International Journal of Multidiscipline

TLEP – International Journal of Multidiscipline
(Technology, Language, Education, and Psychology)
ISSN: 2488-9342 (Print) | 2488-9334 (Online)

Open Access | Peer-Reviewed | Monthly Publication | Impact factor: 8.497 / 2025

Generative AI Based Learning In Computer Scince For Beginners

Saidova Xosiyatxon
Teacher at the Kokand Digital Technologies College

Abstract

The integration of Generative Artificial Intelligence (GenAI), specifically Large Language Models (LLMs) such as ChatGPT and GitHub Copilot, into Computer Science (CS) education presents a paradigm shift in how introductory programming is taught and learned. While these tools offer unprecedented support for syntax correction and code generation, concerns persist regarding their long-term impact on the cognitive development of beginners. This study investigates the effects of GenAI-assisted learning on novice programmers' performance, self-efficacy, and conceptual understanding. Adopting a mixed-methods approach, sixty first-year CS students were divided into an experimental group (AI-assisted) and a control group (traditional resources) to complete a series of Python programming tasks. Quantitative analysis revealed that while the AI-assisted group demonstrated significantly faster completion times and higher code correctness during the intervention, they scored lower on a subsequent non-assisted conceptual assessment compared to the control group. Qualitative data from post-experiment surveys indicated that while GenAI reduced anxiety related to syntax errors, it inadvertently fostered an "illusion of competence," where students believed they understood logic that they had merely generated.

Keywords: Generative AI, Computer Science Education, Novice Programmers, Self-Efficacy, Large Language Models, Computational Thinking, ChatGPT.

Introduction

The landscape of Computer Science (CS) education is currently undergoing its most significant transformation since the advent of the integrated development environment (IDE). For decades, the pedagogical strategy for novice programmers—often referred to as CS1 students—has focused on the arduous process of mastering syntax while simultaneously developing algorithmic thinking skills. This dual burden creates a high barrier to entry, often resulting in substantial attrition rates in introductory computing courses. Novices frequently find themselves demotivated by cryptic compiler error messages and the rigid specificity required by programming languages, which can obscure the underlying logic they are attempting to learn. Historically, educators have relied on human tutors, peer programming, and static online resources like Stack Overflow to mitigate these challenges. However, the public release of Generative Artificial Intelligence (GenAI) tools, particularly Large Language Models (LLMs) like OpenAI's ChatGPT and GitHub's Copilot, has introduced an on-demand, infinite source of coding assistance that is fundamentally changing the learning dynamic.

The promise of GenAI in education is rooted in the theory of personalized scaffolding. In an ideal scenario, an LLM acts as a "knowledgeable other" in the Vygotskian sense, guiding the student through the Zone of Proximal Development (ZPD) by explaining errors, suggesting optimizations, and breaking down complex problems. For a beginner paralyzed by a syntax error, an AI agent can provide immediate remediation, maintaining the student's flow state and engagement. This capability theoretically allows the curriculum to shift focus from the memorization of syntactic rules to higher-order system design and logic. If the friction of coding mechanics is removed, students should theoretically be able to progress faster and build more complex software than was previously possible in a single semester.

However, the integration of these tools is not without peril. A growing body of skepticism within the academic community centers on the distinction between *productivity* and *learning*. While professional developers use GenAI to automate routine tasks and increase efficiency, the goal of a novice is not to produce code, but to produce mental models of how code works. If an AI tool generates a solution that the student utilizes but does not comprehend, the educational objective is subverted. This phenomenon, increasingly described as the "illusion of competence," occurs when students mistake the ability to prompt an AI for the ability to program. Furthermore, the reliance on AI for debugging may atrophy the very problem-solving resilience—often termed "productive struggle"—that is essential for becoming a competent software engineer.

Despite the ubiquity of these tools, empirical research specifically targeting the cognitive impact of GenAI on *beginners* remains in its infancy. Much of the existing discourse is anecdotal or focuses on the accuracy of the models themselves rather than the learning outcomes of the humans using them. This research paper seeks to address this gap by providing empirical data on how GenAI impacts the learning trajectory of novice programmers. Specifically, this study aims to answer two primary research questions: (1) Does access to GenAI tools improve immediate problem-solving performance at the cost of deep conceptual retention? and (2) How does the availability of these tools influence the self-efficacy and anxiety levels of students encountering programming challenges for the first time? By dissecting these dynamics, educators can better understand how to recalibrate curricula for an AI-augmented future.

