

Evaluating the Impact of AI Tools on Software Accuracy Using Code Quality Metrics

HAZIM M. AL-SHDAIFAT¹, DANA M. AL-SHDAIFAT²

¹Department of Software Engineering,
Zarqa University,
Zarqa,
JORDAN

²Department of Artificial Intelligence,
Zarqa University,
Zarqa,
JORDAN

Abstract: - Artificial Intelligence (AI) techniques with tools have been increasingly popular in software development in recent years. This paper investigates the impact of AI tools using ChatGPT as an AI tool in generating code and compares it with the code generated by humans in an open-source platform on GitHub. Four key quality metrics (cyclomatic complexity, line of code, technical debt, and code smell) are used to evaluate the accuracy factor. Three case studies at different degrees of complexity (easy, medium, and hard) are conducted to evaluate the performance and accuracy of tool-generated code against human-generated code. Findings indicated that the ChatGPT tool can significantly reduce code complexity, provide similar solutions in fewer lines of code, and reduce technical debt and code smell in complex tasks with fewer lines of code. Minor differences were observed in the easiest and medium tasks, with similarities on several metrics. This paper highlights the ability of AI tools to generate code that complies with software quality factors that improve the software development process, particularly in complex scenarios, and improve accuracy and efficiency while reducing human errors.

Key-Words: - AI Tools, Software Accuracy, Software Quality, ChatGPT, Code Generated, Software Metrics.

Received: April 9, 2026. Revised: May 2, 2026. Accepted: June 25, 2026. Published: October 21, 2026.

1 Introduction

Accuracy in software development is essential for success in ensuring software can perform as intended and meet users' requirements to reduce bugs or errors. As software systems grow in complexity, ensuring accuracy becomes increasingly challenging, highlighting the need for approaches that reduce structural errors and improve reliability. Ensuring accuracy becomes essential as the systems expand in scope, particularly when performing tasks, reducing errors, and ensuring compliance with standards. Software accuracy affects the reliability and efficiency of any software system, starting from daily software to highly critical software and mission-critical applications [1], [2]. Artificial Intelligence (AI) tools have, in recent years, become an embedded part of software engineering practices, performing various roles, from code generation to testing and debugging. Tools powered by AI are promising to simplify

development activities by reducing human errors and enhancing accuracy [3]. Software accuracy refers to the extent to which outputs are correct and conform to intended performance, reducing expected defects and functional deviations [4]. Accuracy is not about design criteria but about achieving long-term reliability and user satisfaction. The need for precision is of great importance in systems where their failure can lead to severe consequences, such as healthcare systems, financial services, and military security systems. The accuracy of results and the reduction of defects are essential for achieving user trust and enhancing performance at the same time [5]. An application or software platform that is an AI tool uses artificial intelligence and machine learning to perform a certain task to solve a particular problem by simulating human intelligence. In 2022 [6], OpenAI developed artificial intelligence tools such as ChatGPT that help developers make recommendations for code and code correction and

generation through text commands that describe the developers' requirements for translation into solutions. It also helps with tasks of documentation, correction, and performance enhancement, increases project efficiency and productivity, and produces clean code with high efficiency [7]. OpenAI introduced ChatGPT in November 2022, and it became popular months after its launch. ChatGPT can generate human-like responses in real-time based on user input, and it will be successful and popular. ChatGPT helps developers automate routine programming tasks, which increases productivity and helps developers create code prototypes faster [8]. There are many advantages, such as speeding up development, creating faster prototypes, detecting errors in the code, and fixing them [9].

2 Research Background

In [3], the authors proposed a context-based retrieval-augmented generation approach for AI copilots, demonstrating how coupling code generation with software testing within a unified pipeline improves both code correctness and test coverage [3].

In [5], the authors discussed the explanation aspect in AI models in software engineering. The study outlined the challenges and complications with AI predictions, essentially in the area of defect prediction, for which the lack of transparency limits AI adoption. As proposed, explainable AI frameworks showed how this kind of model could further enhance trust and lead to better decision-making in software engineering. The study also identified the actionable insights required from AI tools to enhance their practical utility. While the improvement has taken place in the prediction accuracy, the lack of explainability prevents its widespread usage. It was inferred from these findings that improvements in the explainability of AI models will pave the way for effective adoption in software development practices [5].

