Automated Code Generation Using Artificial Intelligence: Exploring Tools like GitHub Copilot and ChatGPT to Assist Programmers

Arisha Rehan Chotani, Abeer Aslam, Dania Zehra, Fatimah Ansari,
Vaniya Rehan, Taqwa Rasheed, Hira Iftikhar
Department of Computer Science,
National University of Emerging Sciences and Computing, Karachi, Pakistan

Research Supervisor: Miss Nazia Imam
Department of Sciences and Humanities,
National University of Emerging Sciences and Computing, Karachi, Pakistan
Email: nazia.imam@nu.edu.pk

ABSTRACT: This study investigates the impact of AI code generation tools such as ChatGPT and GitHub Copilot on programming practice and education. Through literature review, surveys, and interviews, we assess their reliability, user preferences, and influence on learning. Although these tools improve productivity and help with syntax and logic, they also produce frequent errors and may hinder deep understanding. Our findings underscore the need for critical evaluation and responsible integration of AI coding assistants in educational environments.

KEYWORDS: LLM, AI code generation, programming education, GitHub Copilot, ChatGPT, DeepSeek

OBJECTIVE: As digital tools have redefined language instruction, AI tools are poised to transform programming education and practice. This study provides insight into the effectiveness, user preferences, and limitations of current AI-based programming tools. Its findings will help educators refine curricula, help developers make informed choices, and inform future AI development that supports real-world coding needs.

I. INTRODUCTION

The use of artificial intelligence, particularly large language models (LLMs) such as ChatGPT and GitHub Copilot, has revolutionized modern programming. These tools are now integrated into code editors, helping developers and learners generate, debug and understand code. Similar to the growing importance of digital tools in language instruction, AI tools in programming hold immense pedagogical and practical value. In recent years, these technologies have become essential not only for professionals but also in academic settings, where programming is taught. Although widely celebrated for improving productivity, these tools are not without criticism. Questions about the correctness, reliability, and impact on human cognitive development have prompted researchers to investigate how programmers interact with such systems, how reliable the generated code is, and whether students learn effectively when using AI as a tutor.

A. Literature Review

The rapid advancement of large language models (LLMs) such as ChatGPT and GitHub Copilot has led to their integration into modern software development workflows. These tools promise enhanced productivity, improved code understanding, and even support for novice programmers. However, their actual effectiveness, trustworthiness, and educational implications remain subjects of active research. This literature review explores the current research landscape on AI-based code generation, focusing on the tools' reliability, educational influence, human-AI collaboration, and cognitive impact.

Before examining how AI tools assist programmers or are received by learners, it is essential to evaluate the reliability of their outputs. Multiple studies have investigated the correctness, error patterns, and hallucination behaviors of LLMs like ChatGPT. While models often produce syntactically valid code, research shows that their functional correctness varies significantly, with non-syntactic errors and hallucinations being common. For instance, Liu et al. introduced *EvalPlus*, revealing that conventional benchmarks like HUMANEVAL often overestimate code quality, with more rigorous testing exposing up to a 28.9% drop in correctness [8]. Similarly, recent work on *HalluCode* proposed a taxonomy of hallucination types and found that LLMs frequently generate outputs that are plausible but logically flawed [3]. Chen et al. further categorized common LLM mistakes, such as misunderstood specifications or positional sensitivity, and showed that even GPT-4 struggles to detect the root causes of its own

errors [7]. Complementing these findings, an empirical comparison of Copilot, ChatGPT, and CodeWhisperer demonstrated that although tools achieve high syntactic validity, functional reliability varies widely, with ChatGPT leading in correctness but still producing code that often requires human review [4].
Building on this technical baseline, researchers have explored how these tools are reshaping programming education, particularly in CS1 and CS2 environments. Several studies highlight how LLMs help students improve computational thinking, boost motivation, and provide on-demand support for code generation and debugging. Yilmaz and Karaoglan, for instance, found that students using ChatGPT in a Java course demonstrated significantly higher scores in computational thinking, programming self-efficacy, and motivation compared to those who did not [10]. Similarly, Becker et al. described AI tools as both disruptive and transformative, urging educators to adapt their pedagogical approaches to focus more on code evaluation and ethical tool use rather than traditional code writing [5]. These insights are relevant to understanding the tools' perceived utility and learning effectiveness, indirectly addressing RQ1 (preferences in practical settings) and RQ4 (human vs. LLM-generated output quality). Additional research confirms that students generally view AI assistants like ChatGPT and GitHub Copilot as helpful for grasping programming logic and syntax, though concerns remain about shallow understanding and over-reliance [9],[2].