Literature Review

The theoretical framework for this study draws upon Constructivist learning theories and Cognitive Load Theory (CLT). Constructivism posits that knowledge is actively constructed by the learner, not passively received. In the context of computer science, this construction happens through the iterative process of writing, failing, debugging, and refining code. Ben-Ari (2001) argued that the distinctive nature of computer science, where the "reality" is a virtual machine totally under the user's control, requires explicit construction of mental models. Traditional pedagogy has utilized "scaffolding"—temporary supports that are gradually removed—to aid this process. Before GenAI, scaffolding took the form of skeleton code, block-based languages like Scratch, or Intelligent Tutoring Systems (ITS).

Intelligent Tutoring Systems represent the pre-generative ancestry of current AI tools. Research by VanLehn (2011) demonstrated that ITS could approach the effectiveness of human tutors by providing step-by-step feedback. However, these systems were rigid, rule-based, and limited to specific problem domains. In contrast, modern LLMs are probabilistic and generative, capable of handling open-ended queries across any programming language. This flexibility aligns with the goals of reducing "extraneous cognitive load"—the mental effort wasted on non-essential processing, such as wrestling with obscure syntax—thereby freeing up working memory for "germane cognitive load," or the construction of schemas (Sweller, 1988). Theoretically, if ChatGPT handles the syntax (extraneous load), the student can focus on the algorithm (germane load).

However, recent empirical studies suggest a more complex reality. Prather et al. (2023) conducted a seminal study on the "metacognitive difficulties" introduced by

GenAI. They observed that while students could solve problems faster, they often failed to recognize when the AI provided incorrect or suboptimal code, leading to a new type of error: the uncritical acceptance of generated text. Similarly, Denny et al. (2023) highlighted that while students found Copilot helpful, it occasionally disrupted their problem-solving process by offering solutions before the student had fully formulated the problem, essentially "short-circuiting" the cognitive effort required for learning.

Furthermore, the impact on self-efficacy—a student's belief in their capacity to execute behaviors necessary to produce specific performance attainments—is mixed. Yilmaz and Yilmaz (2023) found that GenAI tools could boost confidence by preventing students from getting "stuck" on minor errors, which is a primary cause of dropout in CS1 courses. Conversely, other researchers argue that this confidence is fragile; if the AI is removed, the student's self-efficacy may collapse, revealing a dependency rather than true mastery. This study seeks to isolate these variables by comparing AI-assisted performance against unassisted conceptual understanding, distinguishing between the *feeling* of learning and the *fact* of learning.

Methodology

To investigate the impact of GenAI on novice programmers, this study employed a quasi-experimental mixed-methods design, combining quantitative performance metrics with qualitative survey data.

Participants The participants were 60 undergraduate students enrolled in an introductory "CS1: Introduction to Programming with Python" course at a mid-sized university. All participants were screened to ensure they had no prior formal programming experience before the semester. The cohort was randomly assigned into two groups: the Experimental Group (n=30), which was allowed access to a custom interface powered by OpenAI's GPT-4 for assistance, and the Control Group (n=30), which was restricted to traditional resources (course textbook, official Python documentation, and lecture notes).

Procedure The experiment was conducted during a controlled laboratory session lasting 120 minutes. Both groups were tasked with solving three programming problems of increasing difficulty:

1. **Task A (Easy):** Write a function to calculate the factorial of a number using a loop.
2. **Task B (Medium):** Create a program to parse a string of text and count the frequency of each unique word, ignoring punctuation.
3. **Task C (Hard):** Implement a basic "Guess the Number" game that includes error handling for non-numeric inputs and limits the user to a maximum of 5 attempts.

The Experimental Group was given a brief tutorial on "prompt engineering"—how to effectively ask the AI for explanations or hints without simply asking for the final solution—though they were technically permitted to ask for code generation. The Control Group worked in a standard IDE environment.