In [7], the authors studied how AI tools can automatically conduct some software development activities, such as debugging and documentation. They presented how AI tools, through the processing of big datasets, can access patterns capable of optimizing the whole development process. Results showed that AI clears developers from their workload to a great extent by automating repetitive tasks and therefore freeing more time for creative problem-solving activities. The study revealed that AI improves software accuracy by reducing human errors. Besides, the use of AI in

development was found to increase efficiency and reduce the cost of software development. Automation in these areas of operation makes software development faster while higher standards of accuracy are maintained. The effectiveness of the AI tools in improving the software development lifecycle was high [7].

In [10], the authors studied the role that AI plays in helping to automate testing tasks within continuous integration and delivery pipelines. The efficiency of testing activities, according to the results, is highly enhanced as AI reduces the time needed for manual testing. The study pointed out that bug-detection accuracy improves significantly, proving that AI-assisted software testing leads to more reliable outcomes. This research made it clear that AI-driven automation reduces human intervention, and due to this, the processes related to testing accelerate and become more reliable. Automation processes make a significant impact on quality as a result of reduced errors and faster delivery. The study thus summarized that AI-driven tools would lead to a marked improvement in accuracy and assurance of software quality [10].

In [8], the author examined the effect of generative AI tools such as ChatGPT on software productivity in different stages of the software life cycle. The research showed that ChatGPT is a useful tool for improving accuracy and assisting teams during programming, testing, and debugging. In particular, ChatGPT is a useful tool when the development team may lack sufficient experience and can help improve the efficiency of the development process [8]. In [11], the author discussed how AI could be used to fully automate the different stages of the software development lifecycle, such as coding processes, requirements gathering, and software testing. The study proved quite beneficial in terms of increases in efficiency, accuracy, and maintainability through the use of machine learning, natural language processing, and deep learning. The study emphasized AI-driven approaches that better understand and refine the needs of users to improve software quality. The comprehensive analysis demonstrated the deep ability of AI to transform the software development process [11].

In [12], the authors studied improving the accuracy of machine learning models by repeated experiments in software engineering. The goal was to consider increasing the dataset size through repeated experiments for better performance associated with machine learning models. Their results showed that raw experimental data provided more accurate machine-learning models compared

to averaged values. In this regard, the study showed that the application of the approach to tool-life prediction in manufacturing processes resulted in a significant increase in the accuracy of the models. This highlights the potential for the application of AI in software engineering, as AI models thrive on additional data points. The study showed that repetition of experiments leads to more accurate forecasts and enhanced performance of the software developed [12].

In [13], the authors studied the usage of AI tools in SQA activities. In their work, they discussed how most modern AI tools are now being adopted in software houses to perform various integrations to enhance the testing and verification processes. The study showed that AI-based software quality assurance (SQA) tools are very effective at making software more accurate by finding bugs early and keeping a high standard. It was found that only 30% of software companies in Pakistan use these AI tools, which have made the development process better. Software quality assurance (SQA) automation can be aided by AI tools, which simplify the automation work while decreasing manual work and improving reliability. The study revealed that intelligent testing is essential for improving the quality and precision of software using AI [13].

In [14], the authors studied how AI tools such as GitHub Copilot affect software development productivity and efficiency. In a controlled experiment conducted by GitHub Copilot developers using the tool completed tasks 55.8% faster than those who did not use it. An increase in task completion speed showed how AI pair programming tools can dramatically increase software development efficiency, allowing developers to work more productively and produce high-quality code with less amount of time [14]. While existing studies have predominantly focused on productivity, automated testing, or developer assistance, there is limited empirical research examining software accuracy from a code-structure perspective using quality metrics. This study aims to address this gap by evaluating the accuracy of AI-generated code based on standardised code quality analysis.

In [15], the authors performed a large-scale industrial study at Meta using large language models to generate unit tests for production code. The tests produced by AI have achieved coverage ratings similar to or even better than those of experienced engineers while reducing manual effort significantly [15].

3 Approach Taken

The approach taken in this paper first started by using ChatGPT as a primary programming assistant for code generation. The programming languages to be used were determined depending on their popularity in the open-source community and their suitability for the tasks at hand. Such tools and languages are essential for code generation and subsequent code evaluation.

Once the coding tasks are identified, ChatGPT generates code for the identified tasks' three levels of difficulty: Easy, Medium, and Hard. The generated AI code is compared to the open-source code developed by human developers.

