OPEN ACCESS

Kashf Journal of Multidisciplinary Research
Vol: 03 - Issue 01 (2026)
P-ISSN: 3007-1992 E-ISSN: 3007-200X
https://kjmr.com.pk

MEASURING THE IMPACT OF AI CODING ASSISTANTS (E.G., GITHUB COPILOT, CHATGPT) ON PROGRAMMING PRODUCTIVITY AMONG BSCS STUDENTS

Dua Nadeem

Email: *nadeemdua786@gmail.com*

Nosheen Asif

Email: *nosheenasif536@gmail.com*

Mariam Tanveer

Email: *mariuum005@gmail.com*

Kinza Javed

Email: *kinzajaved934@gmail.com*

[1,2,3,4] BSCS scholar Air University Multan Campus, Islamabad

Article Info

Abstract

This study examines the impact of AI coding assistants on programming productivity among undergraduate computing students, with a focus on BSCS learners. The rapid adoption of tools such as GitHub Copilot and ChatGPT has transformed how students approach coding tasks, raising questions about whether these tools enhance productivity or alter learning behavior. Existing research largely emphasizes efficiency gains in controlled or professional settings by creating a gap in understanding their real-world academic impact on students. A quantitative, cross-sectional survey design was employed, collecting data from 127 participants through a structured questionnaire. The study measures productivity as a multi-dimensional construct, including task completion time, code quality, debugging efficiency, and perceived problem-solving independence. The findings suggest that AI coding assistants contribute to improved task efficiency and support learning processes, particularly in understanding programming concepts and completing coding tasks more effectively. Students reported benefits in terms of ease of coding and assistance during debugging, while also acknowledging concerns regarding the reliability of AI-generated outputs. Additionally, the use of these tools appears to influence students' approach to problem-solving and their level of independence. In conclusion, AI coding assistants offer notable benefits in supporting programming tasks and learning experiences, but they also introduce challenges related to over-reliance. Their overall impact depends on how they are used, highlighting the importance of guided and balanced integration within programming education to ensure that productivity gains do not come at the expense of independent skill development.

Keywords: *AI coding assistants, Programming productivity, BSCS students, Code quality, Debugging efficiency, Dependency risk, Student perception*

1. Introduction

Artificial intelligence (AI) has increasingly become part of everyday programming practice, moving beyond research labs into classrooms and development environments. Tools such as GitHub Copilot and ChatGPT are now widely used to generate code, suggest solutions, and assist with debugging in real time. For BSCS students in particular, these tools are beginning to influence how programming tasks are approached, shifting the focus from writing code manually to interacting with AI systems for guidance and automation. This shift has led to an important discussion in academics about whether AI coding assistants actually improve programming productivity or simply change how students approach and engage with coding tasks. Existing studies offer mixed perspectives on this issue. On one side, research shows that AI tools can significantly improve efficiency by reducing the time required to complete programming tasks and lowering cognitive effort (Peng et al., 2023; Nijkamp et al., 2022). On the other hand, several studies raise concerns about over-reliance, reduced problem-solving ability and superficial understanding of code (Becker et al., 2023; Barke et al., 2023). In educational contexts, AI tools are also seen as personalized learning aids that can support students during programming (Kasneci et al., 2023), yet their long-term impact on skill development remains uncertain. These contrasting viewpoints highlight the need for a more balanced and student-focused investigation into how AI coding assistants influence programming productivity beyond just speeds.

1.1 Background of the Study

The integration of AI into software development has introduced a new class of tools capable of assisting programmers at different stages of coding. AI coding assistants such as GitHub Copilot and ChatGPT use large language models to generate code snippets, recommend solutionsand even explain programming concepts. Their growing adoption among students reflects a broader shift in how programming is learned and practiced.Several recent studies have explored the impact of these tools. Peng et al. (2023) conducted a controlled experiment and found that developers using GitHub Copilot completed tasks significantly faster compared to those working without AI assistance. This improvement was largely attributed to reduced cognitive load and the ability to rely on relevant code suggestions. Similarly, Nijkamp et al. (2022) demonstrated that large language models can generate functional code across various tasks, suggesting that AI can automate repetitive aspects of programming and allow users to focus on higher-level logic. However, not all findings are entirely positive. Zhang et al. (2023) examined real-world usage of GitHub Copilot and reported that while users experienced increased speed and convenience, they also faced issues such as incorrect code suggestions and the need for constant verification. Becker et al. (2023) further argued that AI tools might reduce deep learning by encouraging students to accept generated solutions without fully understanding them. In a similar vein, Barke et al. (2023) observed that developers often treat AI tools as collaborators, which can improve efficiency but may also lead to uncritical acceptance of outputs. From an educational perspective, Kasneci et al. (2023) highlighted that AI tools like ChatGPT can provide personalized assistance by explaining concepts and guiding students through tasks. While this can enhance learning efficiency, it also raises questions about whether students are developing genuine problem-solving skills. Similarly, Prather et al. (2023) examined novice programmers' usability experiences with GitHub Copilot and reported that students found AI suggestions intuitive yet struggled to critically evaluate the correctness of AI-generated outputs, reinforcing the need for structured guidance in AI-assisted programming education. In another study,

Denny et al. (2023) found that introductory programming students improve performance in CS1 tasks with GitHub Copilot through iterative natural language prompting and refinement of problem descriptions, suggesting that prompt engineering plays a key role in how students engage with AI tools and develop computational thinking skills. A key limitation across these studies is that most focus on professional developers or controlled environments, leaving a gap in understanding how undergraduate students actually use these tools in real academic settings.

1.2 Statement of the Problem

Despite the growing popularity of AI coding assistants, their actual impact on programming productivity among BSCS students remains unclear. Much of the existing research measures productivity in terms of task completion time within controlled environments (Peng et al., 2023), which does not fully reflect real classroom or assignment-based settings. This creates a contextual gap, as findings from professional developers cannot be directly applied to students who are still developing their programming skills. There is also a conceptual gap in how productivity is defined. Most studies focus primarily on speed and efficiency, while ignoring other important dimensions such as code quality, debugging ability and independent problem-solving (Kasneci et al., 2023). At the same time, the literature shows a contradiction, as some studies report improved efficiency while others suggest that AI tools may weaken critical thinking and increase dependency (Becker et al., 2023). In addition, a methodological gapexists due to the limited availability of survey-based or real-world data that captures how students actually use these tools in their daily academic work (Zhang et al., 2023).To address these gaps, this study focuses specifically on BSCS students and evaluates programming productivity as a multi-dimensional concept. It examines not only task completion time but also code quality, debugging efficiency and students' perceived independence. By linking actual usage patterns with productivity outcomes, this research provides a more realistic and comprehensive understanding of how AI coding assistants influence student performance.

Figure 1 presents the conceptual framework developed for this study, showing how AI coding assistant usage is expected to influence programming productivity among BSCS students. In this framework, AI tool usage is considered the main input variable. It includes how often students use these tools, the purposes for which they use them (such as code generation, debugging, or understanding concepts) and the level of dependence on them during programming tasks. Instead of assuming the same effect for all students, we consider that the impact of AI tools can vary. To account for this, moderating factors such as programming skill level, prior experience, learning style, and familiarity with AI tools are included. These factors help explain why AI tools may benefit some students more than others while in some cases they may have limited or even negative effects Programming productivity is treated as a multi-dimensional outcome in this study. Rather than focusing only on task completion time, we also consider code quality, debugging efficiency and problem-solving ability. This approach provides a more complete view of productivity, especially in a learning environment where understanding and independence are important. Overall the framework shows that the effect of AI coding assistants depends on how they are used and the characteristics of the students using them. This allows us to study their impact in a more detailed and realistic way.

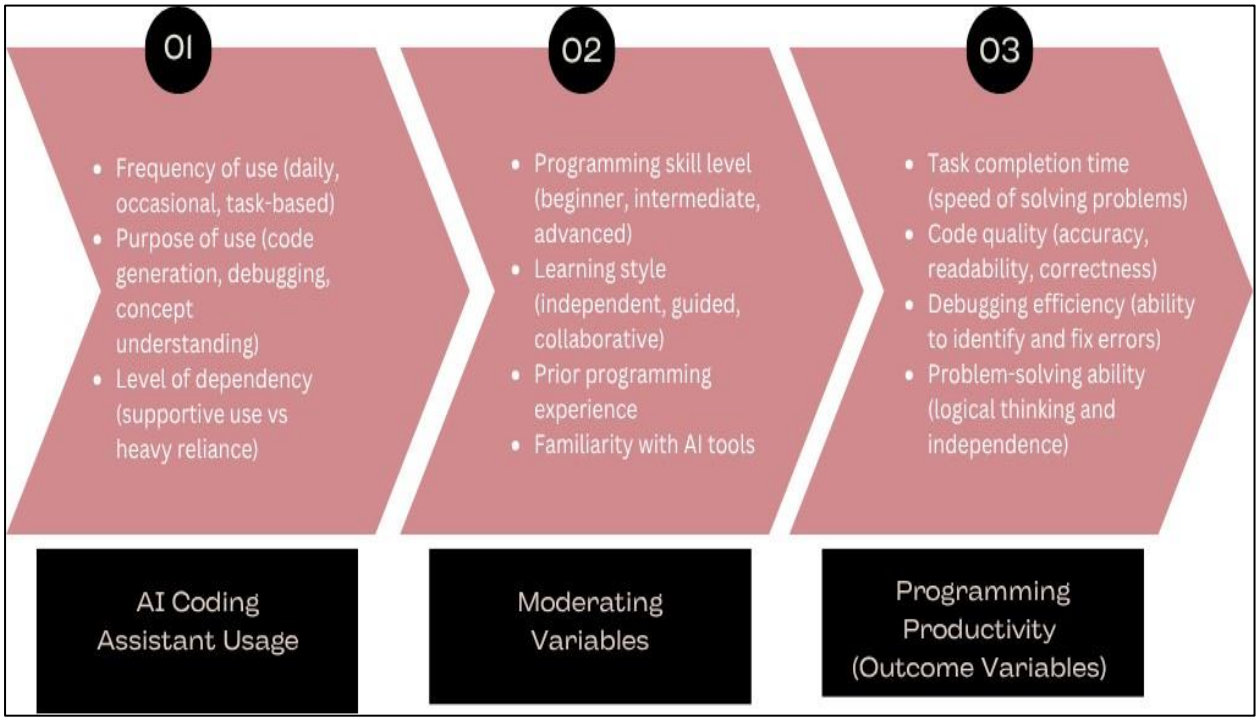


Figure 1 Conceptual Framework of Factors Affecting Programming Productivity with AI Coding Assistants

