International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 5, Sep-Oct-2026, PP:01-10

The Impact of AI-Powered Coding Tools on Students' Learning Habits and Programming

Dr. Manju Rani Assistant Professor[1], Chitrang rajput[2] ,Keleesha[3] Samra saifi[4] Mohamad Eltayeb[5]
School Of Behavioural Sciences, Apeejay Stya University, Gurugram, Haryana, India

Abstract- AI coding assistants can generate code, explain programming concepts, and suggest debugging steps, but their educational value cannot be judged from task completion alone. This focused narrative review examines how AI-assisted programming relates to students' learning behavior, programming performance, independent problem-solving, and psychological processes. It synthesizes recent empirical studies and reviews, including a meta-analysis of 35 controlled studies, a systematic review of 119 papers on AI in programming education, a quasiexperiment with 82 college students, a study of 17 novice programmers, and a randomized controlled trial with 275 students. Across this evidence, AI assistance is often associated with improved performance or efficiency on supported tasks, but gains in conceptual understanding and independent learning are less consistent. Behavioral Science helps explain this distinction through cognitive load, programming self-efficacy, frustration, intrinsic motivation, help-seeking, and metacognitive monitoring. A recent trial found that AI-supported groups performed better on a programming exercise than a no-AI group, yet did not show greater short-term knowledge gains or code-comprehension scores. The review argues that the instructional design of AI support matters: hint-first scaffolding, code verification, explanation, retrieval practice, and independent transfer tasks are more defensible than unrestricted answer generation as default teaching strategies. Because this paper is a focused narrative review, it does not claim exhaustive systematic coverage or report original student data.

Keywords: AI coding assistants, programming education, cognitive load, selfefficacy, self-regulated learning, student behavior, independent

I. INTRODUCTION

Generative artificial intelligence (ai) has introduced a new form of support into programming education. Tools such as chatgpt and github copilot can generate code, interpret error messages, explain unfamiliar syntax, propose alternative algorithms, and help users debug programs. These functions may be particularly attractive to novice programmers, who must coordinate programming syntax, algorithmic reasoning, debugging, and conceptual understanding at the same time.

Programming competence, however, involves more than producing a working program. Students should understand why a solution works, identify assumptions and edge cases, evaluate code quality, and transfer knowledge to a new problem. A correct output produced with substantial ai assistance may not indicate that the student could create or explain the same solution independently. The distinction between performance during a supported task and learning that persists after support is removed is therefore central to evaluating ai in education.

Recent reviews show that ai and machine-learning systems are being used across programming-course design, classroom support, assessment and feedback, and performance monitoring. Manorat et al. (2025) Synthesized 119 papers published from 2012 to 2024, illustrating the breadth of this area. Yet breadth of application does not itself establish that generative coding assistants improve long-term conceptual learning. Empirical studies have reported both benefits and qualifications, reinforcing the need to examine task outcomes alongside students' behavior and psychological experience (alanazi et al., 2025; Sun et al., 2024).

A behavioral science perspective adds an important question: how does the availability of immediate ai assistance change the way students approach difficulty? Students may use an assistant to seek a hint, interpret feedback, or compare approaches. They may also accept an entire solution before making an independent attempt. Such choices are shaped by perceived effort, confidence, motivation, help-seeking habits, and beliefs about what counts as learning. This review therefore treats ai coding assistants not simply as technical tools, but as part of a learning environment that can influence students' cognitive and behavioral processes.

1. Ten-year trajectory of secondary evidence (2016–2025)
A decade-long perspective shows that ai support in programming education did not begin with conversational generative ai. Earlier systems relied on intelligent tutoring, stepwise models, automated grading, program analysis, tests, and targeted feedback. Aleven et al. (2016), In a review of 18 example-tracing intelligent tutors, reported that such tutor systems had reached approximately 44,000 students across educational settings; this is broader intelligent-tutoring evidence, not a direct estimate of coding-assistant use. In programming specifically, gerdes et al. (2017) Described ask-elle, a haskell tutor combining stepwise program

International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 5, Sep-Oct-2026, PP:01-10

development, property-based testing, and on-demand hints. Keuning et al. (2019) Reviewed 101 automated feedback tools and found that feedback more often identified mistakes than explained how to correct them or what to do next. Messer et al. (2023) Later reviewed 121 papers published from 2017 to 2021 and found that programming assessment tools commonly emphasized correctness through unit testing or static analysis, with less attention to maintainability, readability, and documentation.