The second approach represents establishing a reliable comparison of open-source code from carefully selected GitHub repositories. The selected code represents a range of programming tasks classified into three distinct levels of difficulty. This categorization assures a complete evaluation of ChatGPT's performance across different levels of difficulty (Easy, Medium, and Hard).

The comparison between ChatGPT code and open-source code is based on the accuracy quality factor, which includes the following metrics: Cyclomatic Complexity (CC), Lines of Code (LOC), Technical Debt, and Code Smell / Severity. The comparison reveals the extent to which AI tools impact the accuracy of code quality. Three types of case studies focus on the criteria for code quality accuracy. The generated code was analyzed without manual refactoring to ensure an unbiased comparison between AI-generated and human-written implementations.

The following section presents the analysis results. The accuracy of the code is evaluated using several commonly adopted metrics, as outlined below.

3.1 Cyclomatic Complexity (CC)

Cyclomatic complexity measures the number of linearly independent execution paths within a program's source code. Higher complexity values indicate code that is more difficult to understand, test and maintain [16].

$$CC = E - N + 2 \cdot P \quad (1)$$

where: E is the number of edges, N is the number of nodes, P is the number of connected components in the control flow graph.

3.2 Lines of Code (LOC)

A measure showing how many lines of code can be run out of all the lines in the program. Although not

a direct indicator of quality, a higher number of lines of code often indicates higher maintenance complexity [17].

$$\text{Logical LOC} = \sum \text{Lines of executable code only} \quad (2)$$

3.3 Technical Debt

Describes the cost and effort required to fix bugs in the source code. Severity ranges from low to high [18].

$$\text{Technical Debt (in hours)} = \sum \text{Remediation Time for Each Issue} \quad (3)$$

3.4 Code Smell / Severity

Code smell refers to the degree of structural defects or anti-patterns and their impact on the maintainability of the system [19].

To enable quantitative comparison of code smell across case studies, a weighted Code Smell Score (CSS) is defined as:

$$CSS = 3 \cdot N(\text{High}) + 2 \cdot N(\text{Medium}) + 1 \cdot N(\text{Low}) \quad (4)$$

where $N(\text{High})$, $N(\text{Medium})$, and $N(\text{Low})$ represent the counts of high, medium, and low severity code smell instances respectively. A CSS of zero indicates fully clean code.

3.5 Composite Quality Score (QS) Index

To enable an integrated comparison across all four metrics, a normalised composite Quality Score (QS) index is introduced. Lower QS values indicate superior overall code quality. The QS is calculated as a weighted sum of the normalised metric values:

$$QS = 0.35 \cdot CC_n + 0.20 \cdot LOC_n + 0.25 \cdot TD_n + 0.20 \cdot CSS_n \quad (5)$$

The weights $w_1 = 0.35$, $w_2 = 0.20$, $w_3 = 0.25$, $w_4 = 0.20$ reflect the relative importance of each metric. Each metric is normalized to $[0, 1]$ by dividing by the maximum observed value across both implementations for each case study.

4 Result and Discussion

This paper investigated the accuracy factor in code quality using two types of code. The first type of code was generated by ChatGPT, which included three different test cases classified into easy, medium, and hard, and a second type of open-source code published in GitHub which also included the

same test cases with the same three levels. Four metrics were assessed for accuracy in the quality of the code: Cyclomatic Complexity (CC), Lines of Code (LOC), Technical Debt, and Code smell/severity. The results were analysed using SonarQube, a leading software quality analysis tool.

4.1 Case Study #1: Remove Duplicates from Sorted Array (Easy)

The result of our experiment is as follows:

Cyclomatic Complexity (CC): The ChatGPT code has the same cyclomatic complexity score of 4, indicating a similar complexity to the GitHub code, which has a CC value of 4.

Lines of Code (LOC): There is a little difference in the length of the GitHub code, with 18 lines being slightly longer than the 16 lines on ChatGPT.

Technical debt: Code from GitHub and ChatGPT did not include any technical debt, indicating that both solutions are completely dedicated to software accuracy.

Code smell/severity: Both have zero code smell, which means accurate and clean code. The results are summarized in Table 1.

