International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

A Comparative Study of Human-Written Code and AI-Generated Code

Vaishnavi Jadhav,
Mr. Swaraj Sanjay Malve (Assistant Professor)
Department of Master of Computer Application,
JCEI'S JAIHIND INSTITUTE OF MANAGEMENT AND RESEARCH (JIMR)

Artificial Intelligence (AI) is becoming a familiar part of software development. Tools that turn everyday-language instructions into code can help programmers build solutions, find errors, and handle repetitive tasks. Even so, an important question remains: how does AI-generated code compare with code written by a person in terms of correctness, quality, efficiency, and ease of maintenance? This study explores that question by looking at the strengths and limitations of both approaches.

The comparison considers functional correctness, readability, execution time, memory use, maintainability, security, and the effort needed to develop a working solution. It also considers problems that can appear in AI-generated code, such as faulty logic, inefficient implementation, and possible security weaknesses. Looking at these factors together can help show when AI tools are useful and when careful human review is still needed. The study treats AI as a programming aid, not as a substitute for testing and technical judgment.

Keywords: *Human-Written Code, AI-Generated Code, Artificial Intelligence, Software Development, Code Quality, Code Efficiency, Code Maintainability, Automated Code Generation.*

I.INTRODUCTION:

Artificial Intelligence (AI) is changing the way people approach software development. AI-powered coding tools can now generate code from written instructions, suggest ways to complete a program, and help identify errors. These features can be useful for students, developers, and organizations, particularly when a task involves repetitive work. However, the convenience of generating code quickly also raises questions about whether that code is correct, efficient, secure, and easy to maintain.

For many years, programmers have relied on their knowledge, reasoning, and problem-solving skills to turn requirements into working software. Their code reflects their experience and understanding of the task. AI-generated code is produced by models trained on large collections of code and other data. These tools can produce a solution in seconds, but the result may still contain logical mistakes, inefficient steps, security weaknesses, or details that do not match the requirements. For this reason, the two approaches need to be compared in practical and consistent ways.

A program should not be judged only by how quickly it is written. It also needs to produce the right results, be understandable to other developers, use resources sensibly, and remain safe and manageable when changes are needed. Code that works for one input may fail on another, and code generated quickly may need extra testing or debugging before it is ready to use. Comparing human-written and AI-generated code can help make these trade-offs clearer.

1.1 Background of the Study:

AI coding assistants offer several kinds of support: they can suggest code, draft functions, explain programming concepts, and help developers investigate errors. This can make some programming tasks quicker and help beginners get started. However, people are still needed to understand the full requirements, choose suitable approaches, check edge cases, and confirm that the final program behaves as expected.

The two approaches may differ in development time, structure, correctness, and ease of maintenance. Their results depend on the task, the experience of the person writing the code, the AI tool and prompt used, and the way the

solution is tested. It would therefore be misleading to assume that either human-written or AI-generated code is always better. A fair comparison needs to apply the same criteria to both.

1.2 AI coding tools are being adopted quickly, but their speed does not automatically mean that their output is as reliable or maintainable as code written by a person. Human-written code can also contain mistakes or inefficient choices. This study therefore compares both approaches using defined software-quality measures, with the aim of identifying where each performs well and where it has limitations.

The growing adoption of AI coding tools has created a need to understand whether AI-generated code can provide the same level of correctness, quality, efficiency, and maintainability as code written by human programmers. Although AI tools can produce code quickly, faster code generation does not necessarily guarantee accurate, secure, or reliable software. Human-written code can also contain errors and inefficiencies, making a fair comparison important. The main problem addressed by this study is to examine the differences between human-written and AI-generated code using appropriate evaluation criteria and to identify the strengths and limitations of each approach.

The main objectives of this study are to compare the correctness of human-written and AI-generated code; examine their readability, structure, and maintainability; measure execution time and memory use under comparable conditions; identify programming errors and possible security issues; compare the effort needed to develop, test, and debug solutions; and understand how AI tools can support programmers in practice.

