Information Systems Student Research Journal

Research Paper DOI: ISSN:

Evaluation of the Code Quality Generated by Generative AI

Salame Azbarka
The Academic College of Tel-Aviv-Yaffo

Ali Mughrabi
The Academic College of Tel-Aviv-Yaffo

Mario Sabbagh
The Academic College of Tel-Aviv-Yaffo

Idan Roth
The Academic College of Tel-Aviv-Yaffo
0000-0001-6944-2665

Abstract:

This study aims to evaluate and differentiate the effectiveness of the leading Generative-AI tools for code generation by examining key code quality metrics. The research focuses on the evaluation of *correctness*, *readability*, and the *efficiency* of GenAI code. The research methodology comprised two stages: an initial benchmark of four generative AI tools, followed by further evaluation of the two top-performing models from the first stage. The four generative AI tools-ChatGPT, Copilot, Claude, and Gemini- generated a code for ten coding problems, and then for the second benchmark, the best two Generative AI were further compared with an additional sample of 30 problems. The code was generated in Python, and all of the problems are from the competition coding website leetcode.com. The first benchmark results indicate that ChatGPT generated the best quality code with the highest score in most metrics, followed by Copilot, Claude, and finally, Gemini. In the second benchmark, in code correctness, ChatGPT outperformed Copilot, with 17 out of 30 code snippets being completely correct compared to Copilot's 12. In terms of code readability, both ChatGPT and Copilot scored highly, averaging 82.3% and 83.7%, respectively, while Copilot displayed a higher efficiency score. Nevertheless, the differences were statistically insignificant and, thus, may indicate that the tools are largely equivalent overall, though they may use different approaches to memory usage.

Keywords: Artificial Intelligence, Generative AI, ChatGPT, Copilot, Claude, Gemini, Code correctness, Code readability, Code efficiency.

This manuscript underwent peer review. Ricarda Schlimbach served as the managing editor.

1 Introduction

Generative Artificial Intelligence (GenAI) has emerged as a transformative advancement in the field of artificial intelligence, attracting attention from researchers, developers, and the general public alike. Since late 2022, tools such as ChatGPT, DALL-E, and Midjourney have demonstrated GenAI's potential by enabling end-users to generate human-like text, images, and other media content effortlessly (Peñalvo & Ingelmo, 2023). Among these developments, one of the most impactful domains has been code generation—where Large Language Models (LLMs) now routinely assist with programming tasks, debugging, and algorithm design.

The present study investigates the programming capabilities of prominent GenAI tools, specifically focusing on Python code generation. Previous studies mainly assessed Copilot and ChatGPT for code correctness (e.g., Idrisov & Schlippe, 2024; Yetistiren et al., 2022, 2023; Nguyen & Nadi, 2022), but broader evaluations, including readability and efficiency, are needed, especially for chat-based tools like Claude and Gemini. Over time, various quality models have been proposed in the literature, such as Boehm's model (Boehm, 1978) and McCall's model (McCall et al., 1977). To systematically assess the quality of AI-generated code, we evaluate three core dimensions: correctness, readability, and efficiency. These criteria were selected based on their alignment with the widely accepted ISO/IEC 9126 software quality model, which emphasizes key quality characteristics such as functionality (linked to correctness), usability (reflected in readability), and efficiency (as defined by computational performance) (International Organization for Standardization, 2001; Tonella & Abebe, 2008). Our selection is also grounded in existing literature where these three attributes are repeatedly cited as critical in assessing code quality (Börstler et al., 2023; Cormen et al., 2009; Pearce et al., 2025).

Although other dimensions, such as security, sustainability, and maintainability, are also important, they either fall outside the scope of short algorithmic problems (as in LeetCode) or require domain-specific resource profiling that was beyond the current study's design.

ChatGPT, Copilot, Claude, and Gemini are prominent GenAI tools frequently compared in prior studies (e.g., Cho et al., 2025; Ermis et al., 2025; Mavrych et al., 2025). This work will focus on these GenAI tools and propose the following research question:

> **RQ:** How do leading GenAI tools—ChatGPT, Copilot, Claude, and Gemini—compare in terms of the quality of Python code they generate, measured by correctness, readability, and efficiency?

To address this question, we evaluate each GenAI tool on a set of programming problems and benchmark their outputs using standardized and replicable metrics. The following hypotheses guide our evaluation:

H1 (Correctness):

H_0: There is no difference in code correctness between the GenAI tools (Hansson & Ellréus, 2023).

H_1: A statistically significant difference exists in code correctness.

H2 (Readability):

H_0: There is no difference in code readability between the GenAI tools (Sobania et al., 2022).

H_1: A statistically significant difference exists in code readability.

H3 (Efficiency):

H_0: There is no difference in code efficiency between the GenAI tools.

H_1: A statistically significant difference exists in code efficiency.

These hypotheses were tested using a two-phase benchmarking design across 40 LeetCode problems, enabling comparative analysis of GenAI performance across increasing complexity, with the tools ChatGPT, Copilot, Gemini, and Claude, all at the latest version as of June 10, 2024.

In the next section, we will address related work. Research setup and research methods will be described in Sections 3 and 4, respectively, and in Section 5, we will present our findings from each method. In Section 6, we will conclude and discuss our work findings.