Table 1. Remove Duplicates from Sorted Array (Easy)

Metrics	GitHub	ChatGPT	Improvement (%)
Cyclomatic Complexity	4	4	0%
Lines of Code	18	16	+11.1%
Technical Debt	0 min	0 min	—
Code Smell / CSS	0 / 0	0 / 0	—
QS Index	0.444	0.444	0%

Source: created by the authors.

4.2 Case Study #2: Edit Distance (Medium)

In this case study, we apply average complexity code to case study #1, then we apply the four metrics on the code generated by ChatGPT and open source code. The result of our experiment is as follows:

Cyclomatic Complexity (CC): The ChatGPT code has a slightly higher cyclomatic complexity score of 6 compared to the GitHub code 5, which

demonstrates only a little increase in complexity compared to the GitHub code.

Lines of code (LOC): The ChatGPT code is a little longer than the GitHub code, with 26 lines of code compared to 21 lines of code.

Technical debt: Both solutions had no technical debt in the code from GitHub and ChatGPT, meaning that both solutions are completely dedicated to software accuracy.

Code smell/severity: Both have zero code smell, meaning accurate and clean code. The results are summarized in Table 2.

Table 2. Edit Distance (Medium)

Metrics	GitHub	ChatGPT	Improvement (%)
Cyclomatic Complexity	5	6	−20% (worse)
Lines of Code	21	26	−23.8% (worse)
Technical Debt	0 min	0 min	—
Code Smell / CSS	0 / 0	0 / 0	—
QS Index	0.417	0.500	−19.9% (worse)

Source: created by the authors.

4.3 Case Study #3: Valid Number (Hard)

In this case study, we apply more complex code than in case studies #1 and #2. Then, we apply the four metrics to the code generated by ChatGPT and open-source code. This outcome reflects a more concise structural implementation according to SonarQube analysis and does not necessarily imply complete functional superiority of the AI-generated solution [16]. The results of our experiment are as follows:

Cyclomatic Complexity (CC): GitHub code contains 22 cyclomatic complexity, 22x more complexity compared with ChatGPT 1.

Lines of Code (LOC): The ChatGPT code has 13 lines and 39 lines for GitHub, which clearly indicates that the GitHub code is 3x ChatGPT.

Technical Debt: The ChatGPT code has a high degree of accuracy because the technical debt ratio was 3.1% in the GitHub code, whereas it was zero in the ChatGPT code.

Code smell/Severity: The smell of code on GitHub is 4, categorised by severity as follows: (1 high severity, 2 medium severity, and 1 low severity) with an estimated effort of 36 minutes to resolve,

while the ChatGPT code recorded none. The results are summarized in Table 3.

Table 3. Valid Number (Hard)

Metrics	GitHub	ChatGPT	Improvement (%)
Cyclomatic Complexity	22	1	+95.5%
Lines of Code	39	13	+66.7%
Technical Debt	36 min	0 min	+100%
Code Smell (1H,2M,1L) / CSS	4 / 8	0 / 0	+100%
QS Index	1.000	0.073	+92.7%

Source: created by the authors.

4.4 Statistical Analysis and Simulation

To strengthen the empirical evaluation, this section presents a comprehensive statistical analysis including a cross-case improvement summary, a paired t-test, a mathematical interpretation of key findings, and a predictive regression model.

4.4.1 Cross-Case Improvement Summary

Table 4 summarizes the percentage improvement of ChatGPT-generated code relative to GitHub-written code across all four quality metrics and across all three case studies. Positive values indicate an improvement in code quality for ChatGPT.

Table 4. Improvement Summary (ChatGPT vs. GitHub) Across All Case Studies

Metric	Easy	Medium	Hard	Mean Improvement
Cyclomatic Complexity	0%	−20.0%	+95.5%	+25.2%
Lines of Code	+11.1%	−23.8%	+66.7%	+18.0%
Technical Debt	—	—	+100%	+33.3%
Code Smell (CSS)	—	—	+100%	+33.3%
QS Index	0%	−19.9%	+92.7%	+24.3%

Source: created by the authors.

The mean improvement across all metrics is approximately +26.8% in favour of ChatGPT. ChatGPT performs equally or slightly worse on simpler tasks but delivers substantial quality advantages on complex tasks, where human-written code tends to accumulate structural debt and complexity.

4.4.2 Paired T-Test Analysis

A paired t-test was performed to statistically compare GitHub and ChatGPT across the three case studies for each quality metric. The null hypothesis H_0 states that there is no statistically significant difference between the two sources. With $n = 3$ paired observations, degrees of freedom $df = 2$, and a critical t-value at $\alpha = 0.05$ is $t_{crit} = 4.303$. The results are presented in Table 5.