1.3 Research Objectives:

This study asks whether AI-generated code is as functionally correct as human-written code, and whether the two approaches differ in readability, structure, maintainability, execution time, or memory use. It also considers the kinds of errors and security weaknesses that may occur, whether AI tools reduce total development time after testing and debugging are included, and which situations are best suited to human programming, AI assistance, or a combination of both.

I. To compare the functional correctness of human-written code and AI-generated code.
II. The study compares selected programming tasks using common evaluation criteria, including correctness, readability, performance, memory use, maintainability, security, and development effort. Wherever possible, both versions of each solution will use the same programming language, problem statement, test cases, and execution environment. The conclusions will apply to the tasks and conditions actually tested; they should not be taken as a verdict on every AI tool, programming language, or software project.
III. To examine differences in execution time and memory consumption under comparable testing conditions.
IV. To identify common programming errors and potential security issues in both types of code.
V. To compare the development effort required to write, test, debug, and improve code.
VI. Research into AI-assisted programming has grown as coding tools have become more widely available. These tools can turn natural-language instructions into code and help developers complete programming tasks. At the same time, researchers have raised questions about the reliability, security, readability, and long-term maintainability of the generated results. The studies reviewed below offer useful evidence about both the benefits and the limitations of these tools.
VII. To understand how AI-generated code can be used effectively as a support tool in software development.

1.4 Research Questions:

This study aims to address the following research questions:

I. How does the functional correctness of AI-generated code compare with human-written code?
II. Are there differences in readability, code structure, and maintainability between the two approaches?
III. How do their execution time and memory consumption compare under the same conditions?
IV. What types of errors and potential security weaknesses can occur in human-written and AI-generated code?
V. Can AI coding tools reduce overall development time when testing and debugging are also considered?
VI. In which situations is human-written code, AI-generated code, or a combination of both more suitable?

1.5 Scope of the Study:

This study focuses on comparing human-written code and AI-generated code through selected programming tasks and relevant software quality criteria. The comparison may include functional correctness, readability, execution performance, memory usage, maintainability, security, and development effort. To maintain fairness, both approaches should be evaluated using the same programming language, problem statements, test cases, and execution environment wherever applicable.

The study also considers the limitations of AI-generated code, particularly the possibility of incorrect logic, inefficient implementation, and security-related weaknesses. The scope is limited to the selected programming tasks, AI coding tool, evaluation methods, and testing conditions used in the research. The findings should therefore be interpreted in relation to the experimental setup rather than treated as a conclusion about every programming language, AI model, or software development project.

II.LITERATURE OF REVIEW:

The use of Artificial Intelligence (AI) in software development has attracted considerable attention from researchers in recent years. AI-powered coding tools can generate program code from natural-language instructions, suggest code completions, and assist developers in solving programming problems. Although these tools offer the possibility of faster development, questions remain about the correctness, security, readability, and maintainability of the code they produce. Several studies have examined these aspects and provided useful insights into the advantages and limitations of AI-generated code.

2.1 Existing Research:

Yetiştiren et al. (2023) conducted an empirical study to evaluate the quality of code generated by AI coding tools, including GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. The researchers used the HumanEval dataset to assess code validity, correctness, security, reliability, and maintainability. Their findings showed that the tools differed in their ability to generate correct solutions and that AI-generated code still required careful evaluation. The study highlighted the importance of examining multiple software quality factors rather than judging code only by its ability to execute successfully [1].

Moradi Dakhel et al. (2023) investigated GitHub Copilot as an AI pair programmer and compared its generated solutions with human-written solutions for programming tasks. Their study found that Copilot could solve many fundamental programming problems, but some generated solutions contained errors or were difficult to reproduce consistently. The comparison also indicated that human solutions had a higher proportion of correct answers in the evaluated tasks, while some incorrect AI-generated solutions required less effort to repair. The researchers emphasized that AI coding assistants can be useful to experienced developers, but beginners may struggle to recognize errors in generated code [2].