Outside academic settings, developers increasingly engage with LLMs in real-world programming tasks. Studies in this group examine how tools like Copilot affect professional development workflows, collaboration, and trust. This perspective contributes to understanding how integrated tools are received in practical environments—an angle central to RQ1. It also emphasizes that the success of LLMs isn't purely based on accuracy but on how well they integrate into human workflows and support cognitive offloading without eroding developer autonomy. For instance, the ACM Queue study *Taking Flight with Copilot* found that while Copilot transforms the development experience by enhancing efficiency and idea generation, its limited context awareness often hinders accuracy, pointing to a need for better integration and transparency [6]. Similarly, a 2024 study on AI-powered development tools highlighted how Copilot aids productivity but requires cautious use to avoid over-reliance and blind trust in automated suggestions [1].

These insights highlight that despite their promise, LLMs must be used critically, especially in educational or production environments where correctness is paramount.Importantly, while these tools offer significant pedagogical benefits, their responsible integration into curricula remains essential to preserve foundational learning and critical thinking. Together, these findings suggest that human-AI collaboration in coding is less about automation and more about designing systems that align with human decision-making, trust, and workflow practices. Addressing these tensions, this study seeks to explore user preferences (RQ1), investigate the prevalence and types of mistakes in LLM outputs (RQ2, RQ3), and assess how LLM-generated code compares to that of humans (RQ4). By grounding our research in both technical evaluations and user-centered insights, we aim to contribute to a more nuanced understanding of AI-assisted programming.

B. *Problem Statement*

Large language models (LLMs) like ChatGPT and GitHub Copilot are increasingly used in software development and programming education, improving efficiency and learning. However, concerns persist about their reliability, accuracy, and impact on education. These tools often generate code with non-syntactic logic errors, which can affect both novice programmers and professionals who rely on them for correctness. There is limited understanding of how frequently these tools produce errors, how well they detect their own mistakes, and whether their outputs outperform human-written code. This presents challenges in their responsible use, especially in contexts where incorrect code has significant consequences. A timely investigation is needed to assess user preferences between integrated tools (e.g., Copilot) and standalone assistants (e.g., ChatGPT), evaluate the functional accuracy of generated code, and understand their impact on programming habits and education. The study will provide insights to guide best practices for LLM deployment, inform computer science curricula, and help developers optimize AI use in coding. The expected outcomes of this study include a clearer understanding of the effectiveness, limitations, and comparative performance of LLM tools, as well as actionable insights for educators, tool developers, and end users aiming to optimize the use of AI in software development. Therefore, this study aims to fill a research gap by systematically evaluating these tools through empirical and user-centered methods.

C. *Research Questions*

1. Are tools integrated with coding environments (e.g., Copilot) preferred over standalone tools (e.g., ChatGPT)?
2. How commonly do LLMs output incorrect code?
3 How capable are LLM tools in recognizing non-syntactic mistakes in code int terms of accuracy?
4. Is LLM-generated code better than human-generated code in terms of corrected-ness?

II. METHODOLOGY

A. *Research Approach*

A mixed-method approach was adopted for this study to provide a more comprehensive understanding of the effectiveness and perception of LLM-based code generation tools. Both quantitative (survey-based) and qualitative (semi-structured interviews and experimental evaluations) strategies were employed. This approach ensured triangulation of findings and allowed the study to explore both user preferences and code evaluation accuracy in depth.

B. Data Collection

1) Sampling Technique

The target population for this study consisted of undergraduate Computer Science students and junior software developers with recent exposure to tools such as GitHub Copilot and ChatGPT. From this population, a convenience sampling technique was used to select participants from local universities and tech forums. A total of 49 survey respondents participated, and 5 interviewees were selected for follow-up qualitative inquiry. The participants represent a demographic typical of early-stage programmers, many of whom rely on LLMs to support daily coding activities.

2) Research Instruments

Two distinct instruments were used:

1. Survey Questionnaire:

A structured questionnaire containing closed-ended and open-ended items was administered. The close-ended section collected data on tool usage frequency, preferences, and perceived correctness. The open-ended section captured more nuanced reflections on tool performance and limitations.

