A. Implications for Developers

The findings of this study can be advantageous for developers as it offers an automated approach to enhancing the readability and maintainability of their code. By utilizing ChatGPT and SonarQube, developers can refactor their code and obtain measurable parameters that demonstrate the efficiency of the refactoring procedure. This enables developers to recognize the sections in their code that require enhancement, implement suitable measures to boost their code quality and adhere better to coding standards.



(a) Cyclomatic Complexity (b) Cognitive Complexity (c) Code Smell (d) Time Debt (e) Lines of Code

Figure 7: Boxplots before and after the application of refactoring.

B. Implications for Researchers

Future research can compare the effectiveness of ChatGPT with other state-of-the-art AI tools, such as Bard and GitHub Copilot. These tools also utilize natural language processing and machine learning techniques to aid in code generation and refactoring. The comparison can evaluate the performance of these tools across various metrics, such as code quality, development time, and usability. This study focused on a relatively medium codebase with limited issues. Future research can investigate how ChatGPT performs on larger, more complex codebases with a broader range of issues. Additionally, it would be valuable to analyze how ChatGPT handles edge cases, such as code with unconventional patterns or syntax. Further, this study focused on Python code, but it would be interesting to investigate how ChatGPT performs in other programming languages. Different programming languages have different syntax and semantics, which can affect the effectiveness of ChatGPT in generating high-quality code. Future research can evaluate the performance of ChatGPT in languages such as Java, C++, and JavaScript.

C. Implications for Tool Builders

The findings of this study can be utilized by researchers and tool builders to incorporate ChatGPT into code editors or IDEs, utilizing the generated outputs to enhance current tools and create new ones that can automatically refactor code. By adopting the same approach of combining ChatGPT with `SonarQube`, researchers and tool builders can create more advanced AI-powered refactoring tools.

D. Implications for Educators

The findings of this study show that by incorporating automated code refactoring into programming courses, students can learn to write better code by seeing the changes that can be made to improve the quality of their code. This helps students understand how to structure their code, how to optimize their code, and how to eliminate redundant code. Additionally, automated code refactoring tools can help students avoid common coding mistakes and learn how to avoid them. Additionally, courses should teach students how to use automated refactoring tools effectively, so they can continue improving their code after the course is completed.

VI. Threats to Validity

Threats to conclusion validity. It refers to the possibility that the study's findings are due to chance. Concerning sampling bias, the dataset of Python submissions may not represent all Python code. For example, it may only contain code from Stepik and Hyperskill platforms, which may have different coding standards than others. Further, the metrics used in the analysis (i.e., cyclomatic complexity, cognitive complexity, code smells, and time debt) may not be sufficient to fully capture the impact of code refactoring on code quality. Other metrics and evaluation techniques may be required to obtain a more comprehensive understanding of the effects of code refactoring.

Threats to internal validity. It refers to the possibility that the study's findings are due to factors other than the independent variable. One limitation is that the model is only as good as the sample of code provided to it. If the code sample does not represent the entire codebase, the model's suggestions may not be helpful. Additionally, ChatGPT cannot understand the broader context of the code, such as its purpose or intended behavior. As a result, the model's suggestions may not always be appropriate for the specific task at hand. Concerning tool dependency, using ChatGPT and `SonarQube` as tools for code refactoring and analysis may have introduced biases and limitations in the study. The effectiveness and accuracy of these tools may affect the validity of the conclusions drawn from the analysis

Threats to construct validity. It refers to the extent to which the study accurately measures the constructs they are supposed to measure. The first threat relates to inadequate measurement. The metrics used in the analysis may not fully capture the construct of code quality. Other constructs, such as maintainability, readability, and efficiency, may also need to be considered to obtain a more comprehensive understanding of the effects of code refactoring. Another threat relates to inappropriate operationalization. That is, how the constructs are measured may not be appropriate or valid, leading to inaccurate conclusions.

Threats to external validity. It refers to the extent to which the study's findings can be generalized to other populations and settings. For population validity, the experiment's results may not be generalizable to other code populations, such as code written by professional programmers. As for setting validity, the experiment's results may not be generalizable to other settings, such as different programming languages or different coding platforms. Finally, regarding treatment fidelity, the ChatGPT refactoring process may not have been implemented consistently across all code files, which could have led to variations in the results.

VII. Related Work

Recent studies utilized ChatGPT for different software engineering tasks. AlOmar et al. [18] explored the interaction dynamic between developer and ChatGPT. The authors

found that developers used generic and specific refactoring-related keywords in their refactoring requests. DePalma et al. [19] investigated ChatGPT refactoring capabilities to refactor the code. White et al. [20] introduces methods for crafting prompts in software engineering by utilizing patterns to address typical issues encountered with LLMs. The research offers a catalog that organizes these patterns based on the specific problems they aim to resolve. Furthermore, the study explored various prompt patterns designed to enhance code quality, support refactoring, aid in requirements gathering, and improve software design.. In another study, Biswas [21] provides a summary of ChatGPT, which boasts numerous capabilities in computer programming. These skills encompass code auto-completion, debugging, prediction, error correction, optimization, documentation creation, chatbot development, transforming text into code, and addressing technical queries. The author highlighted the ability of ChatGPT to provide explanations and guidance to users and concluded that it is a powerful tool for the programming community. Haque and Li [22] explored the capabilities of ChatGPT as a debugging tool and best practices to integrate into the software development workflow. Their findings show that ChatGPT is a helpful tool for debugging, but it should be used with caution in software development. Ma et al. [17] performed a study evaluating the capabilities and limitations of ChatGPT in software engineering in three aspects: 1) syntax understanding, 2) static behavior understanding and 3) dynamic behavior understanding. The authors concluded that ChatGPT possesses capabilities similar to an Abstract Syntax Tree (AST) parser, ChatGPT is susceptible to hallucination when interpreting code semantic structures and fabricating non-existent facts, which underscores the need to explore methods to verify the correctness of ChatGPT outputs to ensure its dependability in software engineering tasks.