A study by Pearce et al. examined security weaknesses in code generated by GitHub Copilot. The research demonstrated that AI-generated code could contain security vulnerabilities across different programming scenarios. The findings raised concerns about relying on automatically generated code without appropriate security checks, testing, and review. This research is particularly relevant because code may appear functional while still containing weaknesses that could affect the safety and reliability of software systems [3].

Majdinasab et al. (2023) conducted a replication study to reassess the security of code generated by GitHub Copilot. Their work examined Python code and reported that the proportion of vulnerable suggestions decreased from 36.54% in the earlier study to 27.25% in the replication. Despite this improvement, the researchers found that insecure suggestions continued to occur. Their findings support the need for regular security assessment of AI-generated code as coding tools evolve [4].

Schreiber and Tippe (2025) studied security vulnerabilities in AI-generated code collected from public GitHub repositories. They analyzed 7,703 files attributed to several AI coding tools using CodeQL static analysis. The researchers identified 4,241 instances associated with Common Weakness Enumeration (CWE) categories. Their results showed that most analyzed files did not contain identifiable CWE-mapped vulnerabilities, but security risks still existed and varied across programming languages and tools. This study highlights the importance of evaluating AI-generated code in real-world development settings rather than relying only on controlled programming exercises [5].

Author(s) and Year	Research Focus	Method Used	Major Findings
Moradi Dakhel et al. (2023)	AI-assisted programming versus human solutions	Comparative programming experiments	AI could solve many tasks, but some solutions contained errors; human solutions performed better in correctness in the evaluated tasks.
Yetiştiren et al. (2023)	AI-generated code quality.	HumanEval benchmark and code quality metrics	AI tools differed in correctness, reliability, and maintainability
Pearce et al. (2022)	Security of AI-generated code	Security analysis of generated code	AI-generated code could contain security weaknesses.
Majdinasab et al. (2023)	Security of Copilot-generated Python code	Targeted replication and security analysis	Vulnerability rates decreased compared with the earlier study, but insecure suggestions remained.
Schreiber and Tippe (2025)	Security of AI-generated code in public repositories	Large-scale repository analysis using CodeQL	Security findings varied across languages and tools, supporting the need for code review.

2..2 Comparison of Existing Methods:

The reviewed studies evaluate AI-generated code from different perspectives. Some focus on functional correctness and the ability to solve programming problems, while others examine code quality, developer effort, and security vulnerabilities. The following table summarizes their major contributions.

The studies show that AI-generated code has useful capabilities but cannot automatically be considered equivalent to human-written code in every situation. The results also demonstrate that code quality depends on the programming task, the coding tool, the programming language, and the evaluation method.

2.3 Research Gap:

Earlier studies have examined AI-generated code for correctness, overall quality, and security. However, many focus on a particular benchmark, a specific coding assistant, or one aspect of software quality. As a result, it can be difficult

to see how human-written and AI-generated solutions compare across several measures when they are tested under the same conditions.

A useful comparison should give both approaches the same programming problems, test cases, language, and execution environment. It should look beyond whether the program runs and also consider readability, maintainability, execution time, memory use, security, and the time spent developing and fixing the solution.

This study responds to that need by applying a consistent set of evaluation criteria to human-written and AI-generated code. The aim is not to declare one approach the winner in every situation, but to identify their practical strengths and limitations and explain why testing and human review remain important when AI tools are used.

III.RESEARCH METHODOLOGY:

This study uses a comparative approach to examine human-written code alongside AI-generated code. Both approaches will be assessed on the same types of programming tasks, using shared criteria for correctness, readability, execution time, memory use, maintainability, security, and development effort. Keeping the testing conditions as consistent as possible will make the comparison more meaningful.

