Learning, Behavior, and Pedagogy: A Systematic Review of Generative AI Use in Programming Education

Tien-Chi Huang[1] and Hsin-Ping Tseng[2,*]

[1]Department of Information Management, National Taichung University of Science and Technology, Taichung, Taiwan
[2]Doctoral Program of Intelligent Engineering, National Taichung University of Science and Technology, Taichung, Taiwan
Email: tchuang@nutc.edu.tw (T.-C.H.); s1f11336002@nutc.edu.tw (H.-P.T.)
[*]Corresponding author
Manuscript received June 19, 2025; revised July 22, 2025; accepted September 4, 2025; published January 13, 2026

***Abstract*—With the rapid development of Generative Artificial Intelligence (GAI) technology, programming education has emerged as a core application domain. Through a systematic literature review of 45 relevant studies from the Semantic Scholar database from 2023-2025, this study examined the current applications of GAI as an auxiliary learning tool in programming education, and its impact on learning outcomes. The findings reveal that GAI-assisted instruction demonstrates significant effectiveness across seven learning indicators: programming knowledge and skills, computational thinking and logical reasoning, problem-solving ability, programming self-efficacy, learning achievement, code quality, and learning behaviors and engagement. While the majority of studies confirm that GAI enhances student performance in various areas such as task completion, test performance, code structure and quality, and promoting self-directed learning, some studies indicate that GAI use may reduce learning depth and lead to over-dependence in specific tasks or complex reasoning contexts. From a pedagogical perspective, GAI prompts a transformation in teachers' roles from knowledge transmitters to learning facilitators and guides, necessitating corresponding adjustments in curriculum design and assessment approaches. Based on the empirical findings, this study constructs an integrated conceptual model for GAI-assisted programming education integrating four core dimensions: implementation context factors, core influencing factors, learning performance indicators, and learning outcomes. The study identifies AI tool selection, students' foundational abilities, and task complexity as key variables affecting learning effectiveness, and synthesizes seven patterns of student learning behavior changes under GAI assistance, providing concrete theoretical foundations and implementation guidelines for educational practice.**

***Keywords*—Generative Artificial Intelligence (GAI), programming education, ChatGPT, learning outcomes, code quality, self-directed learning; pedagogical adaptation**

I. Introduction

With the rapid development of Generative Artificial Intelligence (GAI) technology and the widespread adoption of tools such as ChatGPT, GAI has become an important instrument across various domains. In the field of programming education, GAI can generate code and problem-solving suggestions in real-time, meeting learners' personalized support needs [1]. However, guiding students to use AI tools appropriately—preventing them from becoming mere answer providers while transforming them into effective auxiliary learning tools that promote deep learning and subsequent application—has emerged as a critical challenge in educational applications. In programming learning, students must not only master programming syntax and logical structures, but also develop debugging and verification capabilities to foster long-term knowledge internalization, and problem-solving abilities.

Previous literature has explored the impact of GAI applications across various learning domains, and analyzed students' understanding and application of GAI-generated content, revealing that students' preferences for GAI demonstrate two orientations: "Substitution" and "Augmentation" [2]. However, existing research predominantly focuses on single dimensions and lacks systematic convergence and synthesis to comprehensively examine changes in student learning behaviors and fundamental learning indicators under GAI assistance. In recent years, research has begun shifting focus from AI tool application patterns to the integration of GAI by students for programming, debugging, and optimization [3]. The role of GAI in programming education extends beyond being merely an auxiliary tool for knowledge transmission; it has become a critical factor influencing students' learning depth and subsequent application capabilities.

Therefore, this study employed a systematic literature review focusing on the application outcomes of generative AI in programming education from 2023 to 2025, integrating learning indicators and learning behavior changes throughout students' learning processes, and further analyzing its impact on code quality, such as structural clarity, readability, error rate, or modularity, as well as task completion efficiency. In particular, the study aimed to address the following research questions:

RQ1: In GAI-assisted learning contexts, which aspects of students' programming learning performance demonstrate significant impact?

RQ2: What specific effects do GAI-assisted learning have on the quality of code produced by students and their completion efficiency?

RQ3: What changes emerge in students' learning behaviors and teachers' instructional strategies during the implementation of GAI in educational applications?

This research aims to assist educators in achieving a balance between curriculum design and AI utilization, ensuring that GAI serves not merely as a technical support tool, but also promotes students' subsequent application capabilities and long-term knowledge consolidation through rigorous assessment and instructional strategies. This involves not only understanding the nature of GAI technology, but also strategic adjustments in curriculum design and instructional implementation within educational practice, which has significant implications for the sustainable development of higher education.

doi: 10.18178/ijiet.2026.16.1.2487

II. Materials and Methods

This study employs a systematic review methodology, supplemented by thematic analysis to organize qualitative dimensions, reviewing empirical research findings on GAI-assisted programming education. The study aims to examine its impact on student programming learning outcomes compared to traditional teaching approaches, including aspects such as programming skill performance, learning efficiency, and learning behaviors. To focus on the latest GAI developments in programming education, this study utilizes Semantic Scholar for literature retrieval. First, Semantic scholar's semantic search and automatic summary (TLDR) features facilitate the rapid comprehension of generative AI applications across various programming education contexts, enhancing the efficiency and precision of literature retrieval. Second, the platform not only encompasses over 200 million cross-disciplinary documents, but also provides comprehensive coverage of computer science, educational technology, and artificial intelligence-related fields, sufficiently supporting this study's requirements for thematic depth and breadth. Moreover, this database is free and unrestricted by institutional subscriptions, thus ensuring reproducibility of the research process. Therefore, this study selected the Semantic Scholar database and employed rigorous search strategies and screening criteria to ensure the representativeness and quality of the subsequently coded literature.

Since the application of generative AI technology in education largely began with the rapid proliferation of ChatGPT in 2023, and most relevant empirical studies were published after 2023, this study sets the literature search scope from January 1, 2023, to May 1, 2025, to encompass the critical period of GAI in programming education. Although Semantic Scholar's semantic search provides flexibility and contextual understanding advantages in exploring emerging topics, to comply with the Preferred Reporting Items for Systematic Reviews (PRISMA)'s emphasis on search strategy reproducibility and transparent reporting principles, this study still employs clearly defined keywords for initial screening to ensure transparency and traceability of the search process. The following keywords were used for result screening: ("generative AI") AND ("ChatGPT" OR "GitHub Copilot") AND (programming OR coding). The literature inclusion and exclusion criteria consisted of five items, as listed in Table 1.

Table 1. Literature inclusion criteria

Inclusion Criteria	Description
Educational Domain	Research must be conducted in programming education contexts.
Use of Generative AI Tools	Studies must explicitly utilize generative AI tools.
Empirical Research Design	Only empirical studies including experimental designs, quasi-experimental, and observational studies are included; literature reviews and meta-analyses are excluded.
Comparative Baseline Design	Studies must include control groups or pre-/post-test to verify differences between AI-assisted and traditional teaching effectiveness.
Learning Outcome Indicators	Studies must contain quantifiable student learning outcome data, such as test scores, project completion rates, and code quality.

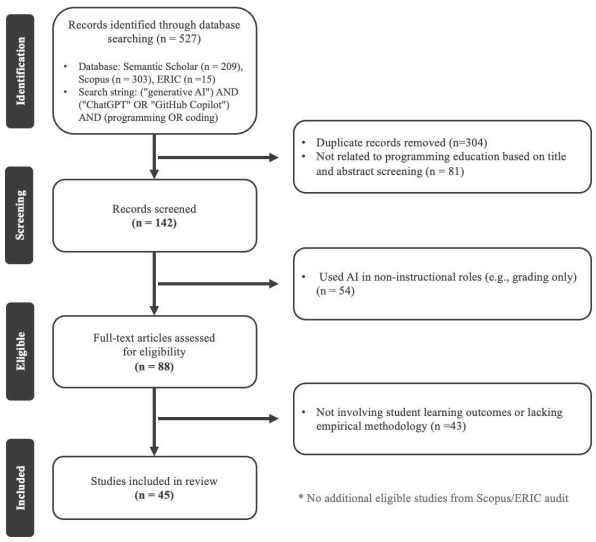

Fig. 1. PRISMA flow diagram.

Subsequently, a systematic review and meta-analysis was conducted following the PRISMA standards for data collection and literature screening. The search yielded 209 relevant documents. The data screening process, as shown in Fig. 1, involved excluding literature unrelated to programming education based on titles and abstracts ($n = 81$), followed by full-text reading to further exclude studies that did not use AI tools in educational application contexts ($n = 44$), and research lacking student learning outcomes or empirical data ($n = 39$). Ultimately, 45 articles were included in the content analysis.

A. Data Extraction and Coding Criteria

This study employed a pre-established coding framework aligned with the research objectives to conduct systematic data extraction from the literature that met the inclusion criteria, facilitating subsequent analysis and comparison. The coding categories are presented in Table 2.

Table 2. Literature coding description

Coding Category	Description
Research Design Type	Primarily includes Randomized Controlled Trial (RCT), Quasi-Experimental Design, Pre-/Post-Test Design, and Comparative Study
AI Teaching Intervention Methods	AI tool names, application methods (teaching assistant, debugging, practice support, etc.), frequency of use, and intervention duration
Learning Context	Includes participants' educational level and programming context
Learning Outcome Indicators	Such as programming performance scores, completion rates, problem-solving abilities, code quality, etc.
Measurement Instruments and Time Points	Scales and assessment methods utilized
Primary Research Results	Comparison of outcomes between groups, statistical significance

III. Results

A. Characteristics of Included Studies

This review included 45 empirical studies, as shown in Table 3. These studies met the pre-established coding criteria and demonstrated diverse research design types, including 13 Randomized Controlled Trials (RCTs) [4–16] and 21 Quasi-Experimental Designs [3, 17–36], Among these, 19 studies employed Pre-/Post-Test Designs [4, 9, 11–14, 17, 18, 21, 22, 24–27, 29, 31, 36–38] and 27 were Comparative Studies [3, 5–8, 10, 11, 15, 16, 19, 20, 23, 28, 30, 32–34, 36, 39–47]. Although some studies did not explicitly state their research design type, the overall results still reflected the methodological diversity and openness in this research domain.