2 Related Work

In this section, we will cover previous work regarding automatic program code generation and evaluation. Nguyen & Nadi (2022) conducted an empirical evaluation of Copilot's code suggestions using 33 LeetCode questions across four programming languages: Python, Java, JavaScript, and C. To assess the correctness of Copilot's suggestions, they utilized the test cases provided by LeetCode and submitted Copilot's generated code solutions to LeetCode's online judging system. This allowed them to determine whether the solutions passed all test cases (fully correct), passed some test cases (partially correct), or failed due to factors like compilation errors or runtime errors. To evaluate how understandable Copilot's code is, Nguyen & Nadi (2022) used two complexity metrics calculated by the SonarQube static analysis tool: cyclomatic complexity and cognitive complexity. The cyclomatic complexity measures how many different paths there are through the code. Higher cyclomatic complexity means more branching (e.g., if statements, loops) in the code, making it more complex. The cognitive complexity aims to capture how difficult the code is for a human to understand intuitively. It doesn't just count branches but also considers things like nested loops and the use of shorthand expressions. Nguyen & Nadi (2022) found that Copilot's suggestions varied in correctness based on the programming language, with Java having the highest rate of accepted solutions (57%) and JavaScript the lowest (27%). Additionally, they reported that Copilot generally generated code with low cyclomatic and cognitive complexity scores, indicating good understandability. However, they also identified some potential shortcomings, such as producing code that could be streamlined further while depending on helper methods that have not been defined.

Yetistiren et al. (2022) also evaluated Copilot, but using the HumanEval dataset of 164 programming problems. The study evaluated the Copilot code for code validity (91.5% error-free), correctness (28.7% completely correct, 51.2% partially correct), and efficiency (compared to human-written code). The findings highlight that good function descriptions/names improve the performance of Copilot and indicate that with programmer guidance, Copilot can significantly impact software development.

Yetistiren et al. (2023) in a different study examined the code validity, code integrity, code security, code reliability, and code maintainability of various AI-powered code generation tools such as Copilot, Amazon CodeWhisperer, and ChatGPT using the same HumanEval dataset (Yetistiren et al., 2022). All three tools did generate valid code that compiles 90% of the time, with ChatGPT being the most successful. For code correctness measured by the number of test cases the code passed successfully, ChatGPT outperformed others, generating 65.2% completely correct solutions. Copilot and CodeWhisperer lagged behind. However, the tools also generated many partially correct solutions. No significant security issues were found, although some code maintenance flaws were identified. Although impressive, the tools still require human input and testing to improve the quality of the generated code (Yetistiren et al., 2022).

In contrast, a previous study by Yetistiren et al. (2022) focused solely on Copilot, which had lower correctness rates. Both studies noted the importance of detailed function descriptions to maintain the quality of the code and emphasized the need for tools for supervision and human testing (Yetistiren et al., 2022; Yetistiren et al., 2023).

Another recent study that is highly relevant to the present research is the work by Hansson & Ellréus (2023). This comprehensive study conducted a controlled quantitative experiment to assess the code generation capabilities of ChatGPT and Copilot in terms of code correctness and quality. The authors identified a research gap, as previous studies had primarily focused on the validity and correctness of code generated by Copilot. At the same time, the quality aspect and the inclusion of ChatGPT had been largely unexplored. They generated algorithms from ChatGPT and Copilot, then evaluated the code's correctness through unit testing and code quality based on principles from the book "Clean Code" by Hansson & Ellréus (2023).

The study's findings revealed that both ChatGPT and Copilot demonstrated the ability to generate functionally correct code based on provided instructions, with ChatGPT achieving an 87.33% correctness rate and Copilot 89%. However, no statistically significant difference was observed between the two GenAI platforms in terms of code correctness. Regarding code quality, ChatGPT generated 80.7% of algorithms without any quality rule violations, while Copilot generated violation-free algorithms 64.7% of the time. The authors found a statistically significant difference in code quality between ChatGPT and Copilot, indicating that ChatGPT generally produces higher quality code (Hansson & Ellréus, 2023).

Idrisov & Schlippe (2024) evaluated and compared the correctness, efficiency, and maintainability of AI-generated program codes against the human-generated program codes. The research focuses on analyzing the interaction of various metrics, such as cyclomatic complexity and maintainability index, to assess the quality of the generated program code. Idrisov & Schlippe (2024) include the analysis of program codes addressing problems of varying difficulty levels and provide a dataset of AI-generated and human-generated program codes for the research community. Idrisov & Schlippe (2024) evaluate computational resources using metrics like time and space complexity, runtime, and memory usage, while also assessing maintainability through metrics such as lines of code, cyclomatic complexity, Halstead complexity, and maintainability index. The study involves GenAIs producing program code in Java, Python, and C++ to solve problems from the coding website leetcode.com. Various GenAIs are compared based on their performance in solving coding problems, with Copilot demonstrating the highest success rate, followed by BingAI Chat, ChatGPT 3.5, Code Llama 2, and InstructCodeT5+. Although ChatGPT generated only four correct program codes, it was the only GenAI capable of providing a correct solution to a coding problem of hard level difficulty (Idrisov & Schlippe, 2024). A summary of these prior studies can be found below (Table 1).

Table 1. Summary of Prior Studies on GenAI Code Generation

Reference	Tools Compared	Programming Languages	Dataset / Benchmark	Evaluation Metrics	Key Findings
Nguyen & Nadi (2022)	Copilot	Python, Java, JavaScript, C	33 LeetCode problems	Correct-ness (via test cases) Cyclomatic Complexity Cognitive Complexity	Java highest correctness (57%) Low complexity scores Helper methods sometimes un-defined
Yetistiren et al. (2022)	Copilot	Various	HumanEval (164 prob-lems)	Code Validity Correct-ness Efficiency	91.5% valid 28.7% completely correct Prompt quality influenced results
Yetistiren et al. (2023)	Copilot, ChatGPT, CodeWhis-perer	-	HumanEval	Correct-ness Validity Maintainability Security	ChatGPT best in correctness (65.2%) All tools produced valid but not always maintainable code
Hansson & Ellréus (2023)	ChatGPT, Copilot	-	Custom "Clean Code" benchmark	Correct-ness (unit tests) Code Quality (Clean Code rules)	No significant correctness difference ChatGPT better quality (fewer violations)
Sobania et al. (2022)	Copilot, Bing Chat, ChatGPT 3.5, Code Llama 2, In-structCodeT5+	Java, Python, C++	LeetCode (varied difficulty)	Correct-ness Efficiency Maintainability Memory usage Complexity metrics	Copilot highest success rate ChatGPT solved hardest problem Efficiency varied

GenAIs has revolutionized content creation, including code generation, with tools like ChatGPT, Copilot, Claude, and Gemini demonstrating remarkable capabilities (Feuerriegel et al., 2024; Strobel et al., 2024). While these tools exemplify the "Assistant" type by refining user inputs (Strobel et al., 2024), their effectiveness in producing high-quality code—measured by correctness, readability, and efficiency—remains underexplored. This study addresses this gap by evaluating the four tools, using Python with varying difficulty, for a comprehensive evaluation using a balanced sample. Python was chosen for its ubiquity and the authors' familiarity with it. The findings aim to enhance understanding of GenAIs practical utility in software development while contributing to broader discussions on responsible deployment (Feuerriegel et al., 2024).