1.3 Research Questions

This study is guided by the following research questions:

1. How does using AI coding assistants affect programming productivity among BSCS students?
2. How often do BSCS students use AI coding assistants for programming-related tasks?
3. Is there a relationship between AI coding assistant usage frequency and programming productivity?
4. How does using AI coding assistants affect productivity factors like task completion time, code quality, and debugging efficiency?
5. Does using AI coding assistants affect students' perceived programming competence and independence?

1.4 Significance of the Study

This study contributes to a clearer understanding of how AI coding assistants function within programming education, particularly from a student perspective. Instead of treating AI tools as general learning aids, this research examines their specific role in shaping programming productivity. By considering multiple dimensions of productivity, the study provides a more balanced evaluation of both the benefits and potential drawbacks of these tools. For students, the findings offer practical insight into whether relying on AI tools actually improves their performance or simply provides short-term convenience. For educators, the study highlights how AI integration may influence learning outcomes, helping them design teaching strategies that encourage both

efficiency and independent thinking. From a research standpoint, this work addresses an important gap by combining real-world usage behavior with multi-dimensional productivity measures, offering a more comprehensive and realistic view of AI's role in programming education.

2. Literature Review

2.1 Theoretical Framework

This study draws on three theoretical frameworks to understand how BSCS students adopt, interact with, and are cognitively shaped by AI coding assistants such as GitHub Copilot and ChatGPT. These frameworks the Technology Acceptance Model (TAM), Constructivist Learning Theory, and Cognitive Load Theory. These collectively form the conceptual lens through which AI tool usage, programming productivity and the relationship between them are explored.

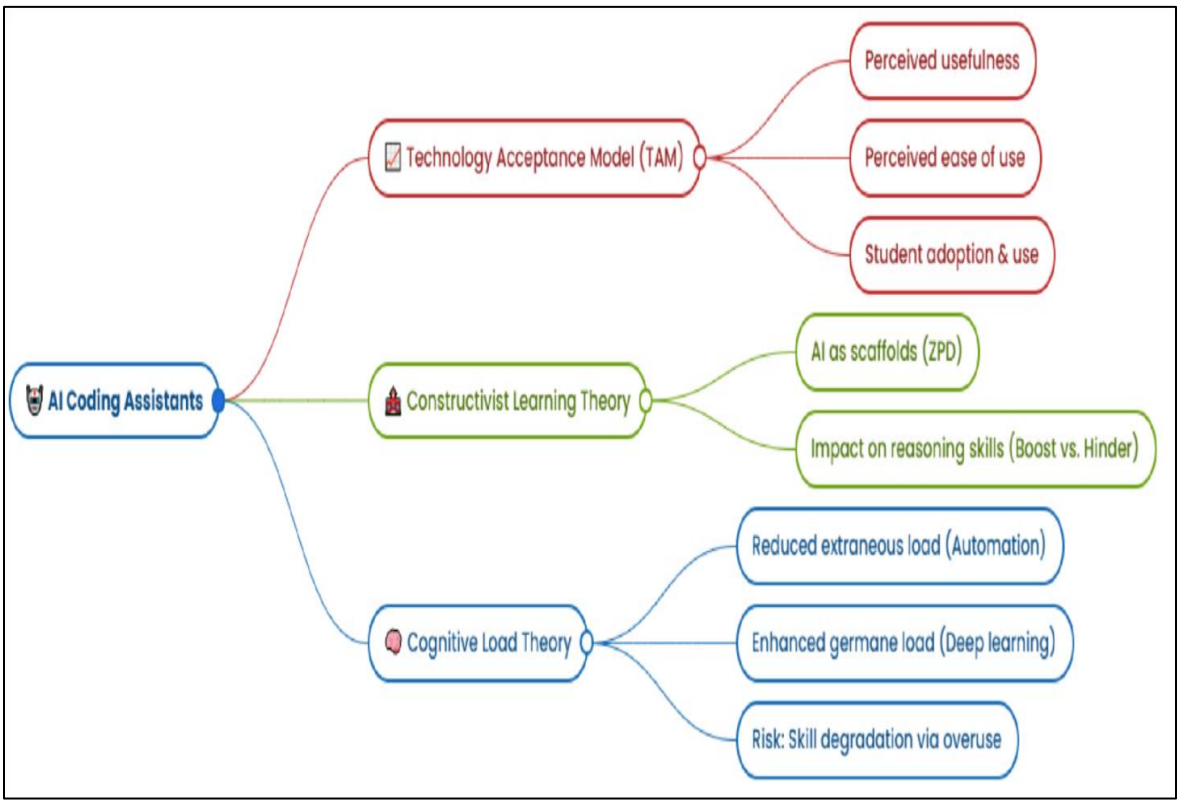


Figure 2 Theoretical Frameworks Supporting the Study

2.1.1 Technology Acceptance Model (TAM)

Davis's (1989) Technology Acceptance Model argues that people adopt new technologies primarily based on two perceptions: how useful they believe a tool to be, and how easy they find it to use. In the context of this study, TAM helps explain why some BSCS students readily incorporate tools like GitHub Copilot and ChatGPT into their coding routines while others do not. When students genuinely believe that these tools will make their work faster and less frustrating, their likelihood of consistent use goes up a pattern well-supported in the literature (Peng et al., 2023). Barke et al.

(2023) built on this, observing that students tend to use AI tools in one of two ways: either to speed through tasks they already understand or to feel their way through problems they have not encountered before. Both patterns, despite their differences, are rooted in the same underlying sense of perceived utility. For this reason, TAM provides the theoretical basis for treating AI tool usage as the study's primary independent variable not just as a measure of accessbut as a reflection of how students actually perceive and engage with these tools.

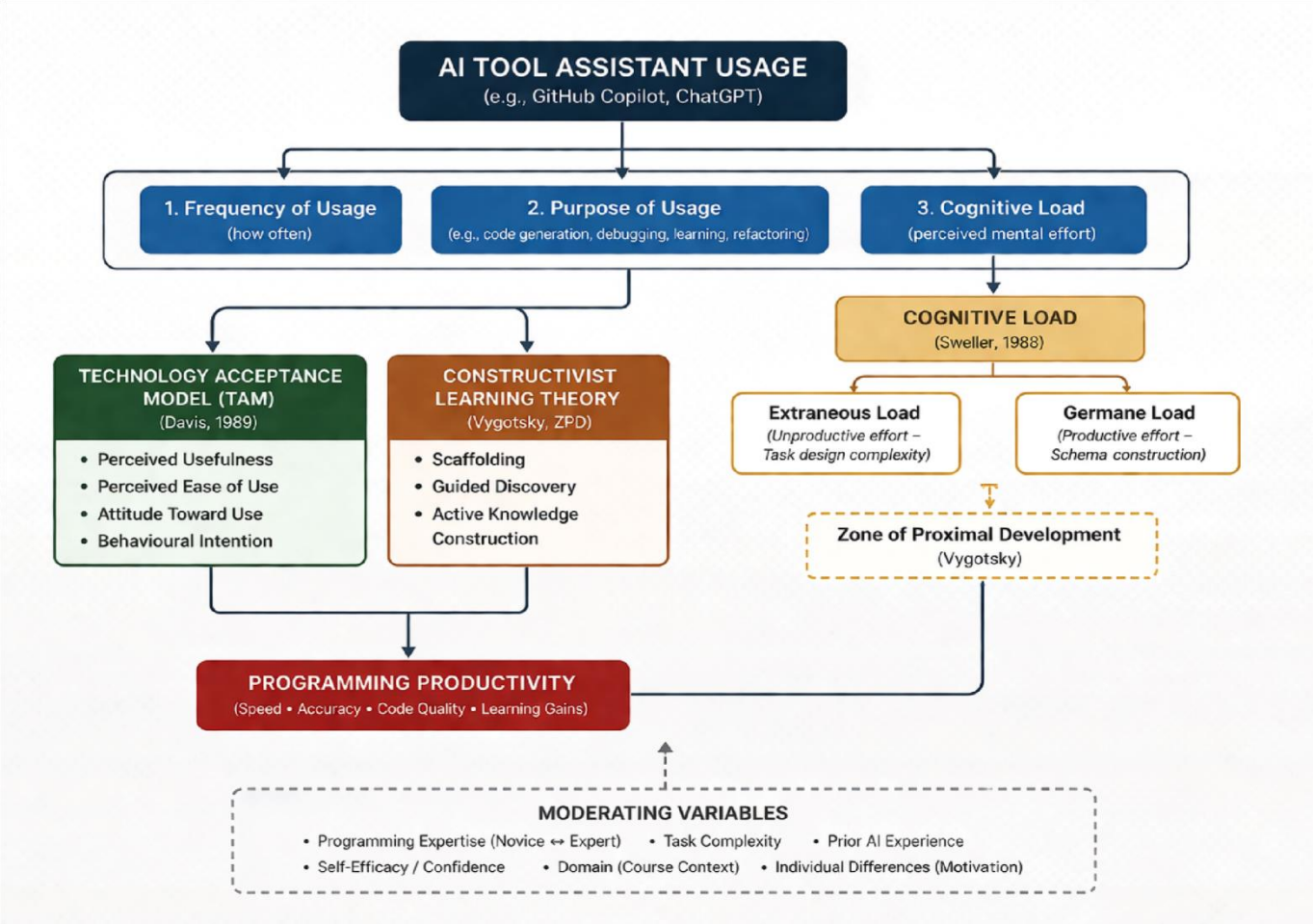


Figure 3 Theoretical Framework of the Study.