3.1 Research Design

The research uses an experimental comparison. Solutions written by a human programmer and solutions generated by an AI coding tool will be prepared for selected programming problems. They will then be tested with the same test cases and, where applicable, the same programming language and execution environment.

The analysis will cover correctness, code quality, performance, and potential security issues. It will also record development and debugging effort, since generating code quickly does not necessarily mean that a verified solution takes less time overall. The comparison will be based on the observed results rather than an assumption that one approach must be better.

3.2 Data Collection

Data will be gathered by writing and testing code for selected programming problems. Human-written solutions will be prepared independently, while an AI coding assistant will generate solutions to the same requirements. Using matching problem statements will help keep the comparison fair.

For each solution, the study will record the source code, test outcomes, execution time, memory use, code-quality observations, identified errors, and time spent developing or debugging it. Published research will also be reviewed to inform the evaluation criteria and help put the results into context.

3.3 Dataset

The dataset will contain selected programming problems and a human-written and AI-generated solution for each one. The tasks may cover calculations, string manipulation, array operations, searching, sorting, and other common programming activities.

Each task will have an expected result and a shared set of test cases. Both versions will receive the same inputs and be evaluated under the same conditions. The recorded measures will include passed tests, execution time, memory use, code-quality observations, and any errors found.

The number of tasks and code samples will be decided before testing begins. Defining the dataset in advance helps reduce the risk of choosing examples that unintentionally favor one approach.

3.4 Tools and Technologies

The following tools and technologies are proposed for conducting the study:

- **Python:** To implement programming solutions and conduct performance experiments.
- **AI Coding Tool:** ChatGPT or another selected AI coding assistant will be used to generate code from predefined programming instructions.
- **pytest:** To execute automated test cases and evaluate functional correctness.
- **Visual Studio Code**: To write, edit, execute, and organize the code samples.
- **Git:** To maintain version history and document changes made during development and debugging.
- **Code Quality Analysis Tools:** Appropriate static analysis tools will be used to examine code structure, complexity, and potential issues.
- Spreadsheet Software: Microsoft Excel or an equivalent tool will be used to organize measurements, prepare comparison tables, and generate graphs.

The software versions and execution conditions will be recorded so that the experiment can be repeated as closely as possible. These are the tools planned for the study; the final report should name the tools that were actually used.

3.5 Proposed Method / Framework

The comparison will follow a set of steps so that both types of code are assessed in a consistent way.

Step 1: Selection of Programming Problems

A set of programming problems will be selected to cover different types of operations and levels of complexity. Each problem will have clearly defined requirements and expected outputs.

Step 2: Development of Human-Written Code

Human programmers will develop solutions independently according to the given problem statements. Their development time and any corrections required during testing will be recorded.

Step 3: Generation of AI-Written Code

The same programming problems will be provided to the selected AI coding tool using predefined instructions. The generated solutions will be saved and evaluated under the same conditions as the human-written solutions.

Step 4: Functional Testing

Both sets of code will be tested using identical test cases, including normal inputs, boundary conditions, and invalid inputs where applicable. The number of successful test cases will be recorded.

Step 5: Performance Evaluation

The code samples will be evaluated for execution time and memory consumption using the same hardware and software environment. Multiple runs may be performed to reduce the influence of measurement variation.

Step 6: Code Quality and Security Analysis

The solutions will be examined for readability, complexity, maintainability, and potential security weaknesses. Where possible, automated analysis will be supplemented by manual review.

Step 7: Comparative Analysis

The collected measurements will be organized into tables and graphs. The results will be compared to identify differences, advantages, limitations, and patterns across the selected programming tasks.

3.6 Algorithms and Evaluation Metrics

The study does not require a new algorithm of its own. Instead, it uses existing programming solutions and established measurement techniques to compare the code samples.