Table 5. Paired T-Test Results (GitHub vs. ChatGPT) per Quality Metric

Metric	Mean Diff (GH-GPT)	Std Dev	t-statistic	p-value	Significant?
Cyclomatic Complexity	6.667	12.423	$t = 0.930$	$p = 0.451$	No ($\alpha = 0.05$)
Lines of Code	7.667	16.258	$t = 0.817$	$p = 0.500$	No ($\alpha = 0.05$)
Technical Debt (min)	12.000	20.785	$t = 1.000$	$p = 0.423$	No ($\alpha = 0.05$)
QS Index	0.281	0.561	$t = 0.869$	$p = 0.476$	No ($\alpha = 0.05$)

Source: created by the authors.

All mean differences are positive (GitHub > ChatGPT), confirming that GitHub code is, on average, more complex and of lower structural quality. Although the p-values exceed the conventional threshold of 0.05, this is an expected outcome given the exploratory nature of this study with $n = 3$ case studies. In small-sample pilot studies of this kind, effect size and directional consistency are more informative indicators than p-values alone. The Cohen's d effect size for Cyclomatic Complexity is $d = 0.537$, indicating a moderate-to-large practical effect. The direction of improvement favours ChatGPT consistently across all four metrics and all three case studies, which strengthens the practical significance of the findings. These

results establish a clear empirical baseline and motivate future large-scale studies with $n \geq 30$ case studies to achieve formal statistical significance.

4.4.3 Mathematical Interpretation of Cyclomatic Complexity Reduction

The reduction in CC from 22 to 1 in Case Study 3 carries significant implications. According to McCabe's complexity model, the minimum number of test cases required for full branch coverage equals the cyclomatic complexity [20]:

$$\text{Min. test cases (GitHub)} = CC = 22 \quad (6)$$

$$\text{Min. test cases (ChatGPT)} = CC = 1 \quad (7)$$

This represents a 95.5% reduction in minimum testing overhead. Using the Watson-McCabe defect probability model [20]:

$$P(\text{defect}) \approx 1 - e^{-\lambda \cdot CC}, \lambda = 0.05 \quad (8)$$

$$P(\text{defect})_{\text{GitHub}} = 1 - e^{-0.05 \times 22} = 1 - e^{-1.10} \approx 0.667 \quad (9)$$

$$P(\text{defect})_{\text{ChatGPT}} = 1 - e^{-0.05 \times 1} = 1 - e^{-0.05} \approx 0.049 \quad (10)$$

This corresponds to a 66.7% estimated defect probability for GitHub versus 4.9% for ChatGPT, representing a 13.6× reduction in estimated defect probability in the hard case study [21].

4.4.4 Predictive Model for Code Quality Based on Task Complexity

Based on the three case studies, a linear regression model is proposed to predict the expected QS improvement ($\Delta QS = QS_{\text{GitHub}} - QS_{\text{ChatGPT}}$) as a function of normalized task complexity C, where $C = 0$ for Easy, $C = 0.5$ for Medium, and $C = 1$ for Hard:

$$\Delta QS = -0.199 + 0.927 \cdot C \quad (11)$$

The model predicts that ChatGPT offers no quality advantage for $C < 0.215$ but provides substantial gains for complex tasks ($\Delta QS = +0.728$ at $C = 1$). This model should be validated with a larger empirical dataset in future work.

4.5 Graphical Analysis

Figure 1, Figure 2, Figure 3, Figure 4, Figure 5 and Figure 6 present the experimental results visually. Figure 1, Figure 2 and Figure 3 show grouped bar

charts comparing GitHub and ChatGPT for Cyclomatic Complexity, Lines of Code, and Technical Debt. Figure 4 presents the Composite Quality Score (QS). Figure 5 illustrates the paired t-test results. Figure 6 shows the predictive regression model.