The arrival of large language model (llm) tools has changed the form of available support, from narrowly structured hints and test feedback to open-ended dialogue and code generation. Large-scale developer surveys provide a secondary indicator of this change: the proportion of respondents reporting that they were using or planning to use ai tools in their development workflow rose from 70% in To 76% in 2024 and 84% in 2025 (stack overflow, 2023, 2024, 2025). In 2025, 44% of respondents who were learning to code reported using ai tools to learn, compared with 37% in 2024. The same survey also recorded substantial reservations: 66% of respondents reported frustration with ai solutions that were almost correct, while 46% distrusted the accuracy of ai output and 33% trusted it (stack overflow, 2025).

Table 1. Selected secondary evidence across the decade (2016–2025)

Year window	/	Secondary source and evidence base	Reported evidence and interpretation
2016	2016	Aleven et al. (2016): review of 18 example-tracing intelligent tutors.	The tutoring technology had been used by approximately 44,000 students and accounted for 40% of datasets in DataShop. This gives a historical benchmark for scalable, structured tutoring, but is not a measure of AI coding assistants.
2017	2017	Gerdes et al. (2017): Ask-Elle programming tutor for Haskell.	The system supported incremental program construction, propertybased testing, and automated hints. The evidence describes an earlier rule-/model-based tutor rather than an LLM-based coding assistant.
Year / window	Year / window	Secondary source and evidence base	Reported evidence and interpretation
2018–2019	2018–2019	Keuning et al. (2019): systematic review coding 101 automatedfeedback tools.	Feedback commonly identified errors; fewer systems helped students determine how to fix them or what step to take next. This highlights the educational value of explanatory, actionable feedback.
2017–2021	2017–2021	Messer et al. (2023): systematic review of 121 research papers on automated grading and feedback.	Most approaches emphasized correctness, unit-test results, or static analysis. The review noted limited evaluation of maintainability, readability, and documentation and limited reproducibility where datasets were unavailable.
2023	2023	Stack Overflow Developer Survey: 89,184 responses to the use/planning item; 67,237 responses to the current-use item.	The report summarized 70% as using or planning to use AI development tools; 44.17% answered that they currently used AI tools. Among respondents learning to code, 82% were using or planning to use them.
2024	2024	Stack Overflow Developer Survey: 60,907 responses to the relevant use/planning question.	76% reported using or planning to use AI tools, and current use was reported as 62%, compared with 44% in 2023. These are annual cross-sectional snapshots rather than a longitudinal student panel.
2024	2024	Sun et al. (2024): quasi-experiment with 82 college students.	ChatGPT-supported programming changed observed behaviors, including debugging and reading feedback; the study did not find a statistically significant between-group difference in measured programming performance.

2025	Alanazi et al. (2025): meta-analysis of 35 controlled studies published during 2020–2024.	Programming performance: SMD = 0.86, 95% CI [0.36, 1.37], p = .0008. Learning success/ease of understanding: SMD = 0.16, 95% CI [−0.23, 0.55], p = .41; this second result was not statistically significant.
2025	Manorat et al. (2025): systematic literature review of 119 papers published during 2012–2024.	AI/ML applications were grouped into course design, classroom implementation, assessment/feedback, and performance monitoring. This broad evidence base includes more than generative coding assistants.
2025	Stack Overflow Developer Survey: 33,662 responses to the AI use/planning item.	84% reported using or planning to use AI tools. Among respondents learning to code, 44% used AI tools to learn to code, up from 37% in 2024. Separately, 66% reported frustration with almost-correct AI solutions and 46% distrusted output accuracy, compared with 33% who trusted it.

Note. This Is A narrative synthesis Of Published secondary Evidence, not a Newly Collected Dataset. The sources Differ in scope, population, design, and outcome; values should not be pooled across rows. Developer-survey findings are not student-only estimates.

These are cross-sectional developer-survey figures with changing respondent counts, not repeated measurements of the same students; they indicate the diffusion and perceived limitations of ai tools, not a causal effect on learning.Evidence directly concerning programming education remains mixed and should be read separately from adoption statistics. Sun et al. (2024) Compared chatgptfacilitated and self-directed programming among 82 college students. The aisupported group displayed different debugging and feedback-reading behaviors, but the study did not find a statistically significant between-group difference in programming performance. Alanazi et al. (2025) Synthesized 35 controlled studies

Published between 2020 and 2024. Their meta-analysis reported a positive effect on programming performance (standardized mean difference [smd] = 0.86, 95% Ci [0.36, 1.37], P = .0008), But did not find a statistically significant advantage for learning success or ease of understanding (smd = 0.16, 95% Ci [−0.23, 0.55], P =.41). Manorat et al. (2025) Reviewed 119 papers published between 2012 and 2024, mapping ai and machine-learning applications across course design, classroom implementation, assessment and feedback, and performance monitoring. Together, these sources span the 2016–2025 evidence window, but they measure different things and should not be combined into one continuous annual learning-outcome series.