2.2 Empirical Studies

The studies reviewed below were published between 2022 and 2025 and examine AI coding assistants across the specific dimensions of productivity that this research sets out to measure: how quickly studentscomplete tasks, how good the resulting code is, how effectively students can debug and how confident they feel working independently.

2.2.1 Task Completion Speed

When it comes to raw speed, the evidence is fairly consistent. The most cited study in this space is Peng et al. (2023), who ran a randomized controlled experiment and found that developers using

GitHub Copilot finished their assigned tasks **55.8% faster** than those working without it. The authors linked this advantage to reduced cognitive load that is Copilot handled enough of the routine work that developers could move through tasks without getting bogged down. Nijkamp et al. (2022) reached a similar conclusion with CodeGen, showing that large language models can reliably automate repetitive and formulaic coding tasks across a range of programming contexts.

It is worth noting, though, that both studies were conducted with professional developers under controlled conditions and not with undergraduate students working through real assignments in an academic setting. That distinction matters, and it is one of the motivating gaps this study aims to address.

2.2.2 Code Quality and Debugging

The picture becomes considerably murkier when the focus shifts from speed to quality. Zhang et al. (2023) examined how developers actually use Copilot in day-to-day practice and found that the code it generates is often unreliable that is generated code is syntactically reasonable on the surface but logically flawed underneath. Fixing these problems added a significant verification burden that ate into the time savings users expected. Barke et al. (2023) observed a related dynamic: rather than writing code themselves, many developers slipped into a reviewer role, evaluating and adjusting what the AI produced. For experienced programmers, this can be an efficient division of labour. For novice students, it is a risk as accepting code they do not fully understand means they are also poorly positioned to spot or correct the errors within it. The downstream effect on debugging ability is real and worth taking seriously.

2.2.3 Perceived Competence and Independent Problem-Solving

Perhaps the most pressing concern raised in the educational literature is what regular AI tool use does to students' belief in their own abilities. Becker et al. (2023) argued that when students can get working code with minimal effort, they lose access to the kind of productive struggle that actually builds programming skill. Problem-solving is not just a means to an end rather it is how students develop the metacognitive awareness that lets them transfer knowledge to new situations. When that process is short-circuited, so is the development it was supposed to produce.

Kasneci et al. (2023) took a more balanced view, acknowledging that ChatGPT can be genuinely useful as a personalized learning companion which is capable of explaining concepts clearly, adapting to a student's level and helping them work through confusion in a low-pressure way. Even so, they cautioned that this benefit has a shadow side. When students grow accustomed to receiving ready-made explanations, they may gradually lose the habit of reasoning things through on their own. The result is a kind of intellectual passivity resulting in students who can follow explanations but struggle to generate their own thinking when no AI is there to prompt them.

2.2.4 Comparisons, Contradictions, and Research Gaps

Looking across the studies, one finding stands out as genuinely settled: AI coding tools make people achieve results faster. That much is consistent across experimental and observational research alike (Peng et al., 2023; Nijkamp et al., 2022). Almost everything else is more debatable. On the question of learning and competence, researchers disagree. Some see AI tools as effective scaffolds that help

students grow into more capable programmers over time (Kasneci et al., 2023). Others are more sceptical, arguing that the ease of AI-generated solutions actively gets in the way of developing genuine skill (Becker et al., 2023). On code quality, the evidence similarly pulls in different directions as speed gains are real, but so are the quality problems that come with poorly verified AI output (Zhang et al., 2023). Three specific gaps in the existing literature directly motivate this study. First, virtually all the high-quality empirical work has been done with professional developers, not with undergraduate students navigating real academic pressures. Second, productivity has almost always been measured as a single variable which is task completion time while leaving code quality, debugging ability, and perceived independence largely unexamined. Third, very little attention has been paid to how individual student characteristics which are things like prior experience, skill level, and familiarity with AI tools help shape the outcomes of AI-assisted coding. This study is designed to address all three of these gaps.

2.3 Conceptual Framework

The conceptual framework brings the theoretical and empirical work above into a single operational model that is directly tied to this study's research questions. It identifies the key variables, explains how they are defined and measured and also describes how they are expected to relate to one another.

2.3.1 Independent Variable: AI Coding Assistant Usage

The study's primary independent variable is AI coding assistant usage, which is understood along two distinct dimensions. The first is frequency of use which means: how often a student turns to tools like GitHub Copilot or ChatGPT while working on programming tasks. The second is purpose of use that is: what the student is actually using the tool for, whether that is generating code, working through a bug, or trying to understand a concept they are unsure about. This two-part operationalization matters because, as both TAM and the broader literature suggest, not all AI tool use is equivalent (Davis, 1989; Barke et al., 2023). A student who uses AI purposefully and selectively is likely to experience very different outcomes from one who reaches for it reflexively before attempting anything independently.

2.3.2 Dependent Variable: Programming Productivity

Rather than treating productivity as a single measure, this study defines **programming** productivity as a multi-dimensional construct with four distinct components. These are: (1) task completion time that is how quickly students finish assigned programming work (Peng et al., 2023); (2) code quality meaning the correctness, efficiency, and readability of the code they produce (Zhang et al., 2023); (3) debugging efficiency which is how well students can identify and fix errors in their code (Barke et al., 2023); and (4) perceived programming competence and independence that is how confident students feel in their own abilities when working without AI assistance (Becker et al., 2023; Kasneci et al., 2023). Defining productivity this broadly is deliberate. Both Cognitive Load Theory and Constructivist Learning Theory make clear that speed alone is an incomplete picture, what actually matters is whether students are genuinely learning and growing, not just finishing tasks faster.

2.3.3 Moderating Variable: Student Characteristics

The framework also recognizes that the relationship between AI tool use and productivity is not the same for every student. **Student characteristics** such as prior programming experience, current skill level, preferred learning style, and familiarity with AI tools, are expected to moderate that relationship in meaningful ways. Barke et al. (2023) found that more experienced programmers tend to use AI in a targeted, strategic way, whereas less experienced students are more likely to lean on it passively. Kasneci et al. (2023) similarly found that how students engage with AI-generated explanations varies considerably depending on how they learn. Building these factors into the model allows the framework to account for the kind of variation in outcomes that studies treating all students as equivalent have tended to overlook.