Thus, this work focuses on code correctness, readability, and efficiency of the code generated by these tools and explores each tool's unique strengths and weaknesses for the benefit of developers.

3 Research Design

In this section, we present an in-depth overview of our research setup, including an introduction to the advanced GenAI models employed in this study, the selection of coding problems from LeetCode, and the evaluation metrics used to assess code quality.

3.1 GenAI Models

This paper evaluates the four GenAI models described below.

3.1.1 OpenAI's ChatGPT 4.0

ChatGPT is an AI chatbot driven by the Generative Pre-Trained Transformer (GPT) language model, released to the public by OpenAI in November 2022. It has gained much attention for its ability to generate detailed humanoid text based on user instructions. The chatbot can be used for writing essays, solving mathematical equations, and writing, as well as generating and debugging code (Hörnemalm, 2023). The GPT-4 model is based on the basic concepts established by its predecessors and combines advances in machine learning, natural language processing, and deep learning (Moritz, 2024).

3.1.2 Microsoft Copilot

Launched in 2023, Copilot is powered by artificial intelligence integrated into various Microsoft products and services. It combines large language models with Microsoft's suite of productivity tools. Copilot can generate text, analyze data, create presentations, and solve coding tasks in development environments (Microsoft, 2025). The authors of this study used the free version of the web chat, which was less powerful than the pro version.

3.1.3 Google Gemini

Gemini, a large language model (LLM) developed by Google DeepMind, was released in December 2023. Gemini can generate a wide variety of data types, including text, code, images, and more, based on the input it receives. Its most prominent feature is the ability to understand and work with different types of data, such as text, images, audio, PDF files, and videos. This versatility enables Gemini to produce comprehensive, context-aware responses, making it valuable for a wide range of tasks and applications (Imran & Almusharraf, 2024).

3.1.4 Anthropic's Claude

Claude is a family of large language models developed by Anthropic, released in March 2024. The model can handle complex tasks such as data analysis, creative writing, and coding across multiple programming languages. Claude 3.0 builds on the constitutional AI principles established in previous versions, focusing on safety and ethical behavior while providing more accurate and contextually relevant responses (Anthropic, 2024).

3.2 LeetCode Platform

LeetCode is a widely used online platform for preparing candidates for software engineering technical interviews, offering a large library of coding problems organized by varying levels of difficulty (Cui et al., 2024).

3.3 Selected Coding Problems for Our Evaluation

A total of 40 programming problems were randomly selected from LeetCode, spanning three difficulty levels: 13 easy (testing basic coding fluency), 14 medium (testing algorithmic thinking), and 13 hard (testing mastery of advanced techniques). All selected problems belong to the Algorithms category in LeetCode, for space and time complexity measurement. The limited number of problems was chosen due to the significant effort required to post the prompts via the web interface, copy the answers, and manually evaluate the metrics. The prompts used to generate the code were taken from the LeetCode website, and multiple generations were not allowed.

3.4 Methodology

Due to resource constraints, the study design was carried out using benchmarks in two phases: The first benchmark was performed by introducing 10 programming problems (3 easy, 4 medium, and 3 hard) to each of the four GenAI tools. Based on the results of this benchmark, the two best-performing tools were selected for the second benchmark. In the second benchmark phase, the remaining set of 30 programming problems (with an equal distribution of easy, medium, and hard problems) was presented to each of the two selected GenAI tools. The code generated from each tool was analyzed for correctness, readability, and efficiency.

3.4.1 Evaluation Metrics

Code Correctness

The correctness of the code was measured according to a scale of 5 levels based on the scale used by (Yetistiren et al., 2022; Yetistiren et al., 2023), with small adjustments, as follows:

- Correct: The code passes all test cases.
- Partially correct: The code has no errors, but the output differs from the expected output in at most 50% of the test cases.
- False: The code has no errors, but the output differs from the expected output in more than 50% of the test cases.
- Compilation error: The submitted code cannot be compiled.
- Runtime error: The code fails in at least one test case due to a runtime error (i.e., division by zero, etc.).

Code Readability

Code readability was measured using a checklist tool for evaluation of the generated code readability, relies on previous work (Hansson & Ellréus, 2023; Martin, 2013; Dantas & Maia, 2021), and includes:

- A line should not be longer than 120 characters.
- The length of the functions should not be more than 20 lines.
- Functions should have no more than three arguments.
- There should be no nested loops more than one level deep.
- There should be no more than one sentence per line.
- Consistent naming conventions for variables, functions, and classes.
- Clear and descriptive comments explaining the purpose and logic of the code.
- Correct indentation and formatting for better visual organization.
- Using meaningful and self-documenting variable and function names.
- Adherence to coding style guides and Python-specific conventions.

All codes generated using GenAI were evaluated according to these rules. The evaluation process included an examination of the degree of compliance with each of these criteria. The final score was calculated as the ratio between the number of criteria the code successfully met and the total number of all criteria, multiplied by 100. These results represented the readability of the code on a scale of 0 to 100 points.

Code Efficiency

Code efficiency was manually measured by analyzing the time and space complexity of the algorithm theoretically, without running the code. This is done by counting the number of basic operations performed by the algorithm as a mathematical function of the size of the input. These results were expressed using a large O notation, which represents the upper limit of the growth rate of the algorithm (Han, 2002). For time and space complexities, the notations included $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, and more; the smaller the better.