2. Semi-Structured Interview Protocol:

Designed to elicit deeper insights related to RQ3 and RQ4, the interview protocol included 10 open-ended questions, focusing on the perceived reliability of LLM tools in recognizing code errors and the comparison between AI-generated and human-generated code. Interviewees were also asked to review real-world code samples and offer comparative judgments.

C. Data Analysis

The survey data was analyzed using descriptive statistics to identify patterns in user preferences, error frequencies, and perceptions of code quality. Key metrics included: Tool preference distribution (integrated vs. standalone) Frequency of encountered errors in LLM-generated code Types of errors most commonly observed Perceived reliability in detecting non-syntactic mistakes Comparative analysis of AI-generated versus human-written code For the qualitative data from interviews, thematic analysis was employed. Codes were clustered under broad themes such as ”Error Recognition,” ”Tool Trustworthiness,” and ”Efficiency Perception.” Cross-tabulations were performed to identify relationships between user demographics, tool preferences, and perceived reliability.

D. Ethical Considerations

Prior to data collection, informed consent was obtained from all participants. Identical information was not collected during interviews or surveys. All audio data from interviews were transcribed and anonymized prior to analysis for the research paper. The study maintained confidentiality by not collecting or storing personally identifiable information beyond basic demographics needed for analysis.

III. Findings

A. Quantitative Findings (Survey Analysis)

A survey of 49 student programmers provided comprehensive insights into the adoption, usage patterns, and perceptions of Large Language Models (LLMs) in coding contexts. The findings reveal both the growing integration of these tools into programming workflows and the persistent challenges they present.

1) Familiarity and Usage

The overwhelming majority of respondents (over 90%) reported familiarity with LLMs, with 85% actively using these tools for programming tasks(Figure1). ChatGPT dominated as the preferred tool among 84% of respondents, while DeepSeek and GitHub Copilot were used by significantly smaller proportions (10.2% and 6.1%, respectively). This pronounced preference for ChatGPT likely stems from its early market presence and widespread name recognition despite the availability of newer alternatives specifically designed for code generation. The distribution suggests that familiarity and accessibility may currently outweigh specialized functionality in driving tool selection among student programmers.

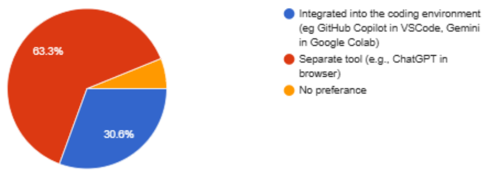

Fig. 1: Prefrence of coding tools

2) Tool Preferences and Integration

When examining preferred modes of interaction with LLMs, 63% of respondents favored using separate tools (such as browser-based ChatGPT), while 30% preferred integrated solutions like GitHub Copilot embedded within their development environment. The remaining 7% expressed no strong preference between these approaches. This pattern suggests that while standalone interfaces currently predominate, a significant minority already recognizes the workflow advantages of integrated solutions, potentially indicating a future shift as IDE-integrated tools mature and become more accessible to student developers.

3) *Usage Patterns and Frequency*

The frequency with which respondents incorporate LLMs into their coding processes demonstrates substantial integration into their development workflows (Refer to Figure1 and 2). Among participants, 14.3% reported using LLMs ”very frequently” during coding sessions, 28.6% used them ”frequently,” and 49% turned to these tools ”sometimes.” Notably, no respondents indicated they ”never” use LLMs when programming. The combined 42.9% of students who regularly (”frequently” or ”very frequently”) employ AI coding assistance represents a significant shift in how programming skills are being developed and applied in educational contexts, suggesting these tools are becoming normalized components of student programming experience.