All the studies confirmed the use of generative AI tools to align with the research theme. Most studies utilized ChatGPT [3–28, 30–45, 47], with only one study using Gemini 1.5 [29] and three studies employing the GitHub Copilot [28, 37, 46]. ChatGPT 3.5 was the most commonly used version, while two studies used both ChatGPT and GitHub Copilot [28, 37].

Regarding programming learning contexts, most studies have been conducted in educational settings, including introductory programming courses at universities (such as CS1) and various specialized domain courses. Among the studies, 11 did not explicitly specify the programming language used in their courses [3, 9, 10, 12, 17, 21, 34, 35, 41, 45, 47], 8 focused on introductory programming courses [7, 15, 31, 33, 38, 40, 44, 46], while the remaining studies covered various languages and course domains including C++ [8, 27], JAVA [5, 25, 37, 42], Data analysis [18, 24], JavaScript to web design [28, 32], Object Oriented Programming (OOP) [13, 43], PHP [19], Python [4, 11, 14, 20, 23, 30], Visual Basic [26], API test [22], Scratch [39], as well as Computer engineering [29], Software engineering [6, 16] and Vibration analysis course [36], demonstrating the application potential of generative AI tools across diverse educational scenarios.

The primary learning outcome indicators included programming knowledge acquisition, computational thinking abilities, problem-solving skills, programming self-efficacy, and code quality. However, these studies exhibited high heterogeneity in their assessment tools and observational dimensions, reflecting that "programming ability" in AI-assisted learning contexts is a multifaceted learning outcome difficult to measure with a single scale.

The synthesis of these results indicates current research's interest in "integrating generative AI into programming education," presenting its potential effectiveness through diverse research methods and learning scenarios.

Table 3. Literature screening results

Ref.	Study Design	Programming Context	AI Tool Used	Primary Outcomes
[25]	Quasi-Experimental Design; Experimental design with pre-test/post-test	Computer Science (JAVA)	ChatGPT 3.5	UML diagram creation, programming implementation, closed-book post-evaluation scores
[29]	Quasi-Experimental Design; Experimental design with pre-test/post-test	Computer engineering	Gemini 1.5	Academic performance, perception of usefulness and ease of use, satisfaction and motivation
[9]	Randomized controlled trial; Experimental design with pre-test/post-test	Programming course	ChatGPT	Computational thinking skills, programming self-efficacy
[44]	Comparative study	Introductory programming	ChatGPT 3.5	Time taken to complete tasks, number of tasks attempted, scores achieved
[43]	Comparative study	Object-Oriented Programming (OOP) course	ChatGPT	Performance data (test scores and grades)
[11]	Randomized controlled trial; Experimental design with pre-test/post-test	Python programming	ChatGPT	Code-authoring performance, code-modification performance, computational thinking
[3]	Quasi-experimental study; Comparative study	Programming course	ChatGPT	Programming performance, perceived usefulness, perceived ease of use, intention to use
[31]	Quasi-experimental study; Experimental design with pre-test/post-test	Python programming	ChatGPT 3.5	Python programming knowledge and skills
[45]	Comparative study	Programming course	ChatGPT 3.5	Computational thinking skills, creative thinking
[24]	Quasi-experimental study; Experimental with pre/post; Comparative study	Data Structures and Algorithms course	ChatGPT	Problem-solving skills, algorithmic thinking, ability to write executable code
[46]	Comparative study (within-subjects design)	Introductory programming	GitHub Copilot AIDE	Programming performance, self-efficacy
[41]	Comparative study	Programming course	ChatGPT	Programming knowledge acquisition, group-level programming product quality
[33]	Quasi-Experimental Design; Experimental design with pre-test/post-test	Introductory programming course	ChatGPT 4	Programming knowledge, performance analysis for different compiler errors
[23]	Quasi-Experimental Design; Comparative study	Python programming	ChatGPT	Programming proficiency, students' experiences using ChatGPT
[19]	Quasi-Experimental Design; Comparative study	PHP programming course	ChatGPT	Coding (Classical, True/False, Multiple Choice questions)
[42]	Comparative study	Introductory Java programming courses	ChatGPT	Adherence to coding conventions, cyclomatic complexity, cognitive complexity
[16]	Randomized controlled trial; Comparative study	Software engineering classroom	ChatGPT	Structure, independence, value, testability, grammar of user stories
[34]	Quasi-experimental study; Comparative study	Component Programming course	ChatGPT	Speed of task completion, depth of understanding, diversity of solutions, critical thinking
[17]	Quasi-experimental study with	Programming course	ChatGPT 3.5	Programming test scores

	pre-test/post-test			
[35]	Quasi-experimental study	Programming course	ChatGPT	Comprehension and application of programming concepts
[36]	Comparative study; Quasi-experimental study	Vibration analysis course for mechanical engineers	ChatGPT	Computational thinking, problem-solving abilities
[38]	Experimental design with pre-test/post-test	Introductory programming	ChatGPT	Normalized learning gain
[32]	Quasi-experimental study; Comparative study	JavaScript functions for Web Design and Coding students	ChatGPT	Navigation performance, test scores
[13]	Randomized controlled trial; Experimental with pre/post; Comparative study	Object-oriented programming concepts	ChatGPT 3.5	Programming performance, code submission success rate
[22]	Experimental with pre/post; Quasi-experimental	API testing (IT and IS students)	ChatGPT	Understanding and using APIs, confidence in using APIs
[26]	Quasi-experimental; Experimental with pre/post	Visual Basic 6 for undergraduates	ChatGPT	Programming skills in Visual Basic 6
[7]	Randomized controlled trial; Comparative study	Introductory programming	ChatGPT 4	Engagement, practice performance, posttest performance, scaffolding apply rate
[14]	Randomized controlled trial; Experimental with pre/post	Python programming	ChatGPT	Code-authoring and code-modification skills
[5]	Randomized controlled trial; Comparative study	Java programming course	ChatGPT 4	Laboratory assignment scores
[30]	Quasi-experimental study; Comparative study	Python programming	ChatGPT 4	Test scores, Python skills enhancement
[39]	Comparative study	Scratch programming for non-CS majors	ChatGPT	Syntax understanding, code writing ability, Scratch feature use, algorithm understanding
[20]	Quasi-experimental study; Comparative study	Python programming	ChatGPT 3	Problem-solving time and success rate
[18]	Quasi-experimental study with pre/post	Programming and data analysis	ChatGPT	Programming and data analysis skills
[4]	Randomized controlled trial; Experimental with pre/post	Python programming	ChatGPT 3.5	Python programming skills
[10]	Randomized controlled trial; Comparative study	Programming via debugging	ChatGPT	Programming performance, computational thinking
[21]	Quasi-experimental study with pre/post	Programming courses	ChatGPT	Programming skills, conceptual understanding, engagement, satisfaction
[40]	Comparative study	Introductory programming	ChatGPT 3.5	Code correctness, code quality
[6]	Randomized controlled trial; Comparative study	Software testing education	ChatGPT 3.5	Effectiveness and efficiency in writing tests
[12]	Randomized controlled trial; Experimental with pre/post	Debugging-focused programming education	ChatGPT 3.5	Comprehensive and accurate hypothesis construction
[47]	Comparative study	Programming assistance	ChatGPT	Code quality, task completion time
[37]	Experimental with pre/post; Interview analysis	Java programming course	ChatGPT	Pre/post-task quiz score, code quality, task completion time, student interviews
[8]	Randomized controlled trial; Comparative study	Object-Oriented Programming using C++	ChatGPT	Practical assignment performance, midterm exam results, final course grade, student feedback on ChatGPT usefulness
[27]	Quasi-experimental study with pre/post	C++ programming course	ChatGPT	Flow experience, self-efficacy, learning achievement
[28]	Quasi-experimental study; Comparative study	JavaScript to web design	ChatGPT, GitHub Copilot	Coding assignment, code generation, debugging, explanation
[15]	Randomized controlled trial; Comparative study	Introductory programming	ChatGPT 3.5	programming posttest score, self-efficacy in programming, task performance, programming errors

B. Effects of AI-Assisted Programming Learning

1) Learning performance metrics

This study provides an in-depth exploration of performance indicators across the 45 studies. Through the analysis of different indicator types and based on Table 3 in Section 3.1, the Primary Outcomes were consolidated into seven major learning performance indicators: programming knowledge/skills, computational thinking/logical reasoning, problem-solving ability, programming self-efficacy/confidence, learning achievement, code quality, and learning behaviors/engagement, with the distribution shown in Fig. 2.