Thus, the code quality assessment was composed of the three quality metrics mentioned above: code correctness, code readability, and code efficiency (Table 2).

Table 2. The Benchmark Criteria

The main criteria	Main criteria weight	Secondary criteria	Secondary criteria weight
Quality	100%	Correctness Readability Efficiency	33.33% 33.33% 33.33%

Benchmark execution included the following activities:

1. LeetCode problems dataset preparation: Selection of 10 programming problems from LeetCode main dataset. Each problem included a function signature and function instructions, which provided the specifications and requirements for the desired code solution.
2. Code generation: The described problems were given to the four GenAI tools for code generation.
3. Quality assessment: The generated code went through a series of tests and quality assessments, focusing on the three quality metrics: correctness, readability, and efficiency.
4. Initial GenAI selection: The two GenAI tools with the highest aggregated code quality score were given the remaining 30 problems for further inquiry, targeted to discover the outperforming GenAI tool.

3.5 Hypothesis Testing Design

3.5.1 Hypothesis H1

Code correctness, measured on an ordinal scale, was evaluated with the non-parametric test, the Mann-Whitney U test, to assess whether a significant difference exists between the distributions of two independent groups, where the assumption of normality may not apply.

3.5.2 Hypothesis H2

Code readability will be tested by the Independent Samples T-test, which is a parametric test that compares the means of two independent groups and is appropriate when the data are continuous and normally distributed. The readability scores are continuous, and the sample size is large enough for the central limit theorem to apply, making the t-test appropriate for comparing the means of the two groups.

3.5.3 Hypothesis H3

The Mann-Whitney U test also tested code readability. Time and space complexity, as represented by Big O notation, can be thought of as an ordinal variable as $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$, etc.

Thus, the Mann-Whitney U test is suitable for evaluating whether a significant difference exists in the efficiency distribution between the two groups.

4 Results

4.1 First Benchmark Phase

To systematically assess the quality of code generated by various GenAI tools, the evaluation was conducted in two distinct benchmarking phases. In the first phase, all four selected GenAI tools—ChatGPT, Copilot, Claude, and Gemini—were presented with a sample of 10 diverse programming problems. This initial benchmark served as a screening step to identify the top-performing tools based on correctness, readability, and efficiency metrics. In the second phase, only the two leading tools from phase one (ChatGPT and Copilot) were evaluated further using an extended set of 30 additional problems. This two-tiered approach enabled a more scalable and focused comparison, balancing breadth with depth in evaluating GenAI code generation capabilities.

4.1.1 H1

In terms of code correctness, ChatGPT achieves the best results, with correct and partially correct results, followed by Copilot and Claude, whereas Gemini displays the poorest results (Table 3).

Table 3. Frequency Table for Code Correctness Results for the GenAIs

GenAI Tool	Correct	Partially Correct	Incorrect	Compilation Error	Runtime Error
ChatGPT	8	0	2	0	0
Copilot	2	4	3	1	0
Claude	1	4	4	1	0
Gemini	0	1	7	2	0

A graphical display is provided below (Figure 1).

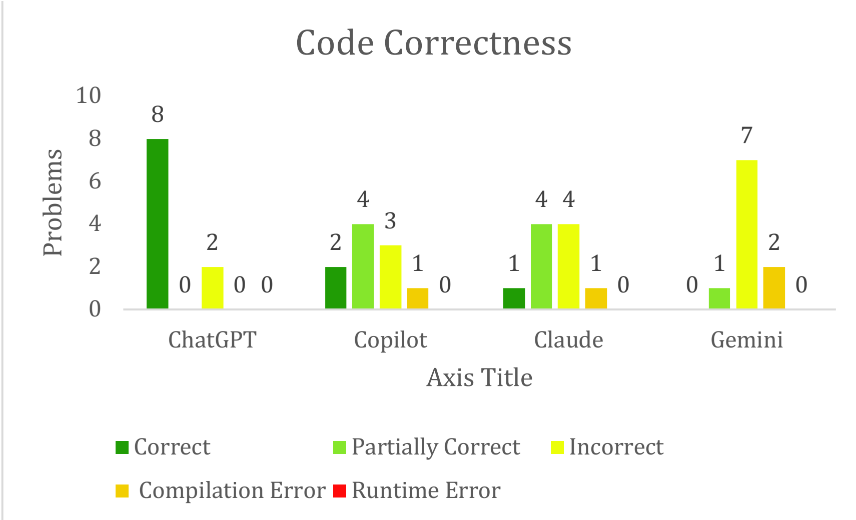

Figure 1. Code Correctness Results

4.1.2 H2

In terms of code readability, ChatGPT displays the best results, followed by Copilot and Gemini, whereas Claude displays the poorest results (Table 4).

Table 4. Code Readability Score

Question / Generative	1	2	3	10	11	12	24	25	26	27	Average score
ChatGPT	100%	100%	100%	90%	100%	80%	100%	90%	100%	100%	**96%**
Copilot	90%	90%	100%	80%	100%	80%	100%	100%	100%	100%	**94%**
Claude	100%	90%	100%	70%	100%	80%	100%	90%	90%	90%	**91%**
Gemini	100%	90%	100%	100%	100%	80%	90%	90%	100%	90%	**94%**

4.1.3 H3

In terms of time complexity, the four GenAI tools displayed similar results in half of the problems. However, overall, ChatGPT and Copilot provided superior results (Table 5).