VIII. Conclusion

ChatGPT's refactoring efforts have significantly improved the codebase's quality. The refactored code has exhibited lower cyclomatic and cognitive complexity values, indicating that it is simpler and more manageable, leading to improved control flow, readability, and maintainability. The reduction in code smells, particularly the elimination of critical and blocker code smells, highlights ChatGPT's success in addressing significant issues that can cause system failures and performance issues. Furthermore, the refactored code's potential to decrease time debt by nearly 24 hours emphasizes its efficiency in reducing the time required for bug fixes and code maintenance. These results suggest that ChatGPT's capabilities as a language model can significantly benefit software development efforts by improving code quality and reducing development time.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and d. Roberts, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=311424

[2] E. A. AlOmar, M. W. Mkaouer, C. Newman, and A. Ouni, "On preserving the behavior in software refactoring: A systematic mapping study," *Information and Software Technology*, p. 106675, 2021.

[3] S. Fernandes, A. Aguiar, and A. Restivo, "Liveref: a tool for live refactoring java code," in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–4.

[4] M. Aniche, E. Maziero, R. Durelli, and V. H. Durelli, "The effectiveness of supervised machine learning algorithms in predicting software refactoring," *IEEE Transactions on Software Engineering*, vol. 48, no. 4, pp. 1432–1450, 2020.

[5] A. S. Nyamawe, "Mining commit messages to enhance software refactorings recommendation: A machine learning approach," *Machine Learning with Applications*, vol. 9, p. 100316, 2022.

[6] Y. Tang, R. Khatchadourian, M. Bagherzadeh, R. Singh, A. Stewart, and A. Raja, "An empirical study of refactorings and technical debt in machine learning systems," in *2021 IEEE/ACM 43rd international conference on software engineering (ICSE)*. IEEE, 2021, pp. 238–250.

[7] (2016) Cyclomatic complexity. [Online]. Available: https://ieeexplore.ieee.org/document/7725232

[8] (2003) Measurement of the cognitive functional complexity of software. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1225955

[9] M. De Stefano, M. S. Gambardella, F. Pecorelli, F. Palomba, and A. De Lucia, "casper: A plug-in for automated code smell detection and refactoring," in *Proceedings of the International Conference on Advanced Visual Interfaces*, 2020, pp. 1–3.

[10] F. Arcelli Fontana, M. Zanoni, and F. Zanoni, "A duplicated code refactoring advisor," in *Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings 16*. Springer, 2015, pp. 3–14.

[11] P. Meananeatra, S. Rongviriyapanish, and T. Apiwattanapong, "Using software metrics to select refactoring for long method bad smell," in *The 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand-Conference 2011*. IEEE, 2011, pp. 492–495.

[12] (2017) How do developers select and prioritize code smells? a preliminary study. [Online]. Available: https://ieeexplore.ieee.org/document/8094447

[13] (2019) Technical debt. [Online]. Available: https://martinfowler.com/bliki/TechnicalDebt.html

[14] A. Trautsch, S. Herbold, and J. Grabowski, "Are automated static analysis tools worth it? an investigation into relative warning density and external software quality on the example of apache open source projects," *Empirical Software Engineering*, vol. 28, no. 3, p. 66, 2023.

[15] J. Al Dallal and A. Abdin, "Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 44–69, 2017.

[16] A. Birillo, I. Vlasov, A. Burylov, V. Selishchev, A. Goncharov, E. Tikhomirova, N. Vyahhi, and T. Bryksin, "Hyperstyle: A tool for assessing the code quality of solutions to programming assignments," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, 2022, pp. 307–313.

[17] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, and Y. Liu, "The scope of chatgpt in software engineering: A thorough investigation," *arXiv preprint arXiv:2305.12138*, 2023.

[18] E. A. AlOmar, A. Venkatakrishnan, M. W. Mkaouer, C. Newman, and A. Ouni, "How to refactor this code? an exploratory study on developer-chatgpt refactoring conversations," in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 202–206.

[19] K. DePalma, I. Miminoshvili, C. Henselder, K. Moss, and E. A. AlOmar, "Exploring chatgpt's code refactoring capabilities: An empirical study," *Expert Systems with Applications*, vol. 249, p. 123602, 2024.

[20] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," *arXiv preprint arXiv:2303.07839*, 2023.

[21] S. Biswas, "Role of chatgpt in computer programming.: Chatgpt in computer programming." *Mesopotamian Journal of Computer Science*, vol. 2023, pp. 8–16, 2023.

[22] M. A. Haque and S. Li, "The potential use of chatgpt for debugging and bug fixing," *EAI Endorsed Transactions on AI and Robotics*, vol. 2, no. 1, pp. e4–e4, 2023.