The following evaluation metrics will be considered:

1. Functional Correctness: Measured by the proportion of test cases passed successfully.

Functional correctness (%) = (Number of test cases passed / Total number of test cases) × 100

2. Execution Time: The time taken by a program to execute under the specified testing conditions. Lower execution time indicates faster execution for the measured task.
3. Memory Consumption: The amount of memory used during execution. Lower memory usage may indicate more efficient resource utilization for the particular task.
4. Code Readability and Maintainability: Evaluated using consistent criteria such as naming conventions, code organization, duplication, clarity, and ease of modification.
5. Security: Potential security weaknesses will be identified using suitable static analysis tools and manual inspection. Security findings will be reported separately from functional correctness.
6. Development Effort: The time required to produce a working solution, including testing, debugging, and corrections. For AI-generated code, the time spent reviewing and fixing generated output will also be included.

Taken together, these measures provide a more complete picture than execution speed alone. Results will be considered task by task, and the limits of each measurement will be kept in mind.

3.7 System Architecture / Workflow

Both types of solution will go through the same general workflow. After the programming problems are selected, the human-written and AI-generated code will be tested and assessed separately before their results are compared.

The workflow is as follows:

Programming Problem Selection → Human-Written Code and AI-Generated Code → Common Testing Environment → Functional Testing and Performance Measurement → Code Quality and Security Analysis → Data Collection → Comparative Analysis → Research Findings.

Using a shared workflow helps keep the evaluation consistent and reduces the influence of differences in testing conditions. The results will then be used to discuss what each approach does well and where it may fall short.

3.8 Reliability and Validity

Reliability will be supported by applying the same problem statements, test cases, and execution conditions to both approaches. Where appropriate, performance will be measured more than once, and the same criteria will be used for every code sample.

To support validity, the tasks will be selected carefully, evaluation measures will be defined before testing, and the tools and conditions will be documented. The study will also acknowledge factors that may affect the results, including task selection, the human programmer's experience, the AI tool used, and differences between languages or development environments.

3.9 Ethical Considerations

Academic integrity will be maintained by acknowledging AI assistance and citing relevant publications. Generated code will be tested and reviewed rather than accepted as correct by default. External resources will be credited, and the results will be reported honestly, including outcomes that differ from expectations.

3.10 Objectives of the Methodology

The methodology is designed to make the comparison fair and transparent. It provides a practical way to collect evidence about code quality and development effort, while helping explain where AI coding tools can be useful and why human judgment, testing, and verification still matter.

IV. IMPLEMENTATION AND EXPERIMENTAL SETUP

This phase describes how the code samples will be prepared and tested. Both approaches need to be evaluated under similar conditions so that the results reflect differences in the implementations as far as possible, rather than differences in the testing setup.

4.1 Experimental Environment

Python will be used for the experiment because it supports quick development, automated testing, and code-quality checks. Visual Studio Code will serve as the development environment, and suitable Python testing libraries will be used to check program outputs against the expected results.

A selected AI coding assistant, such as ChatGPT, will generate solutions for the chosen tasks. Human-written solutions will be created independently from the same problem statements. Both sets will run on the same computer with the same Python version and software configuration.

Details such as the processor, installed RAM, operating system, Python version, and AI tool will be recorded during implementation. Keeping a record of these details will make it easier for others to understand the conditions under which the results were obtained.

4.2 Selection of Programming Tasks

The selected tasks will cover different types of programming operations and levels of difficulty. Examples may include arithmetic, string manipulation, array processing, searching, sorting, and basic data processing.

Each task will have a clear problem statement, expected output, and test cases. The same tasks will be used for both approaches so that neither receives an easier set of requirements.

4.3 Implementation of Human-Written Code

For each task, a human programmer will write a solution using the stated requirements and normal development tools. AI-generated code will not be used to create the initial human-written solution.

Each solution will be saved separately. Where it is measured during the experiment, the time spent writing, testing, and debugging the code, along with the number of corrections, will be recorded to estimate the effort needed to produce a working result.