2.3.4 Relationship Between Variables

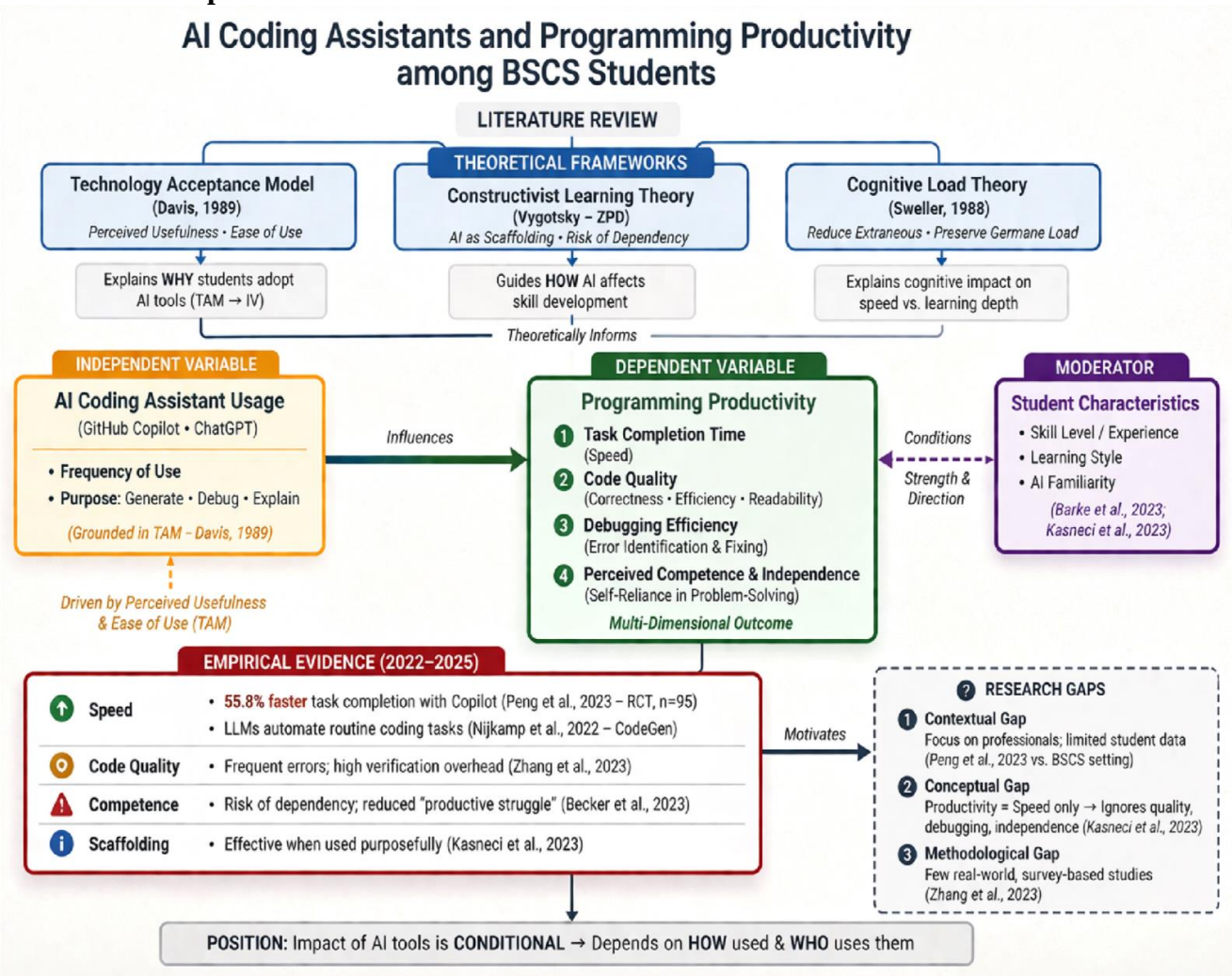


Figure 4 Conceptual Framework of the Study: Integration of Theoretical Frameworks, AI Coding Assistant Usage, Programming Productivity, Empirical Evidence, and Research Gaps

Taken together, the framework proposes that **AI tool usage** has a direct effect on **programming productivity** across all four of its dimensions, and that this effect is shaped whether strengthened or

weakened directly by **student characteristics**. The three theoretical frameworks each contribute a piece of the explanatory picture. TAM accounts for why students choose to use these tools and how frequently (Davis, 1989; Peng et al., 2023). Constructivist Theory explains how the scaffolding AI provides interacts with what students can and cannot yet do on their own, and what that means for genuine skill development (Kasneci et al., 2023; Becker et al., 2023). Cognitive Load Theory illuminates the mechanism: by reducing the mental effort required for routine coding, AI tools free up cognitive resources but in doing so, they may also quietly prevent the effortful processing that leads to lasting understanding (Sweller, 1988; Peng et al., 2023).

Table 1 Summary of Related Literature on AI Coding Assistants and Programming Productivity

Author(s) & Year	Research Focus	Context	Sample/ AI Tool(s)	Key Findings
Peng et al. (2023)	Controlled experiment: GitHub Copilot on task speed	Professional devs (n=95)	GitHub Copilot	Tasks completed 55.8% faster; reduced cognitive load drives gains
Nijkamp et al. (2022)	LLM-based code generation across diverse tasks	Benchmark datasets	CodeGen (LLM)	LLMs automate routine code effectively; quality varies with task complexity
Zhang et al. (2023)	Real-world Copilot usage patterns and challenges	Developers (survey + interview)	GitHub Copilot	Speed gains offset by verification burden; frequent incorrect suggestions
Barke et al. (2023)	How programmers interact with AI code-generating models	Developers (observational)	GitHub Copilot	Two modes: acceleration & exploration; novices risk uncritical acceptance
Becker et al. (2023)	Educational impact of AI code generation on CS education	CS educators & students	GitHub Copilot, ChatGPT	AI disrupts productive struggle; risk of surface learning & dependency
Kasneci et al. (2023)	Opportunities & challenges of LLMs for education	Educational review (broad)	ChatGPT	Effective personalized aid; breeds intellectual passivity if over-relied upon
Davis (1989)	Technology Acceptance Model — foundational theory	Information systems users	N/A (theory)	Perceived usefulness & ease of use drive technology adoption behaviour

Sweller (1988)	Cognitive Load Theory — foundational theory	Educational psychology	N/A (theory)	Working memory limits; extraneous vs. germane load

3. METHODOLOGY

3.1 Research Design

The study used a quantitative descriptive survey research design with a cross sectional. A quantitative design was selected as the study is interested in quantifying relationships and patterns between AI coding assistant usage and programming productivity through numerically coded data that can be statistically analyzed. A descriptive survey design is appropriate because the goal is not to manipulate variables but to systematically describe the current state of AI tool usage and its perceived effects among BSCS students (Creswell & Creswell, 2018).

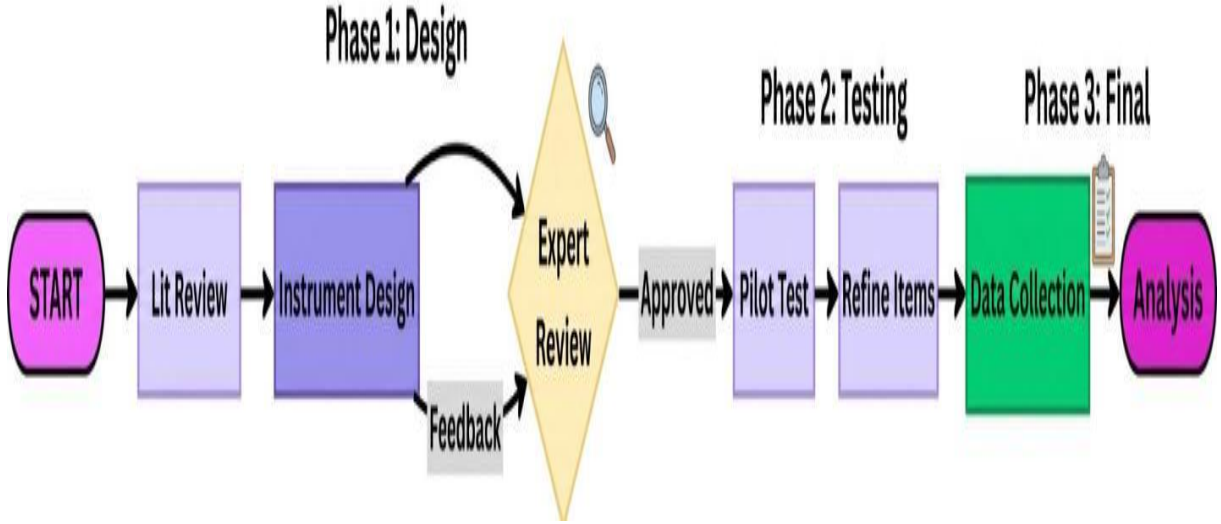


Figure 5 Methodological Flowchart of the Study Process.

A cross-sectional design collects data at one point in time, making it efficient for understanding students' views during an active academic period. It supports the study by examining AI tool usage, its impact on productivity, and its relationship with academic performance, including task completion, code quality, debugging and problem-solving independence.

3.2 Population and Sampling

3.2.1 Target Population

The target population consisted of undergraduate computing students enrolled at universities in Pakistan who actively use programming tools and have exposure to AI coding assistants such as

GitHub Copilot or ChatGPT in academic settings. The sample included students from BSCS, Software Engineering, Information Technology, and computing-adjacent disciplines.

3.2.2 Sampling Technique

Convenience sampling via an online Google Forms survey was employed, given the exploratory and time-constrained nature of this study. Participants were reached through academic coordinators, class representatives, and peer networks. While this approach introduces self-selection considerations, it is widely used and accepted in student-centered educational technology research, particularly at early stages of investigation (Cohen et al., 2018).