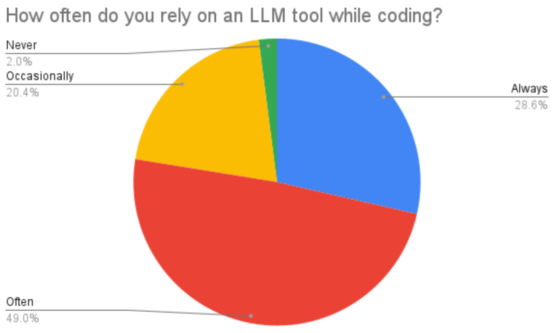

Fig. 2: Frequency of LLM Usage While Coding

4) *Error Frequency and Types*

Despite their widespread adoption, LLM-generated code continues to present reliability challenges. Among respondents, 14% reported encountering errors in AI-generated code ”very frequently,” 31% ”frequently,” 51% ”sometimes,” and only 4% ”rarely.” The persistence of errors across different usage frequencies indicates that even experienced users continue to face quality issues with generated code. The most prevalent error types identified were misunderstandings of the prompt (65.3%), logic/semantic errors (53.1%), incorrect output (40.8%), incomplete code (32.7%), and code that was difficult to understand or maintain (30.6%). This distribution suggests that while current LLMs have largely overcome basic syntax issues, they continue to struggle with higher-level conceptual understanding and contextual reasoning required for reliable code generation. The predominance of prompt misunderstandings points to communication challenges at the human-AI interface as a primary limitation in current implementations.

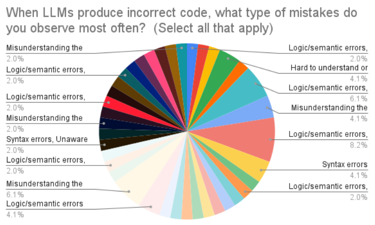

Fig. 3: Common Mistakes in LLM-Generated Code

5) *Effectiveness in Code Review and Error Detection*

Despite generating errors themselves, LLMs were paradoxically valued as code review tools, with approximately 75% of respondents rating them as handling non-syntactic mistakes ”well” or ”very well.” When specifically evaluating LLMs for their ability to identify logic errors in user-written code, 51% found them ”very helpful,” 33% ”somewhat helpful,” 10% were ”neutral,” and only 6% considered them ”not helpful.” This apparent contradiction - tools that produce errors yet effectively detect them - suggests that LLMs may excel at pattern recognition and best-practice identification even when they struggle with original code generation, potentially positioning them as more valuable in review capacities than in initial development for certain use cases.

6) *Perceptions of Code Quality*

Respondents’ evaluations of AI-generated code quality relative to human-written code revealed generally positive but nuanced perceptions. More than two-thirds viewed AI code favorably, with 28.6% rating it as ”much better” and 40.8% as ”slightly better” than typical human-written code. Another 18.4% considered AI and human code roughly equivalent, while only 8.2% perceived AI-generated code as ”slightly worse” than human-written alternatives. Notably, no respondents rated AI code as ”much worse,” indicating that even critics of AI-generated code acknowledge its baseline competence. These generally positive perceptions may help explain the rapid adoption of these tools despite their known limitations, as students perceive value in the output quality despite encountering errors.

B. *Qualitative Findings (Interview Analysis)*

The interviews offered a deeper understanding of user challenges and preferences: A sophomore Electrical Engineering student (A) highlighted that while LLMs work well for Python, their accuracy drops significantly when applied to niche languages like Matlab. A junior Software

Engineering student (B) found Grok useful for .NET-related queries but criticized ChatGPT for frequently generating incorrect code in this context. Both interviewees noted that LLM-generated code tends to use long variable names, making it easier to distinguish from human-written code. There was consensus that LLMs are adept at catching logic and semantic issues, but not always effective with syntax-level debugging in lesser-used languages. Student A preferred ChatGPT over DeepSeek for code explanation, due to its clarity and depth. One particularly insightful comment from a graduate student interviewee was: ”I can tell AI-generated code by the overly descriptive variable names and extensive commenting. Human code tends to be more concise but sometimes harder to follow. The biggest issue with AI code is that it looks correct but often has subtle logical flaws that only appear during specific edge cases.” These qualitative insights reinforced the quantitative data—while LLMs are widely used and appreciated, users remain cautious of their limitations, particularly in less common programming environments.

C. Thematic Analysis of Open Interviews

Two students from engineering backgrounds were interviewed. Person A is a sophomore electrical engineering student who primarily programs in MATLAB. Person B is a junior-year software engineering student and mostly uses LLM tools for web development projects.

Both participants noted the disparity in the accuracy of the code generated by LLMs. Person A said that in code generated in MATLAB, which is a niche programming language, the quality of code in terms of correctness was often compromised. Similarly, Person B noted that ChatGPT often makes mistakes when generating .NET code. This shows low generalization in LLMs.

Furthermore, both students expressed that they were left dissatisfied by chatGPT’s ability in debugging. Student A noted that ChatGPT is good at simplifying concepts as compared to tools like DeepSeek. Student B noted that in its pursuit of simplification, ChatGPT is often incorrect in the code it generates.