All the studies addressed learning indicators related to programming knowledge/skills, demonstrating that students' understanding and application abilities in programming languages and concepts constitute the primary assessment focus. Learning achievement was examined in 41 studies [4–7, 9–22, 24–26, 28–47], represented through quantitative results such as test scores and academic performance. Learning behaviors and engagement were investigated in 38 studies [3, 4, 6–9, 11–15, 17, 18, 20–26, 29–45, 47], while computational thinking and logical reasoning abilities were addressed in 33 studies [3–5, 7, 9–11, 13, 14, 16–25, 27, 30, 32–34, 36, 38–45], focusing on students' abstraction and pattern recognition thinking processes during programming. Additionally, 29 studies examined students' abilities in applying strategies for debugging and problem-solving [3, 5–7, 10–15, 18, 20, 23–25, 27, 28, 31, 33, 34, 36, 37, 39, 41, 42, 44–47], while 21 addressed students' cognition and attitudes toward their own programming learning and application capabilities [3, 5, 8, 9, 14, 15, 19, 20, 22, 23, 27, 29, 31, 35, 37-39, 41, 44–46]. Finally, code quality was examined in 16 studies [3, 5, 8, 9, 14, 15, 19, 20, 22, 23, 27, 29, 31, 35, 37-39, 41, 44–46], emphasizing the structural quality and maintainability of code produced by the students.

This multidimensional categorization provides a more comprehensive presentation of students' multilevel outcomes in generative AI-assisted learning, and reflects the diversity of observational focus across different studies.

Further comparisons of the measurement methods and learning outcomes following generative AI tool intervention are presented in Table 4. Although some studies did not explicitly indicate the direction of intervention effects, most results demonstrated the positive impact of generative AI instruction on student learning performance. A total of 34 studies reported that AI-assisted groups outperformed traditional control groups across multiple dimensions, including learning achievement, problem-solving abilities, and self-efficacy. However, 11 studies showed significant negative differences or no improvement in average performance [5–8, 15, 25, 27–29, 41, 43], indicating that while generative AI tools demonstrate positive effects in most studies, variations and challenges persist.

Fig. 2. Seven major learning performance indicators.

Table 4. Learning performance indicator screening results

Ref.	Measurement Type	AI-Assisted Results	Traditional Results	Converged Outcome
[25]	UML diagram, programming implementation, post-evaluation scores	No significant impact	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Learning behaviors / Engagement
[29]	Academic performance	No statistically significant differences	No statistically significant differences	Prog. Knowledge/Skills, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[9]	Computational thinking skills, programming self-efficacy	Significantly higher	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[44]	Task completion time, scores	Faster completion, comparable scores	Slower completion, comparable scores	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors / Engagement
[43]	Test scores	Scores improved but the effect was average	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[11]	Code-authoring performance	1.15x increased completion rate, 1.8x higher scores	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[3]	Programming performance	M = 84.11, SD = 19.45	M = 78.36, SD = 17.59	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning behaviors / Engagement
[31]	Python programming knowledge and skills	Average increase of 12.50	Average decrease of 3.17	Prog. Knowledge/Skills, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[45]	Computational thinking skills	Enhanced	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors / Engagement
[24]	Problem-solving skills, algorithmic thinking	Significantly higher scores	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Learning behaviors / Engagement
[46]	Programming efficiency	Significantly increased	Lower than AI-assisted condition	Prog. Knowledge/Skills, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality
[41]	Programming knowledge acquisition	No substantial differences	No substantial differences	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors / Engagement
[33]	Programming knowledge, error troubleshooting	Improved error troubleshooting capabilities	Lower performance in error troubleshooting	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[23]	Programming proficiency	Statistically significant but not practically significant	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning behaviors / Engagement
[19]	Coding question performance	Statistically significant positive effect	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Prog. Self-Efficacy, Learning achievement
[42]	Code quality	Significantly improved	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[16]	User story quality	The results are higher on average	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement
[34]	Task completion speed	Nearly three times quicker	Slower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[17]	Programming test scores	Increased from 48.33 to 74.47	Lower than AI-assisted condition	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[35]	Comprehension and application of programming	Significantly higher	Lower than AI-assisted group	Prog. Knowledge/Skills, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors /

	concepts			Engagement
[36]	Quiz scores	Increased performance	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Learning behaviors / Engagement
[38]	Normalized learning gain	M = 52.85, SD = 24.27	M = 41.29, SD = 26.84	Prog. Knowledge/Skills, CT / Logic Reasoning, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[32]	Test scores	Significantly higher	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[13]	Successful submissions	Higher	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[22]	Exam test scores	Significantly higher (~80)	Lower (~50)	Prog. Knowledge/Skills, CT / Logic Reasoning, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors / Engagement
[26]	Programming skills in Visual Basic 6	M = 8.34, SD = 1.42	M = 7.56, SD = 1.84	Prog. Knowledge/Skills, Learning achievement, Learning behaviors / Engagement
[7]	Posttest performance	Higher average accuracy	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Learning behaviors / Engagement
[14]	Post-test evaluation scores	Average results improved	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[5]	Lab assignment scores	Lower than instructor feedback	Better than AI feedback	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality
[30]	Test scores	Improving test accuracy and learning	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[39]	Scratch understanding	Significantly higher	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[20]	Programming success rate	Increased	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Code quality, Learning behaviors / Engagement
[18]	Programming and data analysis skills	Improve accuracy	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement, Learning behaviors / Engagement
[4]	Python programming skills	2 more questions solved	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[10]	Programming performance	Better for 2nd & 3rd levels of AIGC	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Learning achievement
[21]	Programming skills	Better performance	Lower than AI-assisted group	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Learning behaviors / Engagement
[40]	Code correctness	Minimal changes in scores	No mention found	Prog. Knowledge/Skills, CT / Logic Reasoning, Learning achievement, Code quality, Learning behaviors / Engagement
[6]	Effectiveness in writing tests	Negative impact (8.6% fewer tests, 78% not useful)	Better than AI-assisted group	Prog. Knowledge/Skills, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[12]	Debugging skills	12% increase in pre-post scores	Lower than AI-assisted condition	Prog. Knowledge/Skills, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[47]	Code quality	Better for algorithmic tasks	Lower than AI-assisted group	Prog. Knowledge/Skills, Problem-solving ability, Learning achievement, Code quality, Learning behaviors / Engagement
[37]	Programming success rate	Effectively improve completion time	No mention found	Prog. Knowledge/Skills, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement
[8]	Programming test scores	Not significantly different	Not significantly different	Prog. Knowledge/Skills, Prog. Self-Efficacy, Learning behaviors / Engagement
[27]	Post-test evaluation scores and Questionnaire	Significant but non-enhancing effect	Better results than the experimental group	Prog. Knowledge/Skills, CT / Logic Reasoning, Problem-solving ability, Prog. Self-Efficacy
[28]	Programming test scores	Significant but non-enhancing effect	No mention found	Prog. Knowledge/Skills, Problem-solving ability, Learning achievement
[15]	Programming scores	High user satisfaction	Higher Programming Score	Prog. Knowledge/Skills, Problem-solving ability, Prog. Self-Efficacy, Learning achievement, Learning behaviors / Engagement

Note: Programming knowledge / skills (Prog. Knowledge/Skills); Computational thinking / Logical reasoning (CT / Logic Reasoning); Programming self-efficacy / confidence (Prog. Self-Efficacy)

2) Code quality outcomes

Among the 45 studies, 11 examined the potential impact of generative AI tools on the quality of code produced by students, with results presented in Table 5. The majority of studies suggested that AI assistance contributes to improving the quality of code produced by students, with eight studies indicating that AI-assisted groups significantly outperformed the control groups [11, 12, 33, 35, 40, 42, 45, 47], and four demonstrating statistical significance [12, 35, 42, 47] including two studies that found students in AI groups significantly exceeded traditional groups in terms of modularity or correctness [12, 40]. These studies

encompassed multiple dimensions such as coding conventions, code structure standardization, error-handling capabilities, and code logic clarity, demonstrating the potential of generative AI tools for programming language assistance.

However, three studies revealed potential concerns. One study reported that although AI-assisted code output was functional, it exhibited uniformity in strategy design and problem-solving approaches, potentially constraining students' development of diverse thinking and innovative solutions [34]. Another study indicated that AI assistance had negative impacts on code quality [41], while Mezzaro *et al.* (2024) found that in test case writing tasks, both the quality and quantity of tests produced by the AI group were significantly lower than those produced by the control group [6].

These studies employed various code quality indicators, most of which appeared in only single studies, creating challenges for cross-study comparisons. To more clearly present the observational focuses of different studies and address the second research question (RQ2) regarding the impact of generative AI tools on code quality, these indicators were further consolidated into three major dimensions: Code Structure & Quality (five studies [11, 35, 40, 42, 47]), Debugging & Error handling (three studies [6, 12, 33]), and Team-Level diversity & Collaborative outcomes (three studies [34, 41, 45]), as shown in Fig. 3. This classification helps systematically integrate the diverse observational results from various studies, and serves as a foundation for subsequent in-depth comparison and statistical analysis.

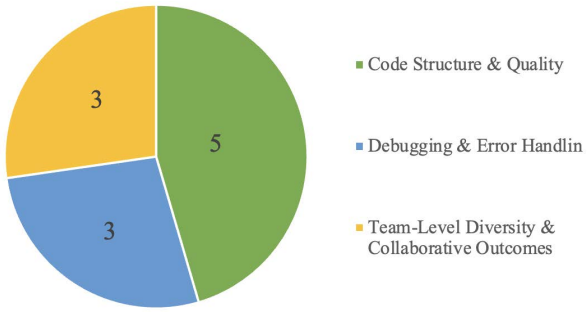

Fig. 3. Three major dimensions of code quality indicators.

Overall, while most studies indicate that AI assistance has positive impacts on code quality, the benefits are inconsistent, suggesting that in practical educational applications, a further balance is needed between technical assistance and students' development of programming comprehension.