Table 5. Time Complexity Results

GenAI	1	2	3	10	11	12	24	25	26	27
ChatGPT	O(n)	$O(n^2)$	O(n*log n)	$O(n^2)$	O(n*log n)	O(n*log n)	O(n*log n)	O(n)	O(n*log n)	O(n)
Copilot	O(n)	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n log n)	O(n)	O(n*log n)	O(n)	O(n*log n)	O(n)
Claude	O(n)	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n*log n + n*k)	____	O(n*log n)	O(n)	O(n*log n)	O(n)
Gemini	____	$O(n^2)$	O(n*k)	____	O(n*k)	O(n*log n)	$O(n^2)$	O(n)	$O(n^2)$	O(n)
	Draw	Draw	ChatGPT	Draw	ChatGPT/ Copilot	Copilot	Draw	Draw	Draw	Draw

In terms of space complexity, the four GenAI tools displayed similar results in approximately half of the problems. Though, overall, Copilot provided superior results, and Gemini's results were inferior (Table 6).

Table 6. Space Complexity Results

GenAI	1	2	3	10	11	12	24	25	26	27
ChatGPT	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(1)
Copilot	S:O(n)	S:O(n)	S:O(n)	S:O(n)	S:O(k)	S:O(n)	S:O(1)	S:O(n)	S:O(1)	S:O(1)
Claude	S:O(1)	S:O(n)	S:O(n)	S:O(n)	S:O(n*k)	____	S:O(n)	S:O(n)	S:O(1)	S:O(1)
Gemini	____	S:O(n)	S:O(n)	____	S:O(k)	S:O(n*logn)	S:O(n)	S:O(n)	S:O(n)	S:O(1)
	Claude	Draw	Draw	Draw	Draw	Copilot/ ChatGPT	Copilot	Draw	Copilot/ Claude	Draw

According to the findings presented, ChatGPT displayed the highest results in the two metrics, code correctness and code readability. In contrast, Copilot displayed the second-highest results in these metrics and also displayed the highest results in space complexity. In time complexity, both ChatGPT and Copilot displayed superior results. Thus, it can be concluded that ChatGPT and Copilot outperform, hence, they will be further compared in the succeeding benchmark phase.

4.2 Second Benchmark Phase

4.2.1 H1

Overall, ChatGPT displayed more correct answers, with 17 correct answers and 4 partially correct answers, than Copilot, with 12 correct answers and 3 partially correct answers. ChatGPT also displayed only 6 incorrect answers, in contrast to Copilot's 13 incorrect answers (Figure 2).

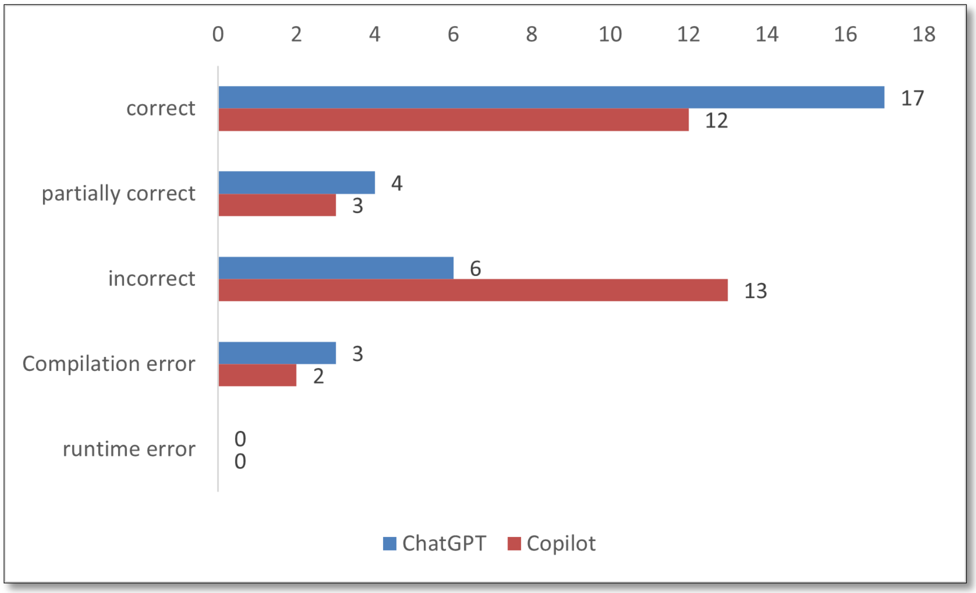

Figure 2. Frequency of Code Correctness Levels for ChatGPT and Copilot

However, Mann-Whitney U test statistical analysis results were insignificant (Table 7).

Table 7. Mann-Whitney U for Code Correctness

Mann-Whitney U test	
U	528.0
z-score	1.15
P	0.2148
Effect size r	0.2105

4.2.2 H2

Code readability construct, comprised of 10 items, was calculated for each of the 30 problems, and then an average score of the 30 problems was calculated for each of the two GenAI tools. The statistics description is presented below (Table 8).

Table 8. Descriptive Statistics for Code Readability

	ChatGPT	Copilot
Number of samples	30	30
Mean	82.33%	83.67%
Standard error of the mean	10.22%	10.79%
95% confidence interval for the mean	[0.623, 1.024]	[0.625, 1.048]
Median	80%	90%
Mode	70%	70%
Range	30%	30%
Interquartile range(IQR)	30%	30%
Sample standard deviation	0.799	0.837

A figure of the average score of the two GenAI tools core readability can be seen below (Figure 3).