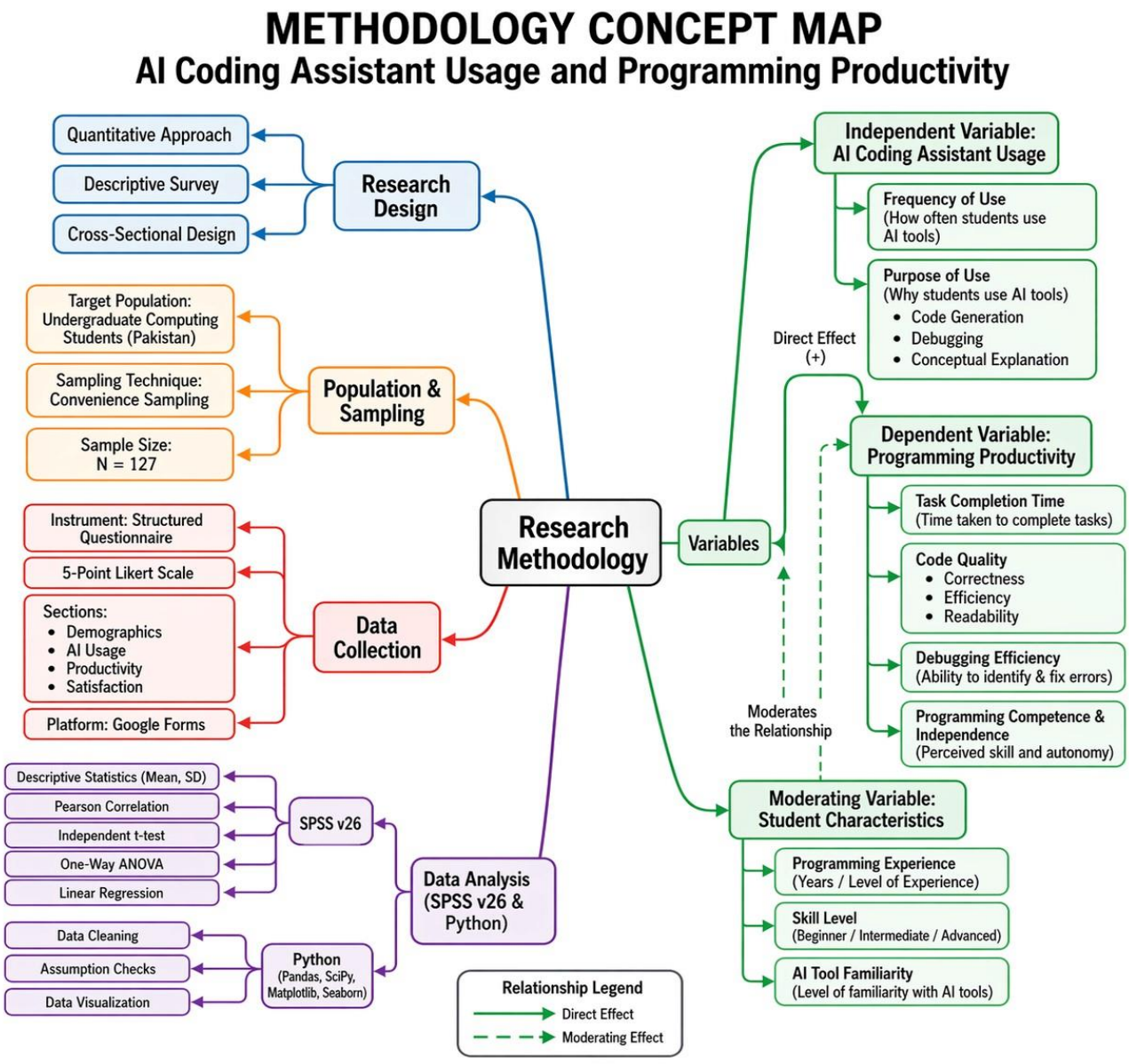


Figure 6 Methodological concept map of the Research Methodology

3.2.3 Sample Size and Adequacy

A minimum sample of n = 100 completed responses was targeted. The final usable sample consisted of N = 127 participants, exceeding this threshold. A sample of this size is sufficient and defensible for the statistical techniques employed, specifically descriptive statistics, Pearson correlation, independent samples t-test, one-way ANOVA, and simple linear regression, provided assumptions are checked and appropriate tests selected based on the actual distribution of data (Field, 2018; Pallant, 2020). Studies in educational technology research with comparable designs have yielded meaningful findings with samples in the range of 80–120 participants (Kasneci et al., 2023). Incomplete responses (more than 20% missing values) were excluded from the final analysis dataset.

3.3 Instrumentation

The main tool was a structured self-administered online questionnaire, hosted on Google Forms. Attitudinal and perception items were measured with a 5-point Likert scale (1= Strongly Disagree to 5 = Strongly Agree) and usage items were measured with a 5-point frequency scale (1= Never to 5= Always). The responses were automatically saved and exported to a .csv file. The .csv file was then imported into SPSS for analysis.

Table 2 Questionnaire Structure and Coverage

Section	*Theme*	*Items Covered*
A	Demographics	Age, gender, semester, area of study, programming experience, AI tool familiarity
B	AI Tool Usage Frequency	Tools used, frequency of use, purpose (code generation, debugging, conceptual help)
C	Perceived Productivity & Learning	Task completion speed, code quality, debugging efficiency, learning outcomes, dependency risk, problem-solving ability
D	Overall Satisfaction	Overall experience, enjoyment, willingness to recommend, satisfaction with AI tool performance

3.4 Validity and Reliability

3.4.1 Content Validity

Content validity was established through expert review. The draft questionnaire was submitted to a panel of subject matter experts comprising faculty in software engineering and educational research methodology. Items were evaluated for clarity, relevance, and alignment with the study's conceptual framework. Items that received insufficient ratings were revised before deployment.

3.4.2 Reliability Cronbach's Alpha

Internal consistency reliability was assessed using Cronbach's Alpha (α) after data collection. A threshold of $\alpha \geq 0.70$ is considered acceptable for survey instruments in educational research

(Nunnally & Bernstein, 1994). Subscales falling below this threshold were noted as limitations and their item-total correlations were inspected accordingly.

3.4.3 Pilot Testing

A pilot test was conducted with a small sample of BSCS students (n = 15–30) prior to full deployment to identify ambiguous items, estimate completion time, and compute preliminary reliability estimates. Feedback was used to refine item wording before the survey was finalised and distributed.

4. Results and Discussions

This section presents the findings of the survey in accordance with the analytical framework outlined in the methodology. The data were examined using descriptive statistics, Pearson correlation, independent samples t-test, one-way ANOVA, and linear regression. Results are organized as follows: demographic profile, AI tool usage frequency, perceived productivity outcomes (task efficiency, code quality, debugging), learning and problem-solving outcomes, dependency-related concerns, overall satisfaction, and inferential statistics testing the study's hypothesis.

4.1 Demographic Analysis

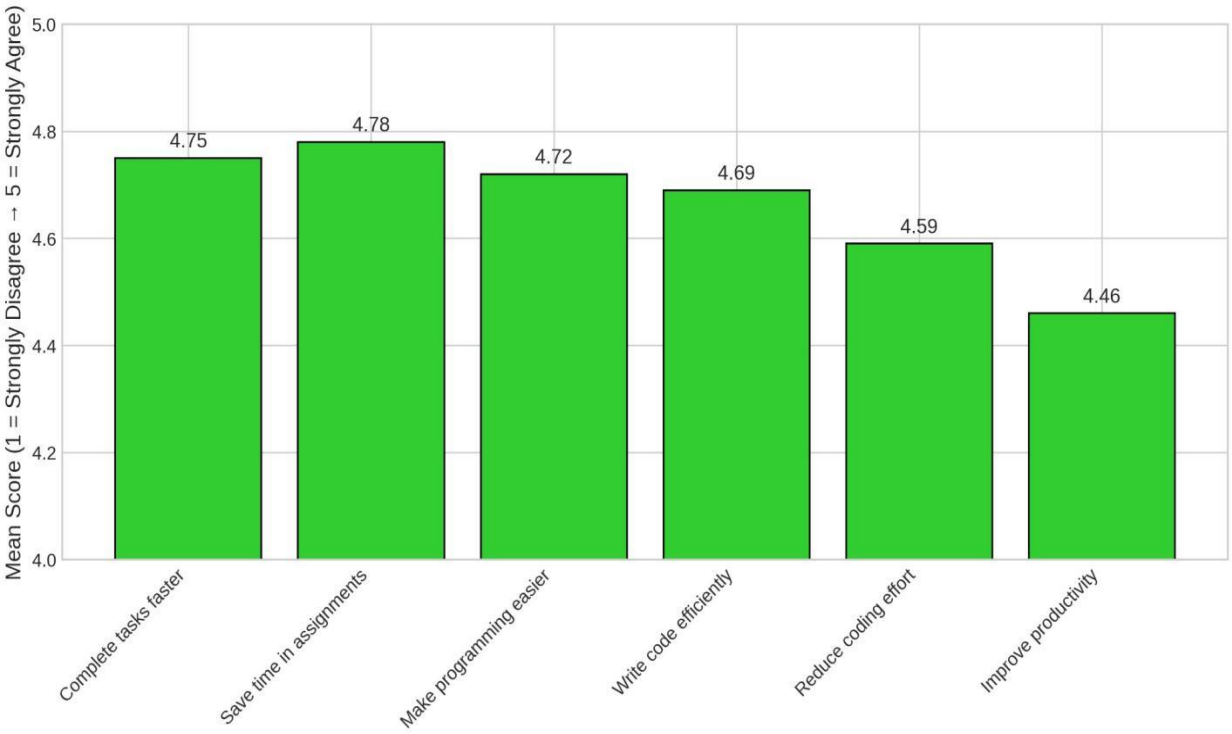
A total of **127 participants** completed the survey, yielding a final usable sample above the minimum threshold of n = 100 established in the methodology. The demographic breakdown is presented in

Table 3 Demographic Profile of Respondents (N = 127) Note. Percentages may not total 100% due to rounding.

Variable	Category	N n	Percentage (%)
Age	18–20	51	40.2%
	21–23	67	52.8%
	24–26	8	6.3%
	27+	1	0.8%
Gender	Female	65	51.2%
	Male	62	48.8%
Semester	1–2	18	14.2%
	3–4	33	26.0%
	5–6	50	39.4%
	7–8	26	20.5%
Area of Study	BSCS	50	39.4%
	Software Engineering	11	8.7%
	Other	65	51.2%
	IT	1	0.8%
Programming Experience	Beginner	47	37.0%
	Intermediate	66	52.0%
	Advanced	14	11.0%
AI Tool Familiarity	Low	23	18.1%

	Moderate	71	55.9%
	High	33	26.0%

Figure 4.2: Perceived Impact on Task Efficiency & Productivity



The majority of participants were aged 21–23 years (52.8%) followed by 18–20 (40.2%), indicating that the sample reflects typical undergraduate enrolment age ranges. Gender distribution was approximately equal: 51.2% female and 48.8% male. Semester distribution revealed that most respondents were in semesters 5–6 (39.4%) and 3–4 (26.0%), suggesting mid-programme exposure to programming coursework and AI tools. BSCS students constituted 39.4% of the sample, with Software Engineering students (8.7%) and a broader "Other" category (51.2%) the latter including students from computing-adjacent disciplines at the same institutions completing the representation. In terms of programming experience, the majority reported intermediate-level proficiency (52.0%)while 37.0% were beginners and 11.0% were advanced. Regarding familiarity with AI tools, 55.9% held moderate familiarity, 26.0% reported high familiarity and 18.1% had low familiarity. These figures indicate a sample with varied but generally non-expert exposure to AI coding assistants which appropriately reflects the undergraduate population under study.