Measures Data was collected across three dimensions:

1. **Performance Efficiency:** Measured by the time taken to submit a passing solution and the number of attempts (submissions) required.
2. **Conceptual Mastery (The "Whiteboard" Test):** Immediately following the laboratory session, both groups were administered a 20-minute pen-and-paper quiz. This quiz asked them to trace the execution of code similar to what they had just written and explain the logic of specific lines. No AI or computers were allowed. This was critical to assess whether the skills transferred to the student's internal knowledge base.

3. **Psychometric Survey:** A post-test questionnaire adapted from the Computer Programming Self-Efficacy Scale (CPSES) was administered to measure anxiety levels, perceived difficulty, and confidence in future tasks.

Data Analysis Quantitative data (time, test scores) were analyzed using independent samples t-tests to identify statistical significance. Survey responses were analyzed using Likert scales, while open-ended feedback was coded thematically to identify patterns in student sentiment regarding the "helpfulness" of the AI.

Results and Analysis

The data collected revealed a distinct dichotomy between immediate task performance and subsequent conceptual retention. The Experimental (AI) group significantly outperformed the Control group in the speed of task completion, but the Control group demonstrated superior retention of the underlying concepts during the unassisted post-test.

Quantitative Findings

As detailed in Table 1 below, the AI-assisted group completed the three programming tasks much faster than their counterparts. For the difficult task (Task C), the AI group finished in an average of 24 minutes, whereas the Control group averaged 41 minutes. Furthermore, the "Pass Rate on First Attempt" was drastically higher for the AI group, suggesting that the AI helped eliminate common syntax errors that typically cause initial submissions to fail.

However, the results of the "Conceptual Mastery Post-Test" told a different story. The Control group, having struggled through the syntax and logic manually, scored an average of 82% on the pen-and-paper exam. The AI group, despite their speed in the lab, averaged only 64%. This 18% gap is statistically significant ($p < 0.05$) and indicates that while the AI group produced better code, they internalized less of the logic.

Table 1: Performance Metrics and Conceptual Retention Scores

Metric	Experimental Group (AI-Assisted)	Control Group (Traditional)	p-value
Mean Time to Complete Task B (min)	14.5	28.2	< 0.01
Mean Time to Complete Task C (min)	24.0	41.0	< 0.01
Code Correctness (Task C)	93%	78%	0.04
Conceptual Post-Test Score (0-100)	64.5	82.1	< 0.01
Syntax Errors per Submission	0.8	4.2	< 0.01

Qualitative and Survey Findings

The survey data (Table 2) illuminated the psychological impact of the tools. The AI group reported significantly lower anxiety and higher self-efficacy *during* the task. When asked, "I felt capable of solving the problem," the AI group strongly agreed. However, when asked "I could solve a similar problem without help," the confidence gap narrowed. Qualitative comments from the AI group revealed a reliance pattern. One student noted, *"I knew what I wanted to do, but I didn't know how to write it. The AI just did the typing for me."* Another admitted, *"I wasn't sure why the loop needed a break statement, but the AI put it there and it worked, so I moved on."*

In contrast, the Control group expressed higher frustration but also higher satisfaction upon completion. A typical comment from this group was: *"It took me forever to figure out the dictionary syntax, but once I got it, I really understood how it stored the data."*

Table 2: Student Perception and Self-Efficacy Survey (Likert Scale 1-5)

Survey Item	Experimental Group (AI) Mean	Control Group (Traditional) Mean	Interpretation
"I felt anxious while coding."	1.8	3.9	AI significantly reduced anxiety.
"I understood every line of code I submitted."	3.2	4.6	Control group had deeper comprehension.
"I am confident I can fix a bug in this code."	4.1	3.5	AI group exhibited inflated confidence.
"The tool helped me learn, not just copy."	3.5	N/A	Mixed feelings on learning utility.

The discrepancy between the AI group's high confidence ("I can fix a bug") and their low post-test scores suggests an "illusion of competence." They conflated the ease of generating a solution with the possession of skill. The analysis confirms that GenAI acts as a potent anxiety reducer and productivity accelerator, but without pedagogical intervention, it risks becoming a crutch that bypasses the cognitive struggle necessary for deep learning.