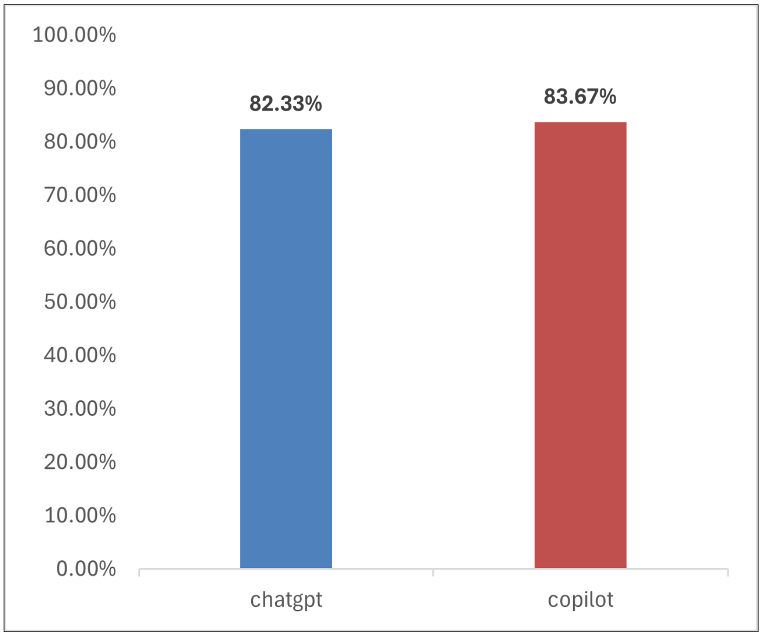

Figure 3. Average Score for Code Readability

A statistical analysis of code readability displayed insignificant results (Table 9).

Table 9. t-Test Results

t-Test: Two-Sample Assuming Unequal Variances		
	ChatGPT	**Copilot**
Mean	82.33%	83.67%
Variance	1.08%	1.21%
Observations	30	30
Hypothesized Mean Difference	0	
df	58	
t Stat	0.482872	
P(T<=t) one-tail	0.315502	
t Critical one-tail	1.671553	
P(T<=t) two-tail	0.631004	
t Critical two-tail	2.001717	

4.2.3 H3

Code efficiency consisted of time complexity and space complexity measures. Time complexity results distribution (Figure 4) and statistics description (Table 10) are presented below.

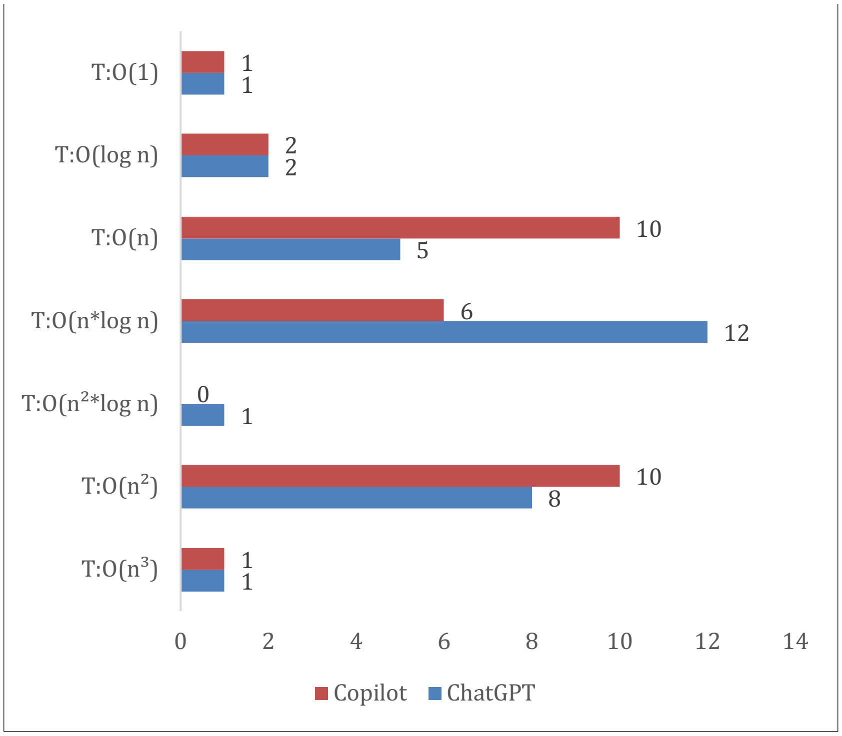

Figure 4. The Frequency of the Time Complexity Results

Table 10. Descriptive Statistics for Time Complexity

	ChatGPT	Copilot
Number of samples	30	30
25th Percentile	O(n)	O(n)
Median	O(n*log n)	O(n*log n)
75th Percentile	$O(n^2)$	$O(n^2)$
Mode	O(n*log n)	O(n)
Range	$(O(1), O(n^3))$	$(O(1), O(n^3))$

The Mann-Whitney U test analysis of time complexity (Table 11) displays insignificant results between ChatGPT and Copilot.

Table 11. Mann-Whitney U for Code Time Complexity

Mann-Whitney U test	
U	487.0
z-score	0.547
P	0.575
Effect size r	0.099

The second code efficiency measure, the space complexity results distribution (Figure 5), and statistics description (Table 12) are presented below.

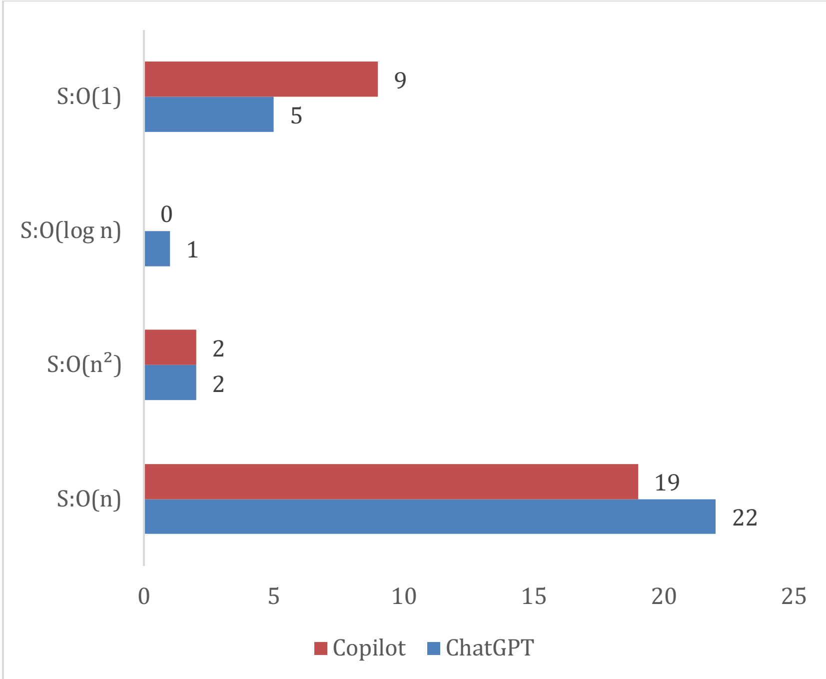

Figure 5. The Frequency of the Space Complexity Results

Table 12. Descriptive Statistics for Space Complexity

	ChatGPT	Copilot
Number of samples	30	30
25th Percentile	O(n)	O(n)
Median	O(n)	O(1)
75th Percentile	O(n)	O(n)
Mode	O(n)	O(n)
Range	(O(1), O(n^2))	(O(1), O(n^2))