4.2 AI Coding Assistant Usage Frequency

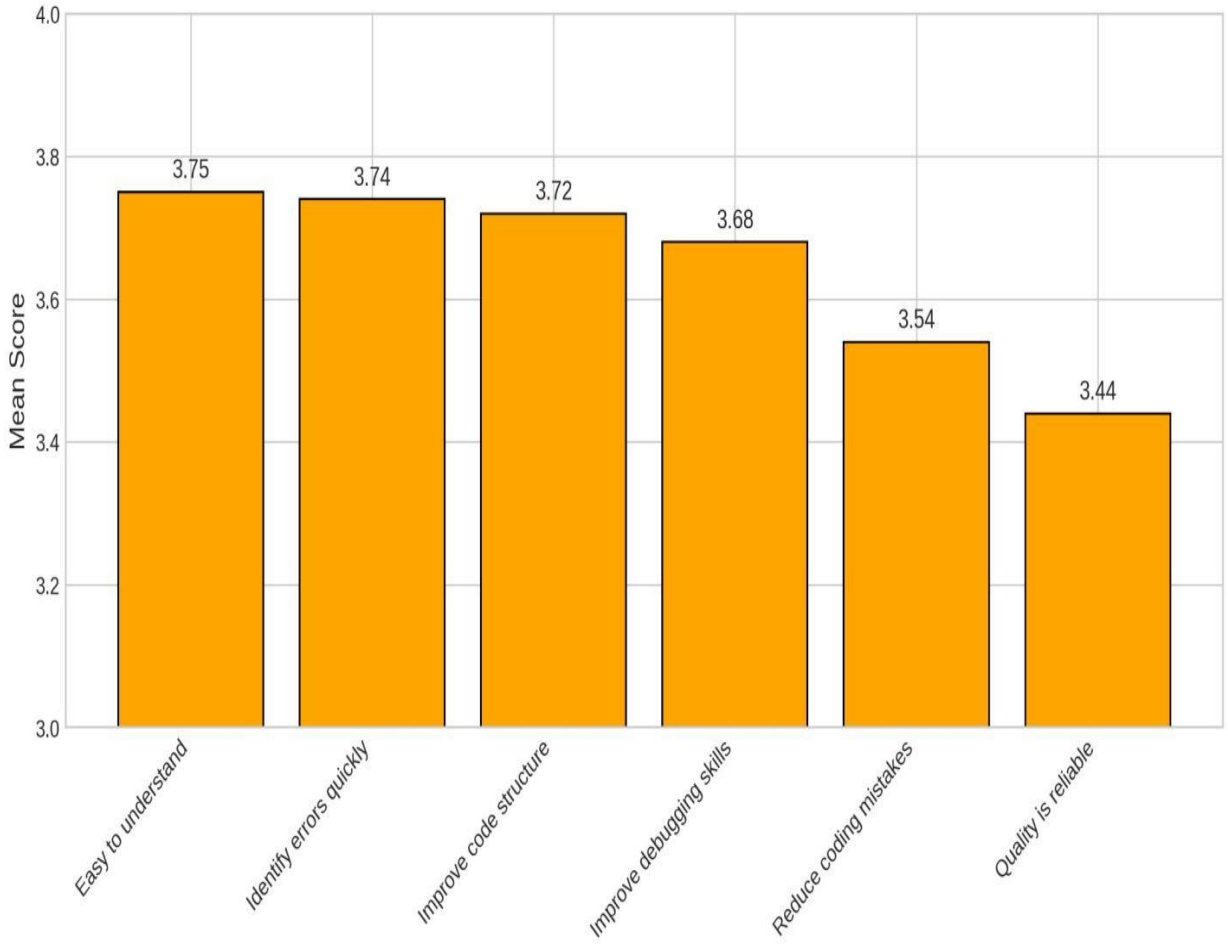
Section B of the questionnaire examined the frequency of AI tool usage across six specific programming scenarios. Responses were recorded on a five-point frequency scale (1 = Never to 5 = Always). Table 5 presents item-level mean scores and standard deviations.

Table 5. Descriptive Statistics AI Tool Usage Frequency (N = 127)

Item	M	SD	Interpretation
I use AI coding assistants while doing programming tasks.	3.26	1.27	Sometimes
I use AI tools to generate code snippets.	3.09	1.33	Sometimes
I use AI tools to understand programming concepts.	3.52	1.33	Often
I use AI tools for debugging errors.	3.20	1.44	Sometimes
I rely on AI tools during assignments.	3.54	1.11	Often
I use AI tools during exams or practice sessions.	3.57	1.36	Often
Overall AI Usage Composite Score	3.36	0.96	—

Note. Scale: 1 = Never, 2 = Rarely, 3 = Sometimes, 4 = Often, 5 = Always. Composite score = mean of all six items.

Figure 4.3: Perceived Code Quality & Debugging Support



The overall AI usage composite score was **M = 3.36 (SD = 0.96)**, reflecting moderate frequency of AI tool use across the sample. The highest-rated usage context was *using AI tools during exams or practice sessions* (M = 3.57, SD = 1.36), followed by *relying on AI tools during assignments* (M = 3.54, SD = 1.11) and *using AI tools to understand programming concepts* (M = 3.52, SD = 1.33). These items approached the threshold of frequent (Often) use, suggesting that students turn to AI tools more consistently in structured academic tasks and conceptual learning than in open-ended coding work. The lowest mean was observed for *using AI tools to generate code snippets* (M = 3.09, SD = 1.33), indicating that passive code generation is less prevalent than conceptual support or assignment assistance. The pattern of responses suggests that AI coding assistants serve primarily as academic support tools among this sample rather than as habitual coding companions.

4.3 Perceived Productivity Outcomes

4.3.1 Task Efficiency and Coding Productivity

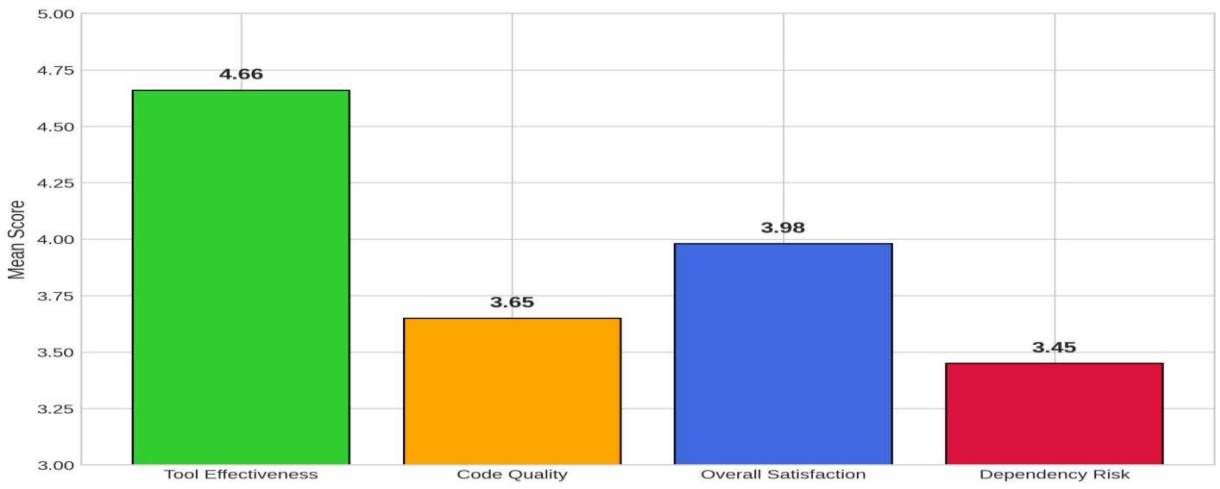
Students reported strongly positive perceptions of AI tools' impact on task completion speed and overall coding efficiency. Table 6 presents the descriptive statistics for the six items comprising this subscale.

Table 6. ***Descriptive Statistics Perceived Task Efficiency and Productivity (N = 127)***

Item	M	SD	Interpretation
AI coding assistants help me complete tasks faster.	4.75	0.98	Strongly Agree
AI tools save time during assignments.	4.78	0.92	Strongly Agree
AI tools make programming easier for me.	4.72	1.03	Strongly Agree
AI tools help me write code more efficiently.	4.69	1.08	Strongly Agree
AI tools reduce my coding effort.	4.59	1.22	Strongly Agree
AI tools improve my productivity.	4.46	1.37	Agree
Composite Tool Effectiveness Score	4.66	0.66	—

Note. Scale: 1 = Strongly Disagree to 5 = Strongly Agree. Composite score = mean of all six items.

Figure 4.4: Comparison of Major Constructs (Mean Scores)



The composite Tool Effectiveness score was **M = 4.66 (SD = 0.66)**, the highest of all subscales measured in this study, indicating near-unanimous agreement that AI tools enhance coding efficiency. The item *AI tools save time during assignments* yielded the highest mean in the entire instrument (M = 4.78, SD = 0.92), closely followed by *AI coding assistants help me complete tasks faster* (M = 4.75, SD = 0.98). These scores indicate that time-saving is the most widely perceived and strongly endorsed benefit of AI tool use among students. Similarly, *AI tools make programming easier* (M = 4.72) and *AI tools help me write code more efficiently* (M = 4.69) received strongly positive endorsements. The relatively lower though still high mean for *AI tools improve my productivity* (M = 4.46, SD = 1.37) alongside a larger standard deviation suggests slightly more variability in how students define and experience productivity gains, potentially reflecting differences in individual workflows and programming experience levels.