Table 5. Code quality screening results

Ref.	Code Quality Metric	AI-Assisted Results	Traditional Results	Context Notes	Final Category
[42]	Adherence to coding conventions, cyclomatic complexity, cognitive complexity	Significantly improved	Lower than AI-assisted group	Introductory Java programming courses	Code Structure & Quality
[40]	Code modularity	Improved (Assignment-3 average modularity ratio: 1.49)	No mention found	Introductory programming Course	Code Structure & Quality
[47]	Code quality for algorithmic and library-related tasks	Significantly better	Lower than AI-assisted group	Programming assistance	Code Structure & Quality
[11]	Code-authoring performance	1.8x higher scores	Lower than AI-assisted group	Python programming	Code Structure & Quality
[33]	Error troubleshooting capabilities	Improved	Lower than AI-assisted group	Introductory programming course	Debugging & Error Handling
[34]	Diversity of solutions	Lower diversity, more uniform solutions	Higher diversity of solutions	Component programming course	Team-Level Diversity & Collaborative Outcomes
[45]	Quality of projects	Higher quality projects aligned with learning objectives	Lower than AI-assisted group	Programming course	Team-Level Diversity & Collaborative Outcomes
[41]	Group-level programming product quality	No substantial differences	No substantial differences	Programming course	Team-Level Diversity & Collaborative Outcomes
[35]	Application of programming concepts	Significantly higher performance	Lower than AI-assisted group	Programming course	Code Structure & Quality
[6]	Quality of written tests	Lower quality (78% not useful)	Higher than AI-assisted group	Software testing education	Debugging & Error Handling
[12]	Accuracy of hypothesis construction in debugging	15.8% improvement	Lower than AI-assisted condition	Programming education (debugging focus)	Debugging & Error Handling

3) *Time efficiency impacts*

In programming development learning contexts, beyond code quality, task completion efficiency represents an important dimension affecting student learning outcomes. Among the 45 empirical studies included in this analysis, eight examined the impact of generative AI tools on students' programming development time efficiency, with the results addressing RQ2, as shown in Table 6. Seven studies reported that generative AI-assisted programming learning enhanced task completion speed [12, 13, 20, 34, 44, 46, 47], whereas only one study indicated that traditional instruction demonstrated superior time performance [22]. These studies encompassed diverse learning contexts, including novice programming, introductory courses, component-based programming, object-oriented design concepts, API testing, general programming learning assistance, and debugging tasks.

Three studies provided detailed metrics for specific quantified time improvements. One study indicated that generative AI-assisted programming learning achieved nearly three times the speed of traditional methods [34], whereas Pankiewicz and Baker (2023) reported that the AI-assisted group reduced problem-solving time by an average of 375s (6.25 minutes) [13]. Meanwhile, Ma *et al.* (2024) demonstrated that in debugging tasks, the AI-assisted

group reduced completion time by 14% [12]. Only one study reported a better time through traditional instruction, where the AI group's average completion time was 49.95 minutes (SD = 17.32 minutes), while the traditional group completed tasks in 34.45 minutes (SD = 14.92 minutes) [22]. Additionally, the remaining four studies, while not providing explicit quantitative data, presented AI tools' time efficiency advantages through qualitative descriptions such as "significantly faster" or "reduced time" [20, 44, 46, 47].

These findings clearly indicate that generative AI tools have positive effects on time efficiency in programming learning, with this effect exhibiting consistency across different learning contexts and task types. This constitutes preliminary evidence regarding the efficiency benefits of AI tools in programming education.

Table 6. Time efficiency impact screening results

Ref.	Time Efficiency Metric	AI-Assisted Results	Traditional Results	Context Notes
[44]	Task completion time	Significantly faster (Group 1: 31.5 min, Group 2: 21.5 min)	Slower (Group 1: 59.8 min, Group 2: 38.9 min)	Introductory programming
[34]	Task completion speed	Nearly three times quicker on average	Slower than AI-assisted group	Component programming course
[46]	Programming efficiency	Significantly increased	Lower than AI-assisted condition	Significantly increased
[13]	Time to solve tasks	375 seconds (6.25 min) less on average	More time required than AI-assisted group	Object-oriented programming concepts
[22]	Time efficiency	Less efficient (mean time: 49.95 minutes, SD: 17.32 minutes)	More efficient (mean time: 34.45 minutes, SD: 14.92 minutes)	API testing for IT and IS students
[20]	Time to solve programming problems	Decreased	Higher than AI-assisted group	Python programming
[47]	Time taken to complete tasks	Faster for algorithmic challenges	Slower than AI-assisted group for algorithmic challenges	Programming assistance
[12]	Completion time for debugging tasks	14% reduction	Higher than AI-assisted condition	Programming education (debugging focus)

C. Thematic Analysis

Following the synthesis of multidimensional empirical results regarding learning outcomes, code quality, and time efficiency, this section further employs thematic analysis to explore changes in student learning behaviors under generative AI tool assistance as well as implementation considerations that teachers may face in instructional adjustments and curriculum design. By integrating quantitative results and observed learning trends from various studies, we conducted qualitative thematic synthesis from behavioral perspectives, systematically presenting student learning behavior characteristics in AI-assisted contexts, and exploring how educational settings can respond to the multifaceted challenges brought about by generative AI tools. This section is divided into two parts, the first focusing on changes in student learning behaviors, and the second examining reflections and recommendations regarding teachers' instructional implementation and curriculum adjustments, addressing the third research question (RQ3).

1) Student learning behaviors

Based on the empirical literature included in this study, the introduction of generative AI tools progressively reshapes student behavioral patterns in programming learning through several positive dimensions. First, regarding learning engagement and motivation, AI tools enhance student involvement and interest in learning activities. Related research indicates that students generally perceive AI tools as useful and convenient to operate, thus encouraging higher usage intentions [3]. Studies also found that compared to traditional instruction, AI-assisted learning provides greater appeal and engagement [7]. Second, AI tools have transformed students' problem-solving abilities, with some students tending to rely completely on AI-generated code, while others combine AI suggestions with their own coding for hybrid problem-solving [14]. Third, AI tools strengthen students' self-directed learning behaviors. Garg and Rajendran [18] found that structured prompts can stimulate students to actively engage in learning tasks, whereas Chang and Chien [30] 's AI-driven quiz platform observed that increased student interaction correlated with better test performance, indicating that AI tools help cultivate self-directed learning momentum. Fourth, AI tools provide immediate feedback regarding error handling and debugging abilities, effectively improving students' performance in syntax errors and debugging tasks. Pankiewicz and Baker [33] reported better student performances in error handling, and Ma *et al*. [12] indicated that pre-/post-test scores for debugging tasks improved by 12%.

However, alongside these enhancements, the literature simultaneously reveals adaptation challenges and shifts in collaborative dynamics requiring pedagogical attention. Fifth, research cautions against the risk of students' over-dependence on AI tools. Mezzaro *et al*. [6] observed that students who over-rely on AI show significant decreases in both the quantity and quality of test cases written. Lehmann *et al*. [4] also noted that if students habitually let AI handle all problems, it may inhibit their motivation and ability for active thinking and deep understanding. Even as research indicates that AI assistance improves problem-solving speed, it may lead to solution uniformity, thereby limiting the diversity of problem-solving strategies [34]. Sixth, most studies show that students demonstrate high learning adaptability to AI tools, quickly becoming familiar with and applying them to programming tasks, reflecting openness and acceptance of emerging learning technologies [25, 29]. Finally, AI tools also change collaborative interaction patterns. Research by Fan *et al*. [41] indicates that under AI assistance, the patterns and content of team programming discussions have transformed, suggesting that AI intervention may open new pathways for collaborative learning.

These results indicate that AI tools are comprehensively changing students' behavioral patterns in programming

learning, as shown in Table 7, including enhanced engagement, self-directed learning, and debugging abilities. Such changes also reflect shifts in students' self-regulation processes. However, careful consideration must be given to the potential risks of overdependence and solution homogenization to effectively balance efficiency and depth in curriculum design.

Table 7. Seven major learning behavior changes

Behavior	Description
Enhanced Engagement	Demonstrating interest and enthusiasm for learning activities
Problem-solving Ability	Adopting more effective and innovative methods to solve problems
Self-directed Learning	Exhibiting stronger learning ability without direct guidance
Error Handling and Debugging Ability	Becoming more proficient in identifying and correcting errors
Over-dependence	Over-relying on tools, potentially hindering independent thinking
Adaptability	Easily adapting to learning new tools and technologies
Collaborative Interaction	Demonstrating stronger communication and cooperation abilities in teamwork

2) *Pedagogical and implementation implications*

As generative AI tools become increasingly widespread in programming education, teachers face unprecedented challenges in instructional adjustment and implementation. According to the empirical literature included in this study, introducing AI-assisted instruction requires the consideration of multiple dimensions. Tool selection significantly affects learning effectiveness, whereas integration methods (such as virtual teaching assistants, quiz prompts, and flipped learning) determine the appropriateness of learning activities and student acceptance. Student ability levels also constitute an influencing factor [4]. Similarly, task complexity must be considered when determining the applicable scope of AI tools. AI is particularly effective for algorithmic and library application tasks [47], but may have limited effectiveness for innovative and conceptual tasks.

Facing these challenges, teachers' roles are transforming from traditional knowledge transmitter to AI learning facilitator and guide. Anishka *et al.* [40] explored the feasibility of using ChatGPT as a virtual teaching assistant, while Er *et al.* [5] indicated that teachers must adjust their feedback approaches to enhance complementarity with AI feedback. The integration of AI tools has become an important element in curriculum design, from flipped teaching strategies to learning support tools [36, 38]. Chang and Chien [30] utilized AI-driven platforms to provide personalized learning materials. Additionally, AI literacy should be incorporated into curriculum design to help students effectively operate and understand AI tools [4, 14].