The Mann-Whitney U test analysis of space complexity (Table 13) displays insignificant results between ChatGPT and Copilot.

Table 13. Mann-Whitney U for Code Space Complexity

Mann-Whitney U test	
U	496.5
z-score	0.687
P	0.405
Effect size r	0.125

A summary of the results, with relative strengths and weaknesses of each GenAI model across the three key quality criteria: correctness, readability, and efficiency, is presented below (Table 14).

Table 14. Descriptive Statistics for Key Code Quality Metrics

GenAI Tool	Correctness	Readability	Efficiency
ChatGPT	Strong — Highest number of correct outputs in both benchmarks	Strong — High readability scores; well-structured and commented code	Moderate — Competitive time complexity; slightly less optimal space usage
Copilot	Moderate — Second-highest correctness; some partially correct outputs	Strong — Comparable to ChatGPT; good use of concise coding conventions	Strong — Best space complexity; efficient code generation patterns
Claude	Weak — High error rate; struggled with medium and hard problems	Weak — Lower readability scores; inconsistently structured code	Moderate — Acceptable complexity in some cases but inconsistent
Gemini	Weak — Frequent incorrect or incomplete solutions	Moderate — Reasonable formatting but limited use of conventions	Weak — Struggled with complexity; inefficient strategies observed

5 Discussion and Conclusion

5.1 Discussion

This study investigated the code generation capabilities of four leading GenAI tools—ChatGPT, Copilot, Claude, and Gemini—across three primary quality dimensions: correctness, readability, and efficiency. These dimensions were examined in relation to three hypotheses (H1–H3) using a two-phase benchmarking design. The expanded results, while confirming some performance differences, also reveal a broader picture about the current maturity of GenAI coding tools.

H1 asked whether a significant difference exists in code correctness between GenAI tools. In both the initial and extended benchmarks, ChatGPT outperformed Copilot and the other tools in generating completely correct code solutions. For example, in the second phase, ChatGPT produced 17 correct responses compared to Copilot's 12. However, despite this numeric advantage, the Mann-Whitney U test showed no statistically significant difference ($p = 0.21$), meaning the observed performance gap could be due to random variation rather than a systematic effect. From a practical standpoint, this lack of significance suggests that both ChatGPT and Copilot can be considered equivalent in producing functionally correct Python code for algorithmic tasks. For developers and educators, this implies that tool selection should consider other factors such as user interface, integration into development environments, and licensing rather than correctness alone.

H2 explored whether the GenAI tools differ significantly in code readability. The readability evaluation, based on a structured checklist of formatting and clarity guidelines, revealed that both ChatGPT and Copilot scored highly, averaging 82.3% and 83.7%, respectively. Once again, statistical analysis using a two-sample t-test indicated no significant difference between the two tools ($p = 0.63$). This finding reinforces the idea that modern GenAI tools have internalized many code readability conventions, likely due to training on high-quality repositories such as GitHub. For software practitioners, this means GenAI tools can be reliably used for generating not only correct but also maintainable code - at least for short to medium-length algorithmic solutions. However, the marginal difference in average scores may still matter in specific scenarios where strict readability guidelines are enforced (e.g., onboarding junior developers or teaching programming).

H3 assessed the efficiency of code in terms of time and space complexity. Both tools demonstrated a strong grasp of algorithmic efficiency, frequently producing solutions with reasonable time and space complexities (e.g., $O(n)$, $O(n \log n)$). However, as with correctness and readability, the observed differences were not statistically significant, according to the Mann-Whitney U tests for both time ($p = 0.575$) and space complexity ($p = 0.405$). This finding has two implications: First, GenAI tools are now capable of producing not only functional but also reasonably optimized code out of the box, in terms of time and space complexities. Second, they may be similarly limited when it comes to edge cases, or when required to fine-tune for performance—suggesting the need for human review when optimization is critical (e.g., in production systems or real-time applications).

While the study included problems across three difficulty levels (easy, medium, hard), no formal statistical analysis was conducted to examine how performance varied by difficulty. However, some qualitative observations suggest patterns worth noting. For example, ChatGPT was the only tool to solve a hard-level problem correctly in some related studies (Idrisov & Schlippe, 2024), and this may also hold in the present benchmark, while Claude and Gemini often struggled with medium and hard problems. A finer-grained analysis (e.g., performance stratified by difficulty) could be a valuable direction for future work.

As highlighted in the Introduction, each tool has its strengths and weaknesses. Based on the findings, ChatGPT shows strength in correctness and code structure, making it especially valuable for novice programmers or educational settings where clarity and explanation matter. Copilot excels in code efficiency and brevity (Yetistiren et al., 2022; Yetistiren et al., 2023; Nguyen & Nadi, 2022), benefiting professional developers who prioritize speed. Claude and Gemini, while innovative, currently lag behind in both correctness and efficiency, though their multimodal capabilities suggest potential for broader applications beyond code. Thus, tool selection should be context-specific, not one-size-fits-all. Developers should consider the nature of their task —whether educational, prototyping, or production deployment— when choosing a GenAI assistant.