4.3.2 Code Quality and Debugging Efficiency

Table 7 presents student perceptions of how AI tools affect code quality, debugging ability, and the accuracy of AI-generated outputs.

Table 7. *Descriptive Statistics Perceived Code Quality and Debugging (N = 127)*

Item	M	SD	Interpretation
AI-generated code is easy to understand.	3.75	0.86	Agree
AI tools help me identify errors quickly.	3.74	1.07	Agree
AI tools help improve my code structure and readability.	3.72	1.02	Agree
AI tools improve my debugging skills.	3.68	1.03	Agree
AI tools reduce the number of coding mistakes I make.	3.54	1.23	Agree
The quality of AI-generated code is reliable.	3.44	0.97	Agree
Composite Code Quality Score	3.65	0.69	—

Note. Scale: 1 = Strongly Disagree to 5 = Strongly Agree. Composite = mean of all six items.

The composite code quality score was **M = 3.65 (SD = 0.69)**, reflecting moderate-to-positive agreement that AI tools contribute to code quality improvements. Students agreed most strongly that *AI-generated code is easy to understand* (M = 3.75, SD = 0.86) and that *AI tools help identify errors quickly* (M = 3.74, SD = 1.07). So AI outputs are generally seen as readable, and their value for spotting errors is appreciated. The lowest item in this subscale was "the quality of AI-generated code is reliable" (M = 3.44, SD = 0.97). The standard deviation here is relatively low, which means the doubt about reliability is fairly consistent across the sample. This is an interesting pattern: students get clear efficiency benefits from AI tools but at the same time hold back on trusting the output without checking it. That points to a culture of verification rather than blind acceptance in this cohort. This skepticism about AI code reliability is justified. Research shows that AI-generated code can include hidden security flaws and logical errors, with a notable portion containing insecure patterns (Pearce et al., 2025). User studies also raise concerns about the real-world security risks of

relying on these tools (Sandoval et al., 2023). Together, these findings highlight that code verification is an essential skill that should be taught alongside AI tool use in programming education.

4.4 Learning, Problem-Solving, and Dependency Outcomes

This subscale examined the dual role of AI tools in supporting learning and problem-solving while also potentially creating dependency. Twelve items were organized around cognitive assistance and risk of over-reliance. On the learning and problem-solving side, students agreed that *AI tools help me learn new programming concepts* (M = 4.16, SD = 0.75) and that *I can solve problems better with AI assistance* (M = 4.17, SD = 0.92). The item *AI tools improve my understanding of coding logic* scored M = 4.08 (SD = 0.99), and *AI tools encourage me to explore new coding techniques* scored M = 3.80 (SD = 1.11). These results suggest that AI tools are perceived not only as task shortcuts, but as active contributors to students' conceptual understanding and exploratory behaviour which is a finding consistent with Constructivist learning principles regarding scaffolding within the Zone of Proximal Development.

Nonetheless, dependency-related items produced moderate-to-concerning means. "*AI tools affect my independent problem-solving ability"* received a mean of M = 3.78 (SD = 1.18) and "*AI tools reduce my need to think deeply about problems"* scored M = 3.65 (SD = 1.23). Notably, the composite Dependency Risk score was **M = 3.45 (SD = 0.70)**. While this is below the agreement threshold, it nonetheless signals a recognized risk among approximately one-third to one-half of respondents. Encouragingly, *I double-check AI-generated code before using it* yielded M = 3.73 (SD = 1.17), and *I trust AI-generated solutions without verifying them* produced the lowest dependency-item mean at M = 3.06 (SD = 1.35), indicating that most students exercise some critical review of AI outputs rather than accepting them uncritically.

4.5 Overall Satisfaction with AI Coding Assistants

Section E of the instrument evaluated students' overall satisfaction, enjoyment, and willingness to recommend AI tools. Table 8 presents the descriptive statistics for this subscale.

Table 8. ***Descriptive Statistics Overall Satisfaction (N = 127)***

Item	M	SD	Interpretation
AI tools enhance my learning experience in programming.	4.15	0.81	Agree
My overall experience with AI coding assistants is positive.	4.10	0.84	Agree
AI tools make programming more enjoyable.	4.02	0.86	Agree
I would recommend AI tools to other students.	4.01	0.93	Agree
I am satisfied with the performance of AI coding assistants.	3.61	1.16	Agree
Composite Satisfaction Score	3.98	0.60	—

Note. Scale: 1 = Strongly Disagree to 5 = Strongly Agree. Composite = mean of all five items.

The composite Satisfaction score was **M = 3.98 (SD = 0.60)**, indicating broadly positive experiences with AI coding assistants. The highest endorsement was for *AI tools enhance my learning experience in programming* (M = 4.15, SD = 0.81) and *my overall experience with AI coding assistants is positive* (M = 4.10, SD = 0.84). The item *I would recommend AI tools to other students* scored M = 4.01 (SD = 0.93), reflecting strong student advocacy. The lowest mean in this subscale was *I am satisfied with the performance of AI coding assistants* (M = 3.61, SD = 1.16), The standard deviation here was the largest in the subscale, which means satisfaction with actual performance varies more than the other items. This may be linked back to the concerns about code reliability picked up in the quality subscale.

4.6 Inferential Statistics

4.6.1 Pearson Correlation Analysis

Pearson correlation coefficients were computed to examine bivariate relationships among the five composite variables. The correlation matrix is presented in Table 9.

Table 9. Pearson Correlation Matrix of Composite Variables (N = 127)

Variable	1	2	3	4
1. AI Usage Frequency	—			
2. Tool Effectiveness	0.157	—		
3. Skill Improvement	0.192*	0.336***	—	
4. Overall Satisfaction	0.133	—	—	—
5. Dependency Risk	0.199*	—	—	—

Note. * p < .05 (two-tailed). *** p < .001 (two-tailed). Values represent Pearson r coefficients.

The most statistically significant and practically meaningful correlation observed was between **Tool Effectiveness and Skill Improvement** (r = 0.336, p < .001), indicating the students who see AI tools as effective for boosting task efficiency are also significantly more likely to report improvements in their coding skills and conceptual understanding. This supports the idea that perceived efficiency gains do not stand apart from broader learning outcomes. AI Usage Frequency was positively correlated with Skill Improvement (r = 0.192, p = .031), suggesting that more frequent use of AI tools is modestly associated with greater perceived skill development. Similarly, AI Usage Frequency was positively correlated with Dependency Risk (r = 0.199, p = .025),

indicating that students who use AI tools more often also report higher levels of dependency-related concerns which is a meaningful finding that highlights a dual nature to frequent AI use: while it supports skill development, it simultaneously increases perceived dependence. AI Usage Frequency did not show statistically significant correlations with Tool Effectiveness ($r = 0.157$, $p = .077$) or Overall Satisfaction ($r = 0.133$, $p = .138$), suggesting that mere frequency of use does not necessarily lead to stronger perceptions of efficiency benefit or overall satisfaction.

4.6.2 Independent Samples T-Test: Gender Differences

An independent samples t-test was conducted to determine whether perceived skill improvement differed significantly between male and female respondents. The analysis yielded **$t(125) = -0.607$, $p = .545$** (two-tailed). Male students reported a composite skill improvement mean of $M = 3.61$ ($SD = 0.76$, $n = 62$), while female students reported $M = 3.68$ ($SD = 0.62$, $n = 65$). The difference is not statistically significant. So in this sample, gender does not moderate the perceived skill improvement outcomes of AI coding assistant use.

4.6.3 One-Way ANOVA: Differences by Area of Study

A one-way ANOVA was conducted to assess whether students' area of study (BSCS, Software Engineering, Other) was associated with differences in perceived productivity improvement. Group means were as follows: BSCS ($M = 4.10$, $SD = 0.58$, $n = 50$), Software Engineering ($M = 4.14$, $SD = 0.40$, $n = 11$), and Other ($M = 4.20$, $SD = 0.56$, $n = 65$). The ANOVA did not reveal a statistically significant difference across groups, $F(2, 123) = 0.438$, $p = .646$. These results indicate that perceived productivity benefits are not significantly different across academic programmes, which means AI tools are seen as similarly useful regardless of which computing discipline a student is in.

5. Discussion

This study adds to a growing understanding of how AI coding assistants affect programming productivity among undergraduate students. Instead of showing that these tools simply improve performance in all areas, the results present a more balanced view. While students experienced clear improvements in efficiency, their perceptions of code quality were only moderate. The connection between using AI tools and developing programming skills was present but not very strong, and there was also some level of dependency, although students were generally aware of it. Overall, these findings are explained by linking them to the theories and previous studies discussed earlier in the literature review. The overall Tool Effectiveness score was the highest among all scales, with a mean of 4.66. Students especially agreed that AI tools help save time and support task completion. This finding is consistent with Peng et al. (2023), who showed that developers completed tasks much faster using GitHub Copilot and with Nijkamp et al. (2022), who highlighted the ability of AI to handle routine programming tasks. This study extends those findings to undergraduate students, suggesting that the benefits of faster work and reduced effort are also clearly seen in academic settings. From the perspective of the Technology Acceptance Model by Davis (1989), the very high Tool Effectiveness scores show that students strongly perceive AI coding assistants as useful. This high level of perceived usefulness helps explain the moderate to high usage levels observed earlier. In line with Barke et al. (2023), students are motivated to use these tools because they make their work easier and more efficient. The relatively high satisfaction score of 3.98 and the strong intention

to recommend with a mean of 4.01 further show that students generally accept and value these tools in their academic work.