Discussion

The findings of this study highlight a critical paradox in the modern Computer Science classroom: the tools that make programming "easier" may simultaneously make *learning* to program harder. The results align with the "metacognitive difficulties" described by Prather et al. (2023). The AI-assisted students in this study successfully bypassed the "extraneous load" of syntax errors, which is theoretically beneficial. However, they also inadvertently bypassed the "germane load" of algorithmic construction. By having the solution generated for them, they engaged in *reviewing* code rather than *creating* it. Bloom's Taxonomy places "creation" at the pinnacle of cognitive work; shifting students immediately to "evaluation" (reviewing AI code) before they can create may skip essential developmental steps.

The "Illusion of Competence" observed in the survey data is particularly concerning. Students in the AI group felt more confident, yet performed worse when the scaffold was removed. This suggests that the feedback loop in AI-assisted learning is distorted. In traditional coding, a syntax error is immediate, negative feedback indicating a lack of knowledge. With AI, the immediate feedback is a working solution, which the student interprets as success, even if the intellectual contribution was minimal. This aligns with recent concerns in educational psychology that LLMs might serve as a "prosthetic" for intelligence rather than a tool for its development.

However, it is crucial not to demonize the tool. The reduction in anxiety (Table 2) is a positive outcome that cannot be overlooked. High anxiety is a known barrier to entry in STEM fields, often disproportionately affecting underrepresented groups. If GenAI can lower the barrier to entry by removing the frustration of missing semicolons, it could theoretically increase retention rates. The challenge, therefore, is not to ban these tools, but to change how they are assessed. The significant drop in post-test scores for the AI group implies that our current assessment methods (submitting working code) are no longer valid proxies for student learning. If a machine can produce the output, the output itself proves nothing about the human's capability.

Consequently, CS curricula must pivot. We must move away from "write this function" assignments toward "explain this AI-generated code," "debug this AI hallucination," or "optimize this valid but inefficient solution." This shifts the learning objective to the evaluation and synthesis levels, where the human must possess a robust mental model to critique the machine.

Conclusion

This study provides empirical evidence that while Generative AI significantly enhances the speed and correctness of novice programming submissions, it poses a tangible risk to conceptual mastery and retention. The AI-assisted students exhibited a "competence-performance gap": their performance was high, but their independent competence was lower than that of the control group. The reduction in anxiety and the elimination of trivial syntax errors are valuable benefits, but they come at the cost of the "productive struggle" required to encode new schemas into long-term memory.

For educators, the implication is clear: the integration of GenAI into the classroom requires a restructuring of assessment and pedagogy. We cannot simply allow students to use these tools for standard assignments and assume learning is taking place. Future research should focus on longitudinal studies to see if this "illusion of competence" fades as students mature into intermediate programmers, and to develop "AI-resilient" assessments that measure logic and system design rather than syntax generation. Ultimately, the goal of CS education in the AI era must be to train pilots who can fly the plane when the autopilot inevitably fails.

References

Ben-Ari, M. (2001). Constructivism in computer science education. Journal of Computers in Mathematics and Science Teaching, 20(1), 45-73.

Denny, P., Becker, B. A., Craig, M., Reeves, B. N., & Prather, J. (2023). Converse, code, and correct: Using large language models in computing education. Proceedings of the 2023 ACM Conference on International Computing Education Research, 1–12. https://doi.org/10.1145/3568813.3600138

Deniz, N. A., & Yilmaz, R. (2023). The impact of generative AI on programming students: Frustration and confidence across learning styles. International Association for Computer Information Systems, 371-385.

OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com

Prather, J., Denny, P., Leinonen, J., & Becker, B. A. (2023). The widening gap: The benefits and harms of generative AI for novice programmers. ACM Transactions on Computing Education. https://doi.org/10.1145/3616377

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Science, 12(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educational Psychologist, 46(4), 197-221. https://doi.org/10.1080/00461520.2011.611369

Yilmaz, R., & Yilmaz, F. G. K. (2023). The effect of generative AI on the self-efficacy of novice programmers. Journal of Educational Computing Research, 61(5), 1123-1145.