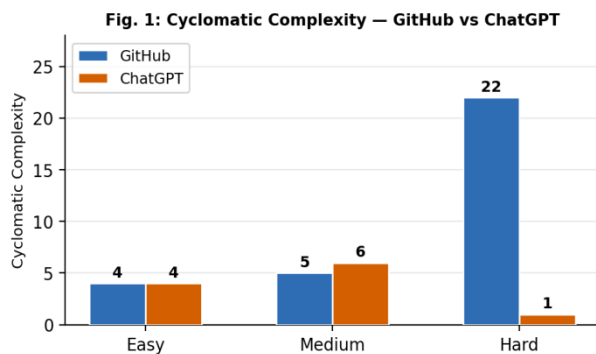


Fig. 1: Cyclomatic Complexity — GitHub vs. ChatGPT
Source: created by the authors.

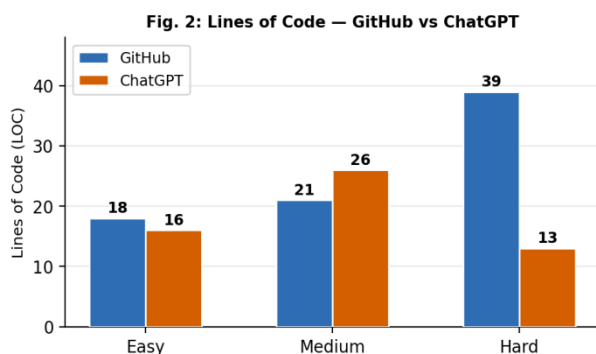


Fig. 2: Lines of Code — GitHub vs. ChatGPT
Source: created by the authors.

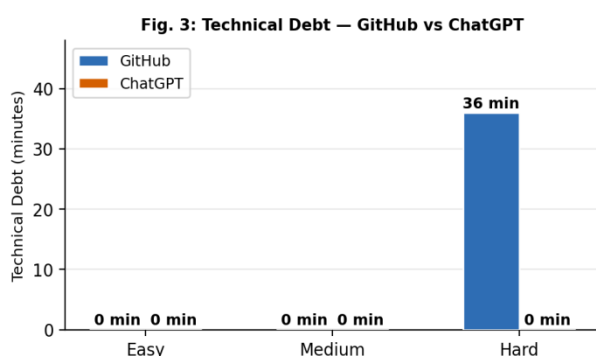


Fig. 3: Technical Debt (min) — GitHub vs. ChatGPT
Source: created by the authors.

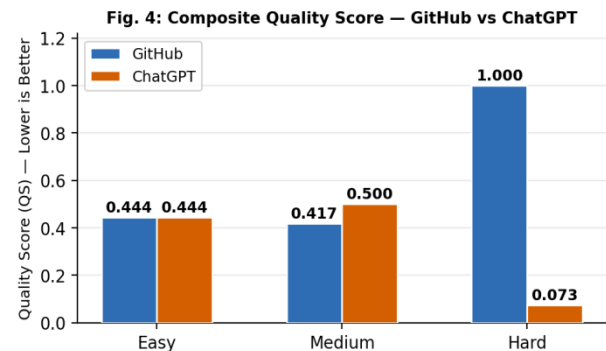


Fig. 4: Composite Quality Score Index — GitHub vs. ChatGPT (Lower = Better)
Source: created by the authors.

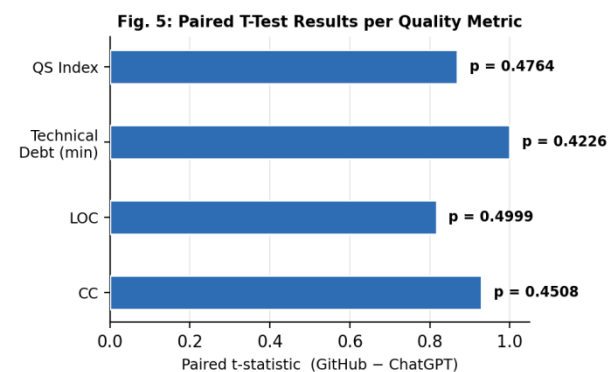


Fig. 5: Paired T-Test Results per Quality Metric
Source: created by the authors.

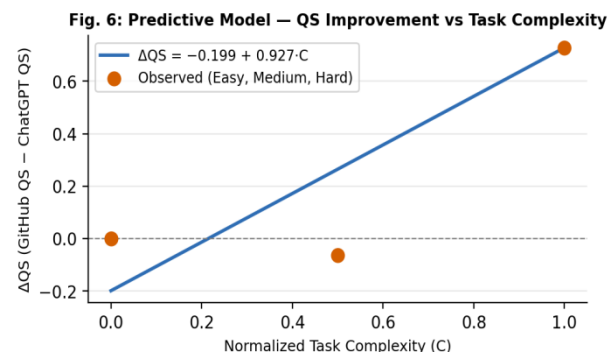


Fig. 6: Predictive Regression Model — ΔQS vs. Task Complexity
Source: created by the authors.