Although students strongly supported the efficiency benefits of AI tools, their views on code quality and reliability were more moderate, with an average score of 3.65. The lowest score in this group was for the statement *The quality of AI-generated code is reliable* with a mean of 3.44. This is important because it reflects earlier findings by Zhang et al. (2023), who reported frequent errors in AI-generated code, and by Barke et al. (2023), who found that developers often review and guide AI outputs instead of accepting them without question. In this study, a similar pattern can be seen. The statement *I trust AI-generated solutions without verifying them* had the lowest score in the dependency scale at 3.06, while *I double-check AI-generated code before using it* scored higher at 3.73. This shows that most students are careful and tend to check AI-generated code before using it. This behavior matches what Cognitive Load Theory by Sweller (1988) describes as useful mental effort that supports learning. However, the need to constantly review and correct AI-generated code may also reduce overall efficiency. This may help explain why the regression results showed only a weak link between how often students use AI tools and how effective they think they are. Using the tools more often does not always lead to better outcomes if students have to spend extra time checking and fixing the code.

The statistically significant positive correlation between Tool Effectiveness and Skill Improvement ($r = 0.336$, $p < .001$) is one of the more theoretically interesting findings of this study. It shows that students who get efficiency benefits from AI tools are also more likely to report broader improvements in their coding skills and conceptual understanding. This pushes back on the idea that efficiency gains have to come at the cost of real learning. The result is consistent with Kasneci et al.'s (2023) view of AI as a personalised learning aid that can support skill development when used with intent. The modest but significant correlation between usage frequency and skill improvement ($r = 0.192$, $p = .031$) gives empirical support for the study's main hypothesis (H1). Through the lens of Vygotsky's Zone of Proximal Development, AI tools appear to act as scaffolds that extend students' capability beyond what they could achieve unassisted. The directional trend in the experience-level ANOVA, where beginners reported slightly higher perceived skill gains than advanced students, fits this reading: scaffolding should help most where the gap between current and potential capability is widest. The non-significant ANOVA result ($F = 1.096$, $p = .337$) means we should not over-interpret it. A larger sample of advanced students would be needed to confirm or rule out this pattern.A larger sample of advanced students would be needed to confirm or rule out this pattern.
The positive correlation between AI usage frequency and dependency risk ($r = 0.199$, $p = .025$) is a finding that deserves educational attention. Students who use AI tools more often are more likely to admit to dependency-related concerns about their own coding independence. The correlation does not prove a causal relationship, but it does mean that increased AI engagement comes with at least a self-perceived drop in independent confidence. This lines up with Becker et al.'s (2023) argument about the long-term risks of sustained AI reliance in educational programming contexts. The relationship is statistically significant, but the lack of correlation with overall satisfaction suggests students are sitting in a more nuanced motivational state. They benefit from AI and at the same time feel uneasy about it. This is a tension that educators will need to address by drawing a clear line between AI-assisted and AI-dependent behaviour.

6. Limitations of the study

This study has several limitations that should be taken into account when interpreting and generalizing its findings. First, the reliance on self-reported perceptual data introduces common method bias, as responses reflect students' subjective beliefs about AI tool impact rather than objectively measured performance outcomes. Perceived skill improvement is not equivalent to demonstrated competence gains, and future studies should combine survey data with objective measures such as grades, code accuracy, or task completion time to give a clearer picture. Second, the use of convenience sampling through online distribution limits the representativeness of the sample. The inclusion of participants from different computing-related fields, along with a large "Other" category, may also make the results less specific to computer science students. Because of this, the findings should not be generalized beyond similar groups of undergraduate students without further research and validation. Third, the cross-sectional design captures AI tool perceptions at a single point in time which means we cannot draw conclusions about whether the observed relationships persist, strengthen, or change over the course of a student's academic program. Dependency risk, in particular, may be a time-sensitive construct that intensifies with continued exposure to AI tools which is something that only longitudinal data can capture. Fourth, the regression model explaining only 2.5% of variance in tool effectiveness by usage frequency confirms that the most important determinants of AI tool benefit remain unmeasured in this study which likely includes qualitative factors such as the intentionality and metacognitive awareness with which students engage with AI tools. Finally, the limited sample of advanced-level students (n = 14) restricts the statistical power of experience-level comparisons, and the directional findings in that ANOVA should be treated as preliminary rather than confirmatory.

7. Future Research Directions

Several directions for future inquiry emerge from the limitations and findings of this study. Longitudinal designs tracking the same cohort of students across multiple semesters would provide invaluable insight into whether AI tool benefits are sustained over time, how dependency evolves with prolonged usage, and whether early beneficial support from AI tools transitions into deeper learning or increased reliance as students' progress through their programs Experimental or semi-experimental studies, where access to AI tools is controlled across different groups, would make it possible to better understand cause and effect. This would go beyond the limits of correlation seen in the current study. Incorporating objective outcome measures such as automated code quality assessments, assignment grade comparisons, or standardized programming tests would considerably strengthen the validity and generalizability of findings. Qualitative follow-up research, including interviews or focus groups with students would provide deeper insight into how and why students use AI tools.. Understanding *why* students choose to use AI tools for certain tasks but not others and what critical reasoning, if any, they apply to AI-generated outputs would substantially advance theoretical models of AI-augmented learning behavior. Future studies should also examine the role of instructor guidance as a moderating variable. Students whose instructors actively discuss appropriate AI use strategies may show different dependency and skill development profiles compared to those receiving no such support. Finally, cross-national comparative studies between computing students in developing and developed country contexts would show whether the benefits and risks of AI tools are similar everywhere or depend on the learning environment.

8. Conclusion

This study looked at how AI coding assistants influence programming productivity among BSCS and other computing students using a survey-based approach with 127 participants. It adds useful insight to an area where most previous research has focused on professional developers rather than students, especially in real academic settings. The findings show that AI coding assistants are now widely used by students. Most participants found these tools helpful and easy to use and many rely on them regularly for tasks like writing code, debugging, and understanding difficult concepts. This suggests that AI tools have become a normal part of students' programming routines. Students generally felt that AI tools make programming more efficient. They reported that these tools help them complete tasks faster, save time, and make coding less difficult. Many also believed that AI can improve code quality but they do not fully trust the output. As a result, most students still check and verify the code before using it. The study also shows that AI tools can support learning. Students reported a better understanding of concepts and improved problem-solving skills when using them. However, the findings suggest that the benefits of AI in learning depend not only on how often students use it but also on how they use it. At the same 43 time, there are some concerns. Students who use AI tools more frequently may become somewhat dependent on them, which could affect their ability to solve problems on their own. No major differences were found between male and female students or across different computing programs, suggesting that these tools are useful for a wide range of students. Overall, AI coding assistants have clear potential to improve productivity, support learning, and make programming tasks easier. However, they should be used carefully. Students need to stay critical, double-check outputs, and continue practicing independent thinking. AI should be treated as a helpful support tool, not a replacement for learning and problem-solving.

References

[1] Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded Copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages,*

[2] *7*(OOPSLA1), 1–27.https://doi.org/10.1145/3586030

[3] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming is hard — or at least it used to be: Educational opportunities and challenges of AI code generation. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*.https://doi.org/10.1145/3545945.3569759

[4] Cohen, J. (1992). A power primer. *Psychological Bulletin, 112*(1), 155–159.https://doi.org/10.1037/0033-2909.112.1.155

[5] Cohen, L., Manion, L., & Morrison, K. (2018). *Research methods in education* (8th ed.). Routledge.

[6] Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th ed.). SAGE Publications.

[7] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly, 13*(3), 319–340.

[8] Denny, P., Kumar, V., &Giacaman, N. (2023). Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education (SIGCSE 2023)* (pp. 1136–1142). https://doi.org/10.1145/3545945.3569823

[9] Field, A. (2018). *Discovering statistics using IBM SPSS statistics* (5th ed.). SAGE Publications.

[10] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., &Hüllermeier, E. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274.https://doi.org/10.1016/j.lindif.2023.102274

[11] Nijkamp, E., Hayashi, H., Xiong, C., Savarese, S., & Zhou, Y. (2022). CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv*.https://doi.org/10.48550/arXiv.2203.13474

[12] Nunnally, J. C., & Bernstein, I. H. (1994). *Psychometric theory* (3rd ed.). McGraw-Hill.

[13] Pallant, J. (2020). *SPSS survival manual: A step by step guide to data analysis using IBM SPSS* (7th ed.). McGraw-Hill.

[14] 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.https://doi.org/10.1145/3610721

[15] Peng, S., Kalliamvakou, E., Cihon, P., &Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv*.https://doi.org/10.48550/arXiv.2302.06590

[16] Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., & Finnie-Ansley, J. (2023). "It's Weird That It Knows What I Want": Usability and interactions of novice programmers with GitHub Copilot. *ACM Transactions on Computer-Human*

[17] *Interaction*, *31*(1), 1–31.https://doi.org/10.1145/3617367

[18] Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S., & Dolan-Gavitt, B. (2023). Lost at C: A user study on the security implications of large language model code assistants. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security 2023)* (pp. 2205–2222). USENIX

[19] Association.https://www.usenix.org/system/files/usenixsecurity23-sandoval.pdf

[20] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285.

[21] Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using GitHub Copilot: An empirical study. *arXiv*.https://doi.org/10.48550/arXiv.2303.08733