The research has demonstrated innovative directions regarding assessment mechanisms and curriculum content. Applying AI-generated prompts to help students clarify compilation errors or developing AI-assisted programming assessment mechanisms demonstrates that AI tools are creating new possibilities in assessment design [17, 33]. Furthermore, curriculum design needs to adjust teaching strategies according to different programming domains. AI tools have varying effects on introductory courses, component-oriented learning, debugging training, and advanced topics, demonstrating the need for adaptability in teaching contexts [16, 25]. Research indicates that curricula should include discussions on ethical issues related to AI use [23, 25] to establish students' responsible attitudes and values toward technology.

In summary, Fig. 4 illustrates the three key dimensions of GAI tool-assisted learning. The introduction of GAI tools has driven programming education toward more personalized, autonomous, and strategic directions. However, without comprehensive instructional design and implementation planning, the potential benefits of AI tools will be difficult to realize fully and may even negatively impact learning quality.

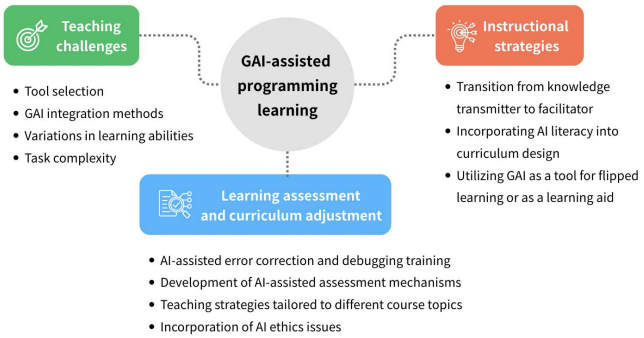

Fig. 4. Key dimensions of GAI tool-assisted learning.

IV. Discussion

A. *Impact of Generative AI on Students' Programming Learning Performance*

This section addresses RQ1 based on the synthesis presented in Fig. 2 and Table 4. Generative AI-assisted learning demonstrates complex and differentiated impacts on students' programming learning outcomes, encompassing seven major learning indicators: programming knowledge and skills, computational thinking, problem-solving ability, self-efficacy, learning achievement, code quality, and learning behaviors and engagement.

For fundamental programming knowledge and skills, the results consistently indicate positive effects. Kazemitabaar *et al.* [11] found that through code output provided by generative AI, beginners achieved significant improvements in both task completion rates and correctness. This was accomplished without weakening their subsequent ability to manually modify code, and instead promoted mastery of basic programming development skills. The immediate feedback mechanism of generative AI enables students to locate and correct errors more quickly, subsequently leading to better learning outcomes and higher-quality programming assignments in post-course assessments [3, 48]. Similarly, Sun *et al.*'s [3] reported significantly lower code error rates in the AI-assisted group. Collectively, GAI facilitates early syntax acquisition, strengthens learner confidence, and reduces novice frustration [9].

Higher-order outcomes diverge. In a quasi-experimental CS1 study, Xue *et al.* [25] observed no significant differences in final programming scores between ChatGPT and control groups, consistent with Jayagopal *et al.* [48], who found faster completion but no superior knowledge mastery. From a

cognitive-load perspective, AI can reduce extraneous cognitive load. As evidenced by the decreased debugging time observed in Table 6, this assistance may reduce extrinsic cognitive load by offering quick solutions and scaffolding support. However, for more complex or unfamiliar tasks, such convenience may increase intrinsic cognitive load, as students might lack sufficient foundational understanding to evaluate or modify AI-generated responses. Furthermore, the overly convenient access to answers may discourage active problem-solving and critical thinking, thereby increasing students' dependency on AI tools. Ready-made answers can also dampen active problem solving, fostering over-dependence without proper guidance [31]. For complex algorithmic work, AI output quality may degrade, risking misleading learning paths [49].

Effects are moderated by prior ability, task features, and context. Students with weaker foundations may accept AI outputs without verification [50], whereas stronger students use AI to refine strategies. AI is reliably helpful in structured, lower-complexity exercises, but benefits weaken for tasks requiring innovation, intricate logic, or algorithm design. Tool/version differences, integration approaches, and instructional settings further shape durability of gains.

GAI's convenience can also influence learning autonomy and academic integrity. Lyu *et al*. [31] found that while short-term performance improved, long-term retention and deep understanding of fundamental programming concepts were limited when students bypassed reasoning and verification. Furthermore, the performance and correctness of AI-generated responses vary across programming task types [6], which may introduce bias and compromise assessment fairness. In terms of equity, disparities in access to AI tools stemming from limited institutional support or restricted availability can exacerbate learning inequalities. As Jayagopal *et al*. [48] suggest, future research should examine how such access gaps impact learning processes and outcomes in low-resource contexts, while also addressing potential integrity risks [51, 52].

Synthesizing the above empirical findings, GAI in programming education presents a dual profile: consistent benefits in foundational skill acquisition and short-term learning efficiency, alongside potential risks to higher-order thinking and long-term knowledge construction. Accordingly, curricula should integrate AI tools strategically, leveraging them to lower entry barriers, enhance motivation, and build confidence, while incorporating verification routines, staged scaffolding, and explicit reflection to sustain active reasoning and deep learning. The differentiated performance of these seven learning indicators provides an important empirical foundation and strategic directions for the appropriate application of AI tools in programming education.

B. Impact of Generative AI on Code Quality and Completion Time

This section addresses RQ2 by examining the mechanisms and pedagogical implications of generative AI (GAI) on students' programming output quality and learning efficiency. Drawing on the empirical findings in Table 5 and Table 6, the analysis focuses on three dimensions: Code Structure & Quality, Debugging & Error Handling, and Team-Level Diversity & Collaborative Outcomes, as well as their broader implications for programming education The distribution among these themes are illustrated in Fig. 3.

Generative AI's impact on code quality demonstrates distinct hierarchical characteristics. At the level of code structure and syntactic correctness, AI tools show significant supportive effects, yet this improvement implies fundamental transformations in learning patterns [3, 11, 35, 40, 42, 47]. Instant availability of suggestions and refactoring hints can reduce opportunities for iterative trial-and-error, limiting the development of intuitive understanding of error causes and corrective logic. For less-prepared novices, this may result in syntactically correct code without deep comprehension or verification—an illusion of competence [33, 40, 42, 50]. In contrast, students with stronger foundations tend to integrate AI as a scaffold to refine strategies, indicating that benefits are uneven and mediated by prior knowledge and metacognitive skills.

Regarding debugging ability development, generative AI presents dual impact effects. On one hand, immediate error diagnosis and correction suggestions can accelerate problem-solving processes and reduce learning frustration [11]; on the other hand, over-reliance on AI diagnosis may weaken students' motivation and ability to establish independent debugging strategies. The debugging process holds unique educational value in programming learning, not only in training logical reasoning abilities, but also in cultivating systematic thinking and persistent solution exploration when facing complex problems. When this process is simplified or replaced by AI tools, students may lose opportunities to develop these core abilities, potentially affecting the depth development of their programming expertise in the long term.

The impact on team collaboration is more complex and carries important pedagogical implications. Generative AI can indeed balance ability gaps among team members, enabling learners of different levels to actively participate in collaborative tasks [31]. However, this balancing effect may obscure important educational values of collaborative learning. Traditional programming team collaboration emphasizes the promotion of learning through peer discussion, knowledge sharing, and collaborative debugging, whereas AI tool interventions may reduce these interaction opportunities. When team members can quickly obtain AI assistance, their interdependence and knowledge exchange needs may be reduced, thereby affecting the depth and effectiveness of collaborative learning.

Although a shortened task completion time presents superficial positive benefits, it requires deep examination from the perspective of balancing learning efficiency and effectiveness [13, 34, 44]. Time efficiency improvements may stem from two different mechanisms: first, genuine enhancement of learning abilities, enabling students to complete tasks more quickly and accurately; second, dependence on AI tools, shortening completion time through external assistance without necessarily enhancing internal capabilities. Distinguishing between these is crucial in evaluating the educational value of AI tools. Reverse cases appearing in research, such as the phenomenon of AI-assisted groups taking longer in API testing courses, provide important opportunities for reflection [22]. This phenomenon may reflect AI tool limitations in specific task types, or

increased cognitive load when students integrate AI suggestions with task requirements.

Behavioral pattern analysis reveals deeper transformations in learning strategies. Students' frequent cycling between AI consultation and code modification demonstrates an emerging "AI-dependent" problem-solving strategy [3]. While this strategy can effectively solve technical problems in the short term, it may cultivate passive learning attitudes and problem-solving habits [52, 53]. Students may gradually lose confidence and the ability to face challenges independently, becoming over-dependent on external tools. This poses potential threats to the long-term goals of programming education.

From an instructional design perspective, these findings point to a critical balancing challenge: how to leverage AI tools to enhance learning efficiency while ensuring that students still develop the necessary core competencies. Assessments should capture not only final outputs but also reasoning processes and strategies. Curricula can adopt staged AI support adjusting assistance by learning stage to preserve autonomous skills while still benefiting from AI-based guidance.

C. Changes in Student Learning Behaviors and Teacher Instructional Strategies under Generative AI Implementation

This section addresses RQ3, exploring how GAI adoption reshapes student behaviors and instructional strategies. Thematic analysis shows that GAI transforms both cognitive processes and pedagogical approaches, as discussed in the following two subsections.