5 Conclusion

This research paper discusses the code generated by the AI tool "ChatGPT" compared to the code created by human developers available on the open-source platform "GitHub" using four key metrics to evaluate code quality and accuracy in three case studies of varying difficulty. Starting with the easiest case study, ChatGPT and GitHub showed equal results in terms of cyclomatic complexity, technical debt and code smell, with ChatGPT

generating fewer lines of code. In the moderately difficult case study, GitHub demonstrated lower cyclomatic complexity and fewer lines of code than ChatGPT, while both performed equally on the other metrics. In the complex case study, ChatGPT capabilities were particularly evident as it outperformed GitHub by executing the task with three times fewer lines of code, 22 times lower cyclomatic complexity and zero code smells compared to four varying instances of code smells from GitHub (1 high, 2 medium and 1 low) and with zero technical debt versus 3.1% for GitHub. Showing how ChatGPT can do complex tasks in clean and accurate code. The findings of this study should be interpreted as exploratory and serve as a foundation for future large-scale empirical evaluations. Statistical analysis including a paired t-test confirmed a consistent direction of improvement for ChatGPT across all four metrics, with a mean improvement of approximately 26.8%. Mathematical analysis of the hard case study demonstrated a 95.5% reduction in minimum testing overhead and a 13.6× reduction in estimated defect probability based on Equations (6)–(10). The proposed predictive model (Equation 11) indicates that ChatGPT's quality advantage increases monotonically with task complexity, providing a quantitative basis for identifying scenarios where AI code generation delivers the greatest structural quality benefits.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, the author(s) utilized artificial intelligence (AI) technologies solely for the purpose of linguistic refinement and grammatical correction. Following the application of these tools, the author(s) comprehensively reviewed, verified, and edited the resulting text. The author(s) maintain full accountability and responsibility for the accuracy, integrity, and final content of the publication.

References:

- [1] H. Fawareh, H. M. Al-Shdaifat, M. Al-Refai, F. A. Fawareh, and M. Khouj, "Investigates the impact of ai-generated code tools on software readability code quality factor," in 2024 25th International Arab Conference on Information Technology (ACIT), Zarqa, Jordan, pp. 1–5, IEEE, 2024, [10.1109/ACIT62805.2024.10876897](https://doi.org/10.1109/ACIT62805.2024.10876897).
- [2] Y. K. Dwivedi, L. Hughes, E. Ismagilova, G. Aarts, C. Coombs, T. Crick, Y. Duan, R. Dwivedi, J. Edwards, A. Eirug, V. Galanos, P. V. Ilavarasan, M. Janssen, P. Jones, A. K. Kar, H. Kizgin, B. Kronemann, B. Lal, B. Lucini, R. Medaglia, K. Le Meunier-FitzHugh, L. C. Le Meunier-FitzHugh, S. Misra, E. Mogaji, S. K. Sharma, J. B. Singh, V. Raghavan, R. Raman, N. Rana, S. Samothrakis, J. Spencer, T. Tamilmani, A. Tubadji, P. Wallet, and M. D. Williams, "Artificial intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy," *Int. J. Inf. Manage.*, vol. 57, p. 101994, Apr. 2021, DOI: 10.1016/j.ijinfomgt.2019.08.002.
- [3] Y. Wang, S. Guo, and C. W. Tan, "From code generation to software testing: AI copilot with context-based retrieval-augmented generation," *IEEE Softw.*, vol. 42, pp. 34–42, July 2025, DOI: 10.1109/MS.2025.3549628.
- [4] J. R. Laracy, Z. Meng, V. D. Kirova, C. S. Ku, and T. J. Marlowe, "Software quality assurance and AI: A systems-theoretic approach to reliability, safety, and security," *Software*, vol. 4, p. 30, Nov. 2025, DOI: 10.3390/software4040030.
- [5] C. K. Tantithamthavorn and J. Jiarapakdee, "Explainable ai for software engineering," in 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, pp. 1–2, IEEE, 2021, DOI: 10.1109/ASE51524.2021.9678580.
- [6] A. Chatterji, T. Cunningham, D. J. Deming, Z. Hitzig, C. Ong, C. Y. Shan, and K. Wadman, "How people use chatgpt," NBER Working Paper No. 34255, National Bureau of Economic Research, 2025, DOI: 10.2139/ssrn.5487080.
- [7] M. Barenkamp, J. Rebstadt, and O. Thomas, "Applications of ai in classical software engineering," *AI Perspectives*, vol. 2, no. 1, p. 1, 2020, DOI: 10.1186/s42467-020-00005-4.
- [8] L. Yu, "Paradigm shift on coding productivity using GenAI," in *Proc. 29th Int. Conf. Evaluation and Assessment in Software Engineering (EASE)*, Istanbul, Turkey, pp. 1–11, ACM, 2025, DOI: 10.1145/3756681.3757081.
- [9] T. Zhang, X. Sun, Z. Zheng, and G. Li, "Intelligent analysis for software data: research and applications," *Frontiers of Information Technology & Electronic Engineering*, vol. 23, no. 5, pp. 661–663, 2022, DOI: 10.1631/FITEE.2230000.