4.4 Implementation of AI-Generated Code

The same problem statements will be entered into the selected AI tool using prompts that specify the language, required behaviour, inputs, outputs, and relevant constraints.

The first version produced by the AI will be saved before any edits. It will then be tested with the same test cases used for the human-written version. Errors, corrections, and any extra work needed to make it function will be documented.

For a fair comparison, repeated prompts or replacement outputs will not be used selectively without being recorded. Any follow-up prompt, edit, or correction will be documented as part of the process.

4.5 Testing and Validation

Both sets of solutions will go through the same testing procedure. Tests will cover ordinary inputs, boundary cases, and invalid inputs where relevant, with expected results defined before the code is assessed.

A tool such as pytest can run the tests and record whether each output matches the expected result. For a given task, a solution will be treated as functionally correct only if it meets the stated requirements and passes the applicable tests.

Testing can reveal runtime failures, incorrect logic, unexpected behaviour, and problems that appear only with certain inputs. These observations will be recorded for later comparison.

4.6 Performance Measurement

Execution time and memory use will be measured under comparable conditions. Each program may be run several times, with the average or another clearly specified summary used to reduce the effect of temporary changes in system activity.

A suitable profiling or monitoring tool will be used to estimate memory use. As far as possible, the same hardware, input data, and execution settings will be used for each solution, and the measurements will be recorded task by task.

Performance results need to be considered alongside correctness and quality. A program that runs faster is not necessarily the better solution if it produces incorrect results or is harder to maintain or raises security concerns.

4.7 Code Quality and Security Evaluation

Code quality will be reviewed using consistent criteria, including clarity, naming, organization, duplication, complexity, and ease of future changes. Static analysis tools may also help identify potential problems.

The security review will focus on weaknesses relevant to the selected tasks. Automated checks may be combined with manual inspection, and any findings will be documented so that the two approaches can be compared.

A static-analysis warning does not, by itself, prove that a vulnerability exists. Each finding will be reviewed and described as a potential issue or a verified problem, as appropriate.

4.8 Data Recording and Comparison

Measurements will be organized in tables or spreadsheets. For each task, the record will include the code type, test cases passed, execution time, memory use, quality observations, security findings, and development or debugging effort where those times have been measured.

Tables, graphs, and descriptive statistics will be used to summarize the results. Statistical comparisons will be considered only if the number and type of observations support them. Differences between individual tasks will be discussed rather than relying only on one overall score.

4.9 Experimental Limitations

The results may be influenced by the number and difficulty of the tasks, the human programmer's experience, the selected AI tool, and the prompts used. Runtime measurements can also vary with system activity and other environmental factors.

Some aspects of readability and maintainability require human judgment. To make those assessments more consistent, the study will use stated criteria and document its review process. Any conclusions will remain limited to the tasks and conditions tested.

V. RESULTS AND DISCUSSION

This section presents the results from a local benchmark of five human-written and AI-generated Python implementations. Each pair was tested with equivalent inputs, and console input/output was excluded from the timed functions. These results describe only the implementations tested here; they should not be treated as a general ranking of human-written and AI-generated code.

5.1 Functional Correctness

Each implementation was checked against expected results defined in advance. For the multiplication-table task, the tests checked the ten products generated. The other tasks checked the expected classification, reversed string, factorial result, or vowel count. The planned test suite contains 34 cases in total.

The reported correctness percentages apply only to these functional test cases. A negative factorial input was treated as an expected rejection and represented by None in the benchmark functions. Invalid text entered at an input prompt was not included in these function-level tests and would need to be recorded separately if tested.

5.2 Execution Time and Memory

Execution time was measured with Python's timeit module. Five timing runs were completed for each implementation, with 1,000 function calls in each run; the median time per call is reported in microseconds. Peak traced Python allocation was measured separately with tracemalloc, so it should not be interpreted as total process RAM.