1) Deep mechanisms and educational implications of student learning behavior changes

The introduction of generative AI tools is fundamentally restructuring students' cognitive processing patterns and learning strategy choices. From the perspective of cognitive load management, generative AI tools create a phenomenon of "cognitive outsourcing," enabling learners to transfer cognitive resources from basic technical operations to higher-order thinking levels of conceptual understanding and problem solving. This transfer mechanism explains the fundamental reasons for the enhanced engagement and strengthened self-directed learning behaviors. When the extrinsic cognitive load is reduced through AI-assistance, learners gain more cognitive space for intrinsic processing, thereby demonstrating higher learning investment and exploratory willingness [54]. However, this mechanism simultaneously implies risks, as excessive cognitive outsourcing may lead to "deskilling" of fundamental abilities, with learners potentially losing the capacity to independently construct solution pathways while enjoying efficient problem-solving.

Self-regulated learning theory provides an important perspective for understanding changes in learning strategies [55]. In traditional programming learning, self-efficacy and regulatory strategies develop through repeated practice and error correction. Under AI assistance, however, learners often transition from generating their own feedback to relying on external, immediate feedback [30]. While this change can boost debugging efficiency and task completion speed, it risks weakening metacognitive capacities including self-monitoring, strategy selection, and evaluation of outcomes when learners grow accustomed to AI's precise diagnoses and solutions.

Problem-solving strategies show a parallel divergence. Learners with strong foundations tend to adopt hybrid problem-solving, blending AI suggestions with personal judgment to enhance cognitive performance; those with weaker foundations often use substitutive problem-solving, accepting AI outputs without verification [14]. This differentiation phenomenon reveals the Matthew effect in the educational applications of AI tools, where existing learning ability gaps may widen further owing to different tool usage strategies.

Transformations in collaborative interaction patterns carry profound social-cognitive implications. AI tools' intervention as "third-party cognitive partners" changes knowledge sharing and mutual assistance patterns in traditional peer learning. When each learner can obtain immediate expert-level assistance, interdependence in team collaboration may weaken, and cognitive conflicts and negotiation processes between learners may become diluted, which are crucial for deep learning. This change may lead collaborative learning to shift from "interdependent learning" to "parallel learning," affecting team problem-solving abilities and collective intelligence construction.

The rapid development of adaptive learning behaviors reflects digital natives' high acceptance of emerging technologies, while also exposing potential conflicts between technological adaptation and learning depth. When learners master AI tools quickly yet lack awareness of their limitations, overconfidence may lead to uncritical acceptance of outputs, undermining accuracy and conceptual understanding.

The root of the overdependence phenomenon lies in cognitive preference shifts between immediate and delayed gratification. Programming learning traditionally requires learners to tolerate errors and frustration, building resilience and problem-solving patience through continuous trial and error. AI tools' provision of immediate answers satisfies learners' desire for quick success, but may weaken their persistence and willingness for deep exploration when facing complex challenges [4]. This behavioral pattern transformation may affect learners' cognitive resilience development, resulting in lower adaptability when faced with complex situations that AI tools cannot handle.

The educational implications of these behavioral changes point to the need to reconstruct programming education paradigms. Educators need to reconsider skill cultivation priorities, shifting from mere programming technique mastery to comprehensive development of critical thinking, tool evaluation abilities, and human-AI collaboration skills. Curriculum design should incorporate explicit "AI literacy" cultivation objectives, helping learners establish accurate recognition of tool capabilities and limitations, and developing abilities to flexibly switch between AI assistance and independent thinking. Simultaneously, assessment mechanisms need to shift from outcome-oriented to process-oriented approaches, emphasizing learners' thinking processes, strategy selection, and reflective abilities, ensuring effective assessment and promotion of deep learning in the era of widespread AI tool adoption.

2) Theoretical foundations and professional development implications of teacher pedagogical strategy changes

Based on the teacher-strategy adjustment patterns identified in this study, generative AI tools drive teachers from passive adaptation to proactive innovation in their professional development trajectories. This transformation reflects a profound shift in educational paradigms, from traditional knowledge transmission models to collaborative learning facilitation models, with transformation mechanisms and professional restructuring requiring in-depth analysis.

Generative AI is redefining teachers' roles from knowledge transmitters to AI learning facilitators. The adoption of ChatGPT as a virtual teaching assistant reflects not only technology integration but also a deeper need to reposition professional value. As Er *et al*. [5] note, adapting feedback to complement AI marks a shift from monopolizing knowledge to coordinating the learning ecosystem.

Curriculum design is shifting from content-oriented to competency-oriented education. Chang and Chien [30] showed that AI-driven platforms can deliver personalized materials, requiring teachers to develop advanced curriculum architecture skills and adjust strategies across programming domains from introductory topics to advanced debugging and component-oriented learning.

Assessment is shifting from measurement to learning support, driven by the widespread adoption of AI tools. Practices such as AI-generated prompts, process-oriented evaluation, and authentic, complex tasks replace rote recall. This shift demands that teachers develop cross-disciplinary skills in technology and educational assessment to design diverse evidence-gathering systems and deliver real-time feedback.

Empirical findings on ethical education integration reflect the expansion of teachers' professional responsibilities. As students gain easy access to AI-generated programming solutions, academic integrity and responsible technology use become essential teaching responsibilities. This requires teachers to pair technical expertise with ethical judgment and value-education skills, ensuring a balance between technological convenience and academic rigor.

The empirical manifestations of professional development challenges reveal the diverse characteristics of teacher learning needs. Teachers face urgent needs for AI literacy enhancement and understanding AI tool operational logic and applicable contexts to avoid losing agency in teaching settings. This technical literacy requirement aligns with Güner's [56] research findings, emphasizing that helping students understand how to effectively use AI tools is more important than whether they can use them; similarly, teachers also need to develop critical application abilities for AI tools. Research indicating teachers' necessity to carefully evaluate AI tool applicability in higher-order thinking or innovative application topics actually reflects the increased importance of professional judgment abilities in the AI era.

The empirical patterns of adaptive teaching strategy development demonstrate the importance of teachers' professional resilience. Research finds teachers designing differentiated strategies based on curriculum objectives, students' foundational abilities, and task complexity, combining diverse assessment mechanisms and learning activities to achieve a dynamic balance between AI convenience and deep learning cultivation. The development of this adaptive capability requires teachers to possess stronger contextual sensitivity and strategic flexibility, in order to maintain teaching effectiveness and professional stability in rapidly changing technological environments.

Synthesizing this study's empirical findings, changes in teachers' pedagogical strategies point to fundamental innovation needs in professional development models. Future teacher education should emphasize comprehensive cultivation of technology integration abilities, ethical judgment literacy, and adaptive instructional design capabilities, helping teachers maintain irreplaceable professional value in the era of widespread AI tools, and achieving organic integration of technological enhancement and humanistic care.

V. Conclusion

Through a systematic literature review of 45 empirical studies, this paper provides an in-depth exploration of the application effects and impact mechanisms of generative AI tools in programming education. The findings reveal that GAI-assisted instruction presents complex and differentiated impact patterns in programming learning, with effects influenced by multiple factors, including learners' foundational abilities, task characteristics, and implementation contexts. Given the diverse findings and complex impact mechanisms presented in existing research, it is necessary to establish an integrated theoretical framework to systematically understand these interactive relationships. Based on this important research requirement, this study constructed an integrated conceptual model for GAI-assisted programming education, as shown in Fig. 5.

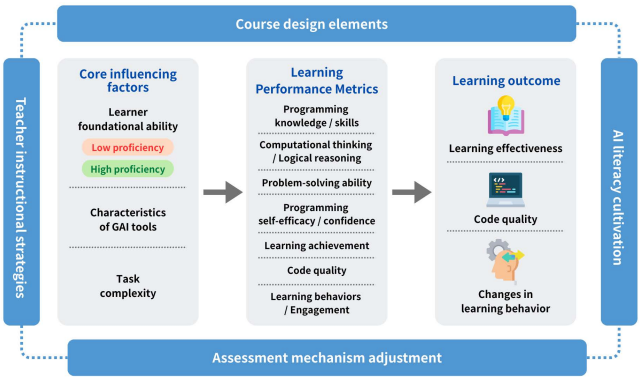

Fig. 5. Integrated conceptual model for GAI-assisted programming education.

This model presents a multidimensional integration framework for generative AI tools in programming education, systematically integrating four core dimensions–implementation context factors, core influencing factors, learning performance indicators, and learning outcome dimensions. As such, it provides a comprehensive analytical framework for understanding the complex impact mechanisms of GAI in programming education. The implementation context factors dimension, located at the model's periphery, encompasses educational environmental factors including teacher instructional strategies, curriculum design elements, assessment mechanism adjustments, and AI literacy cultivation. The core influencing factors dimension focuses on three key impact factors: learners' foundational abilities, GAI tool characteristics, and task complexity. The

learning performance indicators dimension establishes seven assessment aspects synthesized in this study, including programming knowledge and skills, computational thinking and logical reasoning, problem-solving ability, programming self-efficacy, learning achievement, code quality, and learning behaviors and engagement. The learning outcome dimension integrates three output performances: learning effectiveness, code quality, and learning behavior changes.

Regarding learning behavior changes, this study identifies seven major transformation aspects. Enhanced engagement is reflected in student perceptions of the utility and operational convenience of AI tools, along with higher usage intentions and learning investment. Problem-solving strategies show differentiation phenomena, with some students flexibly combining AI feedback with personal thinking to leverage human-AI collaboration advantages, whereas students with weaker foundational abilities may lack verification and thinking processes. Strengthened self-directed learning behaviors are manifested in structured prompts stimulating students to actively engage in learning tasks, with students who frequently interact with AI demonstrating higher efficiency in test performance. Improvements in error handling and debugging abilities stem from the immediate feedback provided by AI tools, effectively improving students' performance in syntax errors and debugging tasks. Adaptive learning behaviors show that students can quickly familiarize themselves with and apply AI tools to programming tasks. Changes in collaborative interaction patterns reflect AI tools' intervention as third-party cognitive partners, altering knowledge-sharing patterns in traditional peer learning. However, over-dependence risks require equal attention, with research indicating that students who over-rely on AI show significant decreases in both the quantity and quality of test cases written.