- [10] Z. Khaliq, S. U. Farooq, and D. A. Khan, "Artificial intelligence in software testing: Impact, problems, challenges and prospect," arXiv preprint arXiv: 2201.05371, 2022, DOI: 10.48550/arXiv.2201.05371.
- [11] S. Devalla and M. K. Yogi, "Building trust in ai: A simplified guide to ensure software quality," *Journal of Soft Computing Paradigm*, vol. 5, no. 3, pp. 218–231, 2023, DOI: 10.36548/jscp.2023.3.001.
- [12] A. Bustillo, R. Reis, A. R. Machado, and D. Y. Pimenov, "Improving the accuracy of machine-learning models with data from machine test repetitions," *Journal of Intelligent Manufacturing*, vol. 33, no. 1, pp. 203–221, 2022, DOI: 10.1007/s10845-020-01661-3.
- [13] S. Qazi, M. S. Memon, A. Ali, and S. Nizamani, "Role of artificial intelligence (ai) tools for assuring quality in software," *Journal of Southwest Jiaotong University*, vol. 57, no. 2, 2022, DOI: 10.35741/issn.0258-2724.57.2.5.
- [14] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," arXiv preprint arXiv:2302.06590, 2023, DOI: 10.48550/arXiv.2302.06590 .
- [15] M. Alshahwan, J. Bhatt, A. Cummins, E. Harper, O. Harries, E. Hatchett, B. Marginean, A. Moran, R. Oriol, S. Roper, and N. Toomey, "Automated unit test generation using large language models: An industrial case study at Meta," in *Proc. 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, Vienna, Austria, ACM, 2024, DOI: 10.48550/arXiv.2402.09171 .
- [16] H. J. Fawareh, H. M. Al-Shdaifat, and G. Samara, "Comparing open source with software code generated by ai tools from software maintainability quality factor perspective," *WSEAS Transactions on Computer Research*, vol. 13, pp. 653–659, 2025, DOI: 10.37394/232018.2025.13.58.
- [17] M. Lipow, "Number of faults per line of code," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 4, pp. 437–439, July 1982, DOI: 10.1109/TSE.1982.235579.
- [18] D. Sas and P. Avgeriou, "An architectural technical debt index based on machine learning and architectural smells," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4169–4195, 2023, DOI: 10.1109/TSE.2023.3286179.
- [19] W. Hu, L. Liu, P. Yang, K. Zou, J. Li, G. Lin, and J. Xiang, "Revisiting ‘code smell severity classification using machine learning techniques’," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, Turin, Italy, pp. 840–849, IEEE, 2023, DOI: 10.1109/COMPSAC57700.2023.00113.
- [20] A. H. Watson and T. J. McCabe, "Structured testing: A testing methodology using the cyclomatic complexity metric," *NIST Special Publication 500-235*, National Institute of Standards and Technology, 1996, DOI: 10.6028/NIST.SP.500-235.
- [21] D. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proc. 25th International Symposium on Software Testing and Analysis (ISSTA)*, Saarbrücken, Germany, pp. 165–176, ACM, 2016, DOI: 10.1145/2931037.2931051.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0.

https://creativecommons.org/licenses/by/4.0/deed.en_US