Additionally, 2 more interviews were carried out with undergraduate students who have high levels of exposure to AI-assisted programming. These responses indicated similar trends and also brought few new ideas, in terms of comparison of tools, satisfaction, and improvements they wish to see.

Both users indicated employing tools such as ChatGPT and DeepSeek for various purposes such as code generation, debugging, and brainstorming on projects. Simple tasks were coped with easily, but both were dissatisfied by about 20–30%—quite often due to ambiguous or outdated answers, particularly for domain-specific or complex logic issues. One interviewee mentioned DeepSeek giving faster and better answers than ChatGPT, which was perceived as sometimes too wordy and ambiguous.

Both users agreed to common hallucinations in LLM output, e.g., illogical inconsistencies or made-up APIs, citations, affirming the necessity of human intervention. Debugging assistance was found useful for surface problems but inadequate for more complex bugs out of context. Early enthusiasm with the tools was replaced by a more conservative, evaluative approach as constraints were realized. Users suggested enhancements such as improved context preservation, IDE integration, and more transparent explanations behind code proposals to improve reliability and trust in users.

D. Challenges and Limitations

1) Over-Reliance on LLMs

Many users, especially beginners, tend to trust the output of LLMs without verifying the logic, which could result in the propagation of incorrect or inefficient code. As one survey respondent noted, ”I sometimes find myself accepting Copilot suggestions without thorough review, which has led to bugs that were difficult to trace later.” This highlights a critical concern in educational settings where learning fundamental concepts might be compromised by over-dependence on AI assistance.

2) Domain-Specific Performance

The effectiveness of LLMs varies significantly across programming languages. Tools perform well with popular languages like Python and JavaScript but often falter with niche languages like Matlab or domain-specific code. Our interview data revealed that when working with specialized frameworks or less common libraries, users reported spending considerably more time fixing incorrect code generated by LLMs.

3) Survey Scope Limitations

The majority of survey respondents were from Computer Science or Engineering backgrounds, which may limit the generalizability of the findings across broader programming populations. Additionally, most participants were students or early-career professionals, potentially underrepresenting the perspectives of experienced developers.

4) Bias in User Experience

Familiarity with tools like ChatGPT might have influenced their higher ratings, as newer or lesser-known tools like DeepSeek received limited attention and feedback. This ”familiarity bias” could impact the comparative assessments between different LLM tools.

E. Ethical Concerns

1) Academic Integrity

The use of LLMs in academic settings raises concerns about plagiarism and unfair assistance. Students may submit LLM-generated code as their own, blurring the lines between learning and automation. One instructor interviewed noted: ”It’s increasingly difficult to assess individual understanding when assignments can be partially or fully completed by AI tools.”

2) Data Privacy and Licensing

There is a lack of transparency regarding the datasets LLMs are trained on. Some tools might generate code that is inadvertently copied from copyrighted sources, potentially leading to intellectual property concerns. This is particularly problematic when using these tools for commercial or production applications.

3) Bias in Training Data

LLMs can reflect biases present in their training data, potentially leading to inaccurate or culturally insensitive code/documentation. This can perpetuate existing inequities in technology and exclude certain perspectives or approaches.

4) Skill Degradation

Heavy dependence on AI tools might hinder the development of critical problem-solving and debugging skills in early-stage programmers. As one survey respondent (a teaching assistant) commented: ”I’ve noticed students becoming less adept at troubleshooting when they’ve relied heavily on AI tools from the beginning. They struggle when asked to work independently without these aids.”

IV. Conclusion

This study investigated the impact of AI-powered code generation tools on programming practices and education. Our findings confirm that tools like ChatGPT and GitHub Copilot are now deeply integrated into the workflows of student programmers and junior developers, with nearly 75% of survey respondents using them regularly. Despite the introduction of newer models, ChatGPT remains the preferred choice for most users (73%), likely due to its early market presence and user familiarity. Regarding our research questions, we found that:

1. Tool Integration Preference: While 57% of users currently prefer standalone tools like ChatGPT in a browser, a strong majority (67%) consider seamless integration with coding environments to be important. This suggests a potential shift toward integrated solutions as they become more sophisticated and accessible.

2. Error Frequency: Almost all users (96%) encounter errors in LLM-generated code at least sometimes, with 45% reporting frequent or very frequent errors. This underscores the continued need for human oversight and verification.