5.2 Conclusion

The evaluation may imply that GenAI for code generation has matured, with ChatGPT leading in terms of code correctness and readability. These results support the potential for GenAI tools to serve as effective assistants in the software development process, reducing the time and effort required to produce syntactically correct and maintainable code. However, the competitive performance of Copilot in efficiency metrics also highlights that there is no single "best" solution; rather, each tool offers unique strengths that could be leveraged depending on the context. The overall conclusion is that while AI-generated code can achieve high quality under certain conditions, human oversight remains essential to ensure that the generated code meets all functional and performance requirements.

5.3 Limitations

Despite the promising results, several limitations should be noted:

- Sample Size and Diversity: The evaluation was based on a set of 40 coding problems from LeetCode, which may not fully represent the diversity of real-world programming challenges. Future studies should include a broader array of problems across various domains.
- Selection bias: The two top-performing GenAI tools from the first benchmark phase were analyzed further in the second phase. While the initial phase included questions of varying programming problem complexities, a different mix of questions might yield different outcomes.
- Language and Environment Specificity: The study focused solely on Python code generation, limiting the generalizability of the findings to other programming languages and development environments.
- Metric Scope: Although correctness, readability, and efficiency are critical, additional quality dimensions such as maintainability, security, and scalability were not fully explored.
- Dynamic Nature of GenAI: GenAI tools are rapidly evolving. The performance of these tools may change with subsequent updates, meaning that the results of this study represent a snapshot in time.

Future research using more diverse datasets, additional languages, and expanded code quality metrics, such as maintainability, security, and scalability, could further illuminate GenAI's potential in supporting coding activities.

6 References

Anthropic. (2024, March 4). Introducing the next generation of Claude. https://www.anthropic.com/news/claude-3-family.

Boehm, B. W. (1978). Characteristics of Software Quality, 1st edition, North-Holland Publishing Company.

Börstler, J., Bennin, K. E., Hooshangi, S., Jeuring, J., Keuning, H., Kleiner, C., ... & van Assema, J. (2023). Developers talking about code quality. Empirical Software Engineering, 28(6), 128.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd edition). MIT Press and McGraw-Hill.

Cui, J., Zhang, R., Zhou, F., Li, R., Song, Y., & Gehringer, E. (2024, July). How Much Effort Do You Need to Expend on a Technical Interview? A Study of LeetCode Problem Solving Statistics. In 2024 36th International Conference on Software Engineering Education and Training (CSEE&T) (pp. 1-10). IEEE.

Dantas, C. E. C., & Maia, M. A. (2021). Readability and understandability scores for snippet assessment: An exploratory study. arXiv preprint arXiv:2108.09181.

Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2023). Generative AI. Business & Information Systems Engineering, 66(1), 111–126. https://doi.org/10.1007/s12599-023-00834-7

Han, Y. (2002, May). Deterministic sorting in O (n log log n) time and linear space. In Proceedings of the thirty-fourth annual ACM symposium on theory of computing (pp. 602-608).

Hansson, E., & Ellréus, O. (2023). Code correctness and quality in the era of AI code generation: Examining ChatGPT and GitHub copilot.

Hörnemalm, A. (2023). ChatGPT as a software development tool: The future of development. Master's Thesis, Umeå University, Faculty of Science and Technology, Umeå, Sweden, 2023. Available online: https://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-209909

Idrisov, B., & Schlippe, T. (2024). Program code generation with generative AIs. Algorithms, 17(2), 62.

Imran, M., & Almusharraf, N. (2024). Google Gemini as a next generation AI educational tool: a review of emerging educational technology. Smart Learning Environments, 11(1), 22.

International Organization for Standardization. (2001). ISO/IEC 9126-1:2001: Software engineering—Product quality—Part 1: Quality model. Geneva, Switzerland: ISO.

Microsoft. (2025, August 18). Overview of Microsoft 365 Copilot Chat. Microsoft Learn. https://learn.microsoft.com/en-us/copilot/overview

Martin, R. C. (2013). Clean Code Refactoring, Patterns, Testing, and Techniques for Clean Code: German Edition. MITP-Verlags GmbH & Co. KG.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in Software quality, Rome Air Develop. Center Report. TR-77-369.

Moritz, E. (2024). Chatting with chat (GPT-4): Quid Est understanding. London, Canada: Centre for Digital Philosophy at Western University.

Nguyen, N., & Nadi, S. (2022, May). An empirical evaluation of GitHub copilot's code suggestions. In Proceedings of the 19th International Conference on Mining Software Repositories (pp. 1-5).

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2025). Asleep at the keyboard? assessing the security of github copilot's code contributions. Communications of the ACM, 68(2), 96-105.

Peñalvo, F. J. G., & Ingelmo, A. V. (2023). What do we mean by GenAI? A systematic mapping of the evolution, trends, and techniques involved in Generative AI. IJIMAI, 8(4), 7-16.

Sobania, D., Briesch, M., & Rothlauf, F. (2022, July). Choose your programming copilot: a comparison of the program synthesis performance of github copilot and genetic programming. In Proceedings of the genetic and evolutionary computation conference (pp. 1019-1027).

Strobel, G., Banh, L., Möller, F., & Schoormann, T. (2024). Exploring Generative Artificial Intelligence: A Taxonomy and Types. Proceedings of the HICSS.

Tonella, P., & Abebe, S. L. (2008). Code quality from the programmer's perspective. Proceedings of Science, XII Advanced Computing and Analysis Techniques in Physics Research, Erice, Italy, 5, 153.

Yetistiren, B., Ozsoy, I., & Tuzun, E. (2022, November). Assessing the quality of GitHub copilot's code generation. In Proceedings of the 18th international conference on predictive models and data analytics in software engineering (pp. 62-71).

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

Appendix A: LeetCode Questions

A supplementary dataset of 40 LeetCode programming problems, categorized by difficulty level (13 easy, 14 medium, and 13 hard), is available for reference here. These problems evaluate foundational coding skills, algorithmic problem solving, and advanced technical mastery, respectively.