Adjustments in teachers' instructional strategies present transformation patterns from passive adaptation to proactive innovation. Research shows teachers beginning to adopt diversified teaching modes, introducing blended learning and flipped classrooms, and actively developing prompt-based assessment tools and personalized learning paths. The intervention of generative AI prompts teachers to redefine classroom roles, transforming from traditional knowledge transmitters to guides and facilitators who assist students in collaborative learning with AI tools. AI tool integration has become an important element in curriculum design, ranging from flipped teaching strategies to the integrated application of learning support tools. Innovations in assessment mechanisms include the use of AI-generated prompts to help students clarify compilation errors, and the development of AI-assisted programming assessment mechanisms. These findings provide concrete guidance for programming education practice. When implementing GAI-assisted instruction, educators should use the seven learning indicators synthesized in this study as foundations for assessing knowledge mastery and learning behavior transformation. Curriculum design and assessment methods should combine diverse teaching strategies such as flipped classrooms, project-based learning, and peer assessment, balancing the convenience provided by AI technology with the development of students' autonomous learning and thinking. The conceptual model presented in Fig. 5 provides teachers with a systematic reference framework, assisting them in making differentiated instructional strategy adjustments based on students' foundational abilities, achieving balanced development among learning convenience, problem-solving abilities, and long-term knowledge internalization.

Through multidimensional systematic integration of generative AI's application effects and challenges in programming learning, this study proposes seven learning indicators, three code quality indicators, and seven learning behaviors as assessment foundations for curriculum design and teaching practice. Teachers can select appropriate indicator dimensions for instructional design based on curriculum objectives and student characteristics. Fig. 5 further illustrates how teachers can make differentiated instructional strategy adjustments based on students' foundational ability differences, using the seven learning indicators as a foundation to assist in achieving balanced development among learning convenience, problem-solving abilities, and long-term knowledge internalization.

In a semester-long programming course, teacher instructional strategies in the initial phase focus on building a shared foundation of programming syntax, logical reasoning, and basic problem-solving skills without the use of AI tools. This approach ensures that students, regardless of their initial proficiency level, acquire the necessary competencies before AI integration. Once these foundational abilities are established, AI tools are progressively introduced, accompanied by explicit prompt-design guidance to help students explore the characteristics and affordances of generative AI in controlled contexts. Task complexity is gradually increased, prompting learners to apply computational thinking, logical reasoning, and problem-solving strategies while critically evaluating AI-generated outputs. The assessment mechanism incorporates self-regulation prompts, peer review, and iterative feedback cycles, encouraging students to reflect on AI responses, compare alternative solutions, and articulate the underlying logic of each code segment. Such an approach fosters a balanced development across learning performance metrics—such as programming self-efficacy, engagement, and code quality—and supports long-term learning outcomes, including enhanced AI literacy and sustainable learning behaviors.

Despite the systematic organization and analysis of relevant literature, this study has several limitations. The study focuses on rapidly developing short-term applications in recent years, with long-term learning knowledge applications and sustained effectiveness requiring further investigation in subsequent research. The applicability of generative AI tools and learning behavior differences across different languages and diverse task contexts also warrant deeper empirical comparison. Research samples are concentrated in Western higher education systems, limiting the cultural representativeness of findings. Although a few studies have emerged from Asian or African contexts, comparative empirical evidence remains scarce. More systematic cross-cultural research is needed to uncover culturally specific adoption patterns, pedagogical preferences, and behavioral responses to GAI tools. Additionally, 42 out of the 45 reviewed studies focused specifically on ChatGPT,

highlighting a tool-specific bias in current research. This dominance may limit the generalizability of findings to other generative AI tools with distinct interfaces, functionalities, or integration mechanisms, and also contributes to the overall heterogeneity in assessment approaches and observational dimensions, further complicating meaningful cross-study comparisons.

Although additional searches were conducted in Scopus and ERIC, no further eligible studies were identified after duplicate removal and relevance screening. The final corpus therefore relied primarily on Semantic Scholar, which provided broader coverage of emergent GAI keywords and ensured replicability through its open-access infrastructure. In total, this review synthesized 45 empirical studies. However, a full quantitative meta-analysis was not feasible. Despite all studies adopting experimental or quasi-experimental designs with treatment and control groups, 19 did not report sufficient statistical details (e.g., means, standard deviations, or test values) required for effect size estimation, and some reported only average differences, percentages, or project scores. Consequently, this study adopted a systematic literature review to capture broader learning patterns, identifying seven major learning indicators, three dimensions of code quality, and seven categories of learning behavior change, instead of producing aggregated quantitative effect sizes through a meta-analysis.

Future research should broaden database coverage and assess the potential impact of including additional sources on the comprehensiveness of the evidence base, while also developing conceptual models centered on students' foundational abilities by analyzing the interactive effects of AI tool use and student learning outcomes. Establishing standardized measurement tools and assessment systems for GAI-assisted learning effects would provide a reliable foundation for cross-study comparisons. While current studies predominantly report on short-term learning improvements, there is limited understanding of whether these benefits persist over time or translate into long-term programming proficiency. Future longitudinal research is necessary to assess retention effects, transferability to advanced tasks, and whether AI-assisted learning fosters durable cognitive change. Cross-cultural and cross-educational system comparative research would help understand the universal principles and context-specific factors in GAI educational applications.

Overall, generative AI tools provide multifaceted support and a transformative potential for programming learning in higher education. Educators can understand and apply the GAI technology more systematically through the integrated conceptual model developed in this study. Cultivating students' critical thinking, autonomous learning, and long-term knowledge internalization, while promoting learning convenience, remains a core issue that requires joint attention from educational practice and research. Through continued research and educational applications, generative AI tools can exert a profound influence on learning effectiveness and innovation stimulation.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, T.-C. Huang and H.-P. Tseng; methodology, T.-C. Huang and H.-P. Tseng; software, H.-P. Tseng; formal analysis, T.-C. Huang and H.-P. Tseng; investigation, H.-P. Tseng; resources, T.-C. Huang; data curation, H.-P. Tseng; writing—original draft preparation, T.-C. Huang; writing—review and editing, T.-C. Huang and H.-P. Tseng; visualization, H.-P. Tseng; supervision, T.-C. Huang; project administration, T.-C. Huang and H.-P. Tseng; funding acquisition, T.-C. Huang. All authors had approved the final version.

Funding

This research was funded by Ministry of Science and Technology, Taiwan, grant number [NSTC 112-2410-H-025-027-MY3].

References

[1] K. Nikolopoulou, "Generative artificial intelligence and sustainable higher education: Mapping the potential," *Journal of Digital Educational Technology,* vol. 5, no. 1, ep2506, 2025.

[2] P. Wang, Y. Jing, and S. Shen, "A systematic literature review on the application of Generative Artificial Intelligence (GAI) in teaching within higher education: Instructional contexts, process, and strategies," *The Internet and Higher Education,* 100996, 2025.

[3] D. Sun, A. Boudouaia, C. Zhu, and Y. Li, "Would ChatGPT-facilitated programming mode impact college students' programming behaviors, performances, and perceptions? An empirical study," *International Journal of Educational Technology in Higher Education,* vol. 21, no. 1, p. 14, 2024.

[4] M. Lehmann, P. B. Cornelius, and F. J. Sting, *AI Meets the Classroom: When do Large Language Models Harm Learning?* arXiv preprint arXiv:2409.09047, 2025.

[5] E. Er, G. Akçapınar, A. Bayazıt, O. Noroozi, and S. K. Banihashem, "Assessing student perceptions and use of instructor versus AI-generated feedback," *British Journal of Educational Technology,* vol. 56, no. 3, pp. 1074–1091, 2025.

[6] S. Mezzaro, A. Gambi, and G. Fraser, "An empirical study on how large language models impact software testing learning," in *Proc. the 28th International Conference on Evaluation and Assessment in Software Engineering*, 2024, pp. 555–564.

[7] X. Hou, Z. Wu, X. Wang, and B. J. Ericson, "Codetailor: Llm-powered personalized parsons puzzles for engaging support while learning programming," in *Proc. the Eleventh ACM Conference on Learning@ Scale*, 2024, pp. 51–62.

[8] T. Kosar, D. Ostojić, Y. D. Liu, and M. Mernik, "Computer science education in chatgpt era: Experiences from an experiment in a programming course for novice programmers," *Mathematics,* vol. 12, no. 5, p. 629, 2024.

[9] R. Yilmaz and F. G. K. Yilmaz, "The effect of generative Artificial Intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation," *Computers and Education: Artificial Intelligence,* vol. 4, 100147, 2023.

[10] S. Shanshan and G. Sen, "Empowering learners with AI-generated content for programming learning and computational thinking: The lens of extended effective use theory," *Journal of Computer Assisted Learning,* vol. 40, no. 4, pp. 1941–1958, 2024.

[11] M. Kazemitabaar, X. Hou, A. Henley, B. J. Ericson, D. Weintrop, and T. Grossman, "How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment," in *Proc. the 23rd Koli calling international conference on computing education research*, 2023, pp. 1–12.

[12] Q. Ma, H. Shen, K. Koedinger, and S. T. Wu, "How to teach programming in the ai era? using llms as a teachable agent for debugging," in *Proc. International Conference on Artificial Intelligence in Education*, 2024: Springer, pp. 265–279.

[13] M. Pankiewicz and R. S. Baker, *Large Language Models (GPT) for Automating Feedback on Programming Assignments*, arXiv preprint arXiv:2307.00150, 2023.