3. Non-syntactic Error Recognition: LLMs demonstrate good ability in recognizing non-syntactic mistakes, with 75% of respondents rating their performance as good or very good. However, qualitative data revealed limitations in specialized programming contexts.

4. Code Quality Comparison: Opinion is divided on whether LLM-generated code surpasses human-written code, with 49% believing AI code is better, 24% seeing them as comparable, and 27% favoring human code. This variation likely reflects differences in task complexity, programming domain, and user expertise.

The prevalence of certain error types—particularly misunderstanding prompts (67%), logic/semantic errors (49%), and incorrect outputs (47%)—highlights specific areas where these tools need improvement. These findings emphasize that while LLMs are valuable programming assistants, they are not yet reliable enough to replace human judgment, especially for complex or specialized programming tasks. Most importantly, our research identifies a tension in educational contexts: LLMs can enhance learning by providing immediate feedback, explaining concepts, and streamlining repetitive tasks, but they may also enable surface-level understanding and dependency that undermines deeper learning outcomes.

A. Recommendations for further work

Based on our findings, we offer the following recommendations:

For Educators:

Develop clear guidelines for appropriate LLM use in programming courses Design assignments that test conceptual understanding rather than just code production Incorporate critical evaluation of AI-generated code into the curriculum Use pair programming approaches where students alternately write code and review AI suggestions.

For Tool Developers:

Improve context understanding to reduce prompt misinterpretation errors Enhance transparency about the limitations of generated code Develop better error explanation capabilities, particularly for logical flaws

Create specialized models for less common programming languages and frameworks

For Students and Junior Developers:

Validate LLM-generated code through testing rather than trusting it implicitly Use AI tools as learning aids by requesting explanations of suggested code Practice solving problems independently before consulting AI assistants Develop the habit of critically reviewing all AI suggestions.

For Future Research:

Conduct longitudinal studies on how LLM use affects programming skill development Investigate how different prompting strategies impact code quality Explore ways to combine human and AI strengths in programming education Examine the impact of LLM use on programming assessment and evaluation As these technologies continue to evolve rapidly, ongoing assessment of their capabilities, limitations, and educational impact will be essential. The integration of AI into programming practice represents both an opportunity and a challenge for computer science education, requiring thoughtful approaches that maximize benefits while mitigating potential drawbacks.

V. Acknowledgements

The authors wish to thank all survey respondents and interviewees who contributed their time and insights to this research. Special appreciation goes to the Department of Computer Science at the National University of Emerging Sciences and Computing for supporting this project, and to Ms. Nazia Imam for her guidance throughout the research process.

References

[1] Ali, A. J. M. (2024). Automating the code: Exploring ai-powered development tool github copilot. *International Journal of Latest Research in Programming*.

[2] Anonymous (2023a). Chatgpt in programming education: Chatgpt as a programming assistant – an exploratory analysis of chatgpt's use cases and limitations in teaching programming concepts. *arXiv preprint arXiv:2304.56789*.

[3] Anonymous (2023b). Exploring and evaluating hallucinations in llm-powered code generation: Hallucode: A benchmark and taxonomy for hallucination detection in code. *arXiv preprint arXiv:2303.23456*.

[4] Anonymous (2024). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *arXiv preprint arXiv:2402.34567*.

[5] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. (2023). Programming is hard – or at least it used to be: Educational opportunities and challenges of ai code generation. *SIGCSE '23 Proceedings*.

[6] Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., and Gazit, I. (2023). Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools. *ACM Queue*, 20(6):10–32.

[7] Chen, X. et al. (2024). A deep dive into large language model code generation mistakes: What and why? analysis of non-syntactic errors in gpt-4 generated code. *arXiv preprint arXiv:2401.11234*.

[8] Liu, J., Xia, C. S., Wang, Y., and Zhang, L. (2023). Is your code generated by chatgpt really correct? evalplus: Rigorous testing of llm-generated code. *arXiv preprint arXiv:2302.12345*.

[9] Nizamudeen, F., Gatti, L., Bouali, N., and Ahmed, F. (2023). Investigating the impact of code generation tools (chatgpt & github copilot) on programming education. *Education and Information Technologies*.

[10] Yilmaz, R. and Karaoglan, F. G. (2023). The effect of generative ai tool use on students' computational thinking skills, programming self-efficacy, and motivation. *Journal of Educational Technology & Society*.