The performance differences depended on the particular implementation and input. For the string task, slicing required less work than converting the string to a list and joining it again. The built-in and iterative factorial versions also showed different timing and memory-allocation patterns. These observations describe the code samples tested, not the general abilities of human programmers or AI tools.

5.3 Code Quality

Readability, naming, simplicity, maintainability, and algorithmic efficiency were assessed using a preliminary manual rubric, with scores from 1 (weak) to 5 (strong). These scores reflect the reviewer's judgment rather than an automated measurement. A stronger evaluation would define the rubric in detail and involve more than one independent reviewer.

5.4 Security Review

The manual review did not find shell commands, file operations, network access, or dynamic code execution in these short examples. However, the numeric programs may raise ValueError if a user enters non-integer text, and the factorial examples do not limit the size of the input. These are input-handling and resource-management concerns, not confirmed exploitable vulnerabilities. No automated static-analysis scan was performed for this revision.

5.5 Development Time

Development time was not recorded while the original code was being written. For that reason, this study does not report numerical comparisons for initial coding, review, testing, debugging, or total time to a verified solution. Those measurements would need to be collected in a timed follow-up experiment rather than estimated after the fact.

5.6 Summary and Limitations

In this benchmark, the two implementations sometimes showed different execution-time and memory-use patterns even though they solved the same task. Correctness results are limited to the defined test set, the quality scores depend on manual judgment, and the security review involved source inspection only. The experiment covered five introductory Python tasks in one local environment, so it cannot support broad claims about all programmers or AI systems. Future work could include more tasks, several human participants, repeated AI generations, larger inputs, automated quality and security checks, and development time recorded as the work happens.

For reproducibility, the recorded environment was Linux-6.18.44-x86_64-with-glibc2.41 and Python 3.13.5. Timing and memory results may differ across computers, Python builds, system loads, and input sizes.

VI. CONCLUSION:

AI has made it easier to generate code, explore possible solutions, and reduce some repetitive programming work. These tools can be helpful for students and experienced developers, but their growing use also raises practical questions about correctness, efficiency, security, and maintainability. This study compares human-written and AI-generated code to better understand where each approach may be useful and where caution is needed.

The comparison considers correctness, execution time, memory use, readability, maintainability, security, and development effort. These factors matter because useful software must do more than produce the expected answer once: it should also be understandable, reasonably efficient, secure, and straightforward to update. Even when AI produces code quickly, testing and debugging may still be needed before the solution can be relied on.

Human programmers bring reasoning, context, creativity, and an understanding of project-specific needs. AI coding tools can offer quick drafts, suggestions, and help with routine tasks. Neither approach should be assumed to be best in every case. The outcome depends on the problem, the implementation, the programmer's experience, and how carefully the solution is tested.

The main takeaway is that AI can be a useful assistant, but it should not replace careful programming judgment. Combining AI-generated suggestions with human review, testing, and verification may help developers work more effectively while still producing reliable, maintainable software. Any firm conclusion about comparative performance must be based on the benchmark results actually collected, with the study's limitations clearly acknowledged.

VII.REFERENCES:

1. B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün, "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*, 2023. Available: https://arxiv.org/abs/2304.10778
2. A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "GitHub Copilot AI pair programmer: Asset or Liability?" *Journal of Systems and Software*, vol. 203, article 111734, 2023. doi: 10.1016/j.jss.2023.111734.
3. H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," in *Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP)*, pp. 754–768, 2022. doi: 10.1109/SP46214.2022.9833571.
4. C. Chen et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021. Available: https://arxiv.org/abs/2107.03374

5. J. Austin et al., "Program Synthesis with Large Language Models," *arXiv preprint arXiv:2108.07732*, 2021. Available: https://arxiv.org/abs/2108.07732

6. V. Majdinasab et al., "Assessing the Security of GitHub Copilot Generated Code—A Targeted Replication Study," *arXiv preprint arXiv:2311.11177*, 2023. Available: https://arxiv.org/abs/2311.11177