[14] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI code generators on supporting novice learners in introductory programming," in *Proc. the*

2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1–23.

[15] D. M. Johnson, W. Doss, and C. M. Estepp, "Using ChatGPT with novice Arduino programmers: Effects on performance, interest, self-efficacy, and programming ability," *Journal of Research in Technical Careers,* vol. 8, no. 1, p. 1, 2024.

[16] A. Brockenbrough and D. Salinas, "Using generative AI to create user stories in the software engineering classroom," in *Proc. 2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, 2024, IEEE, pp. 1–5.

[17] G. Akçapınar and E. Sidan, "AI chatbots in programming education: guiding success or encouraging plagiarism," *Discover Artificial Intelligence,* vol. 4, no. 1, p. 87, 2024.

[18] A. Garg and R. Rajendran, "Analyzing the role of generative AI in fostering self-directed learning through structured prompt engineering," in *Proc. International Conference on Intelligent Tutoring Systems*, Springer, 2024, pp. 232–243.

[19] J. Al Hajj and M. Sah, "Assessing the impact of ChatGPT in a PHP programming course," in *Proc. 2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS)*, IEEE, 2023, pp. 1–10.

[20] N. Torres, "A reverse code completion approach for enhancing novice programming skills," in *Proc. 2024 43rd International Conference of the Chilean Computer Science Society (SCCC)*, IEEE, 2024, pp. 1–8.

[21] O. L. D. Santos and D. Cury, "Challenging the confirmation bias: Using ChatGPT as a virtual peer for peer instruction in computer programming education," in *Proc. 2023 IEEE Frontiers in Education Conference (FIE)*, IEEE, 2023, pp. 1–7.

[22] Y. D. Setiawan, L. G. O. P. Yudha, Y. A. Mulyono, V. M. A. Simalango, and O. Karnalim, "ChatGPT impact analysis on API testing: A controlled experiment," *Journal of Applied Informatics and Computing,* vol. 8, no. 2, pp. 350–357, 2024.

[23] J. B. Jalon Jr, G. A. Chua, and M. de Luna Torres, "ChatGPT as a Learning Assistant: Its Impact on Students Learning and Experiences," *International Journal of Education in Mathematics, Science and Technology,* vol. 12, no. 6, pp. 1603–1619, 2024.

[24] B. Qureshi, "ChatGPT in computer science curriculum assessment: An analysis of its successes and shortcomings," in *Proc. the 2023 9th International Conference on e-Society, e-Learning and e-Technologies*, 2023, pp. 7–13.

[25] Y. Xue, H. Chen, G. R. Bai, R. Tairas, and Y. Huang, "Does chatgpt help with introductory programming? An experiment of students using chatgpt in cs1," in *Proc. the 46th International Conference on Software Engineering: Software Engineering Education and Training*, 2024, pp. 331–341.

[26] A.-M. M. Gasaymeh and R. M. AlMohtadi, "The effect of Flipped Interactive Learning (FIL) based on ChatGPT on students' skills in a large programming class," *International Journal of Information and Education Technology,* vol. 14, no. 11, 2024.

[27] T.-C. Yang, Y.-C. Hsu, and J.-Y. Wu, "The effectiveness of ChatGPT in assisting high school students in programming learning: Evidence from a quasi-experimental research," *Interactive Learning Environments,* pp. 1–18, 2025.

[28] G. Jošt, V. Taneski, and S. Karakatič, "The impact of large language models on programming education and student learning outcomes," *Applied Sciences,* vol. 14, no. 10, 4115, 2024.

[29] R. Mellado, C. Cubillos, and G. Ahumada, "Effectiveness of generative artificial intelligence in learning programming to higher education students," in *Proc. 2024 IEEE International Conference on Automation/XXVI Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, IEEE, 2024, pp. 1–7.

[30] C.-K. Chang, "Enhancing academic performance with generative AI-based quiz platform," in *Proc. 2024 IEEE International Conference on Advanced Learning Technologies (ICALT)*, IEEE, 2024, pp. 193–195.

[31] W. Lyu, Y. Wang, T. Chung, Y. Sun, and Y. Zhang, "Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study," in *Proc. the Eleventh ACM Conference on Learning@ Scale*, 2024, pp. 63–74.

[32] M. Fırat and S. Kuleli, "GPT vs. Google: A comparative study of self-code learning in ODL students," *Journal of Educational Technology and Online Learning,* vol. 7, no. 3, pp. 308–320, 2024.

[33] M. Pankiewicz and R. S. Baker, "Navigating compiler errors with AI assistance—A study of GPT hints in an introductory programming course," in *Proc. the 2024 on Innovation and Technology in Computer Science Education V. 1*, 2024, pp. 94–100.

[34] N. Baláž, J. Porubän, M. Horváth, and T. Kormaník, "Using ChatGPT during implementation of programs in education," in *Proc. 5th International Computer Programming Education Conference (ICPEC 2024)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, pp. 18: 1–18: 9.

[35] S. Abdulla, S. Ismail, Y. Fawzy, and A. Elhag, "Using ChatGPT in teaching computer programming and studying its impact on students performance," *Electronic Journal of e-Learning,* vol. 22, no. 6, pp. 66–81, 2024.

[36] S. McGill and R. McGill, "WIP: Generative AI as an enhanced study aid in engineering courses," presented at ASEE Mid-Atlantic Section Spring Conference, 2024.

[37] M. L. Maher, S. Y. Tadimalla, and D. Dhamani, "An exploratory study on the impact of ai tools on the student experience in programming courses: an intersectional analysis approach," in *Proc. 2023 IEEE Frontiers in Education Conference (FIE)*, IEEE, 2023, pp. 1–5.

[38] G. Huesca *et al.*, "Effectiveness of using ChatGPT as a tool to strengthen benefits of the flipped learning strategy," *Education Sciences,* vol. 14, no. 6, p. 660, 2024.

[39] K. Kwangil, "A study on the effectiveness of generative AI utilization in programming education-focusing on ChatGPT and scratch programming," *Convergence Security Journal,* vol. 24, no. 3, pp. 33–39, 2024.

[40] A. Mehta, N. Gupta, A. Balachandran, D. Kumar, and P. Jalote, *Can Chatgpt Play the Role of a Teaching Assistant in an Introductory Programming Course?* arXiv preprint arXiv:2312.07343, 2023.

[41] F. Ouyang, M. Guo, N. Zhang, X. Bai, and P. Jiao, "Comparing the effects of instructor manual feedback and ChatGPT intelligent feedback on collaborative programming in China's higher education," *IEEE Transactions on Learning Technologies,* 2024.

[42] P. Haindl and G. Weinberger, "Does ChatGPT help novice programmers write better code? Results from static code analysis," *IEEE Access,* 2024.

[43] M. H. Y. Binhammad, A. Othman, L. Abuljadayel, H. Al Mheiri, M. Alkaabi, and M. Almarri, "Investigating how generative AI can create personalized learning materials tailored to individual student needs," *Creative Education,* vol. 15, no. 7, pp. 1499–1523, 2024.

[44] C. Lee, J. Myung, J. Han, J. Jin, and A. Oh, *Learning from Teaching Assistants to Program with Subgoals: Exploring the Potential for AI Teaching Assistants*, arXiv preprint arXiv:2309.10419, 2023.

[45] Y. Chen, S. Xiao, Y. Song, Z. Li, L. Sun, and L. Chen, "MindScratch: A visual programming support tool for classroom learning based on multimodal generative AI," *International Journal of Human–Computer Interaction,* pp. 1–19, 2025.

[46] N. Gardella, R. Pettit, and S. L. Riggs, "Performance, Workload, emotion, and self-efficacy of novice programmers using AI code generation," in *Proc. the 2024 on Innovation and Technology in Computer Science Education*, vol. 1, 2024, pp. 290–296.

[47] J. Liu, X. Tang, L. Li, P. Chen, and Y. Liu, "Which is a better programming assistant? A comparative study between chatgpt and stack overflow," arXiv preprint arXiv:2308.13851, 2023.

[48] S. Li, J. Liu, and Q. Dong, "Generative artificial intelligence-supported programming education: Effects on learning performance, self-efficacy and processes," *Australasian Journal of Educational Technology,* 2025.

[49] G. Puthumanaillam and M. Ornik, *The Lazy Student's Dream: ChatGPT Passing an Engineering Course on Its Own*, arXiv preprint arXiv:2503.05760, 2025.

[50] J. Prather *et al.*, "The widening gap: The benefits and harms of generative ai for novice programmers," in *Proc. the 2024 ACM Conference on International Computing Education Research*, vol. 1, 2024, pp. 469–486.

[51] S. Berrezueta-Guzman, S. Krusche, and S. Wagner, *From Coders to Critics: Empowering Students through Peer Assessment in the Age of AI Copilots*, arXiv preprint arXiv:2505.22093, 2025.

[52] P. Denny *et al.*, "Computing education in the era of generative AI," *Communications of the ACM,* vol. 67, no. 2, pp. 56–67, 2024.

[53] A. Scholl and N. Kiesler, *How Novice Programmers Use and Experience ChatGPT When Solving Programming Exercises in an Introductory Course*, arXiv preprint arXiv:2407.20792, 2024.

[54] M. Giannakos *et al.*, "The promise and challenges of generative AI in education," *Behaviour & Information Technology,* pp. 1–27, 2024.

[55] B. J. Zimmerman, "Becoming a self-regulated learner: An overview," *Theory into Practice,* vol. 41, no. 2, pp. 64–70, 2002.

[56] H. Güner and E. Er, "AI in the classroom: Exploring students' interaction with ChatGPT in programming learning," *Education and Information Technologies,* pp. 1–27, 2025.

Copyright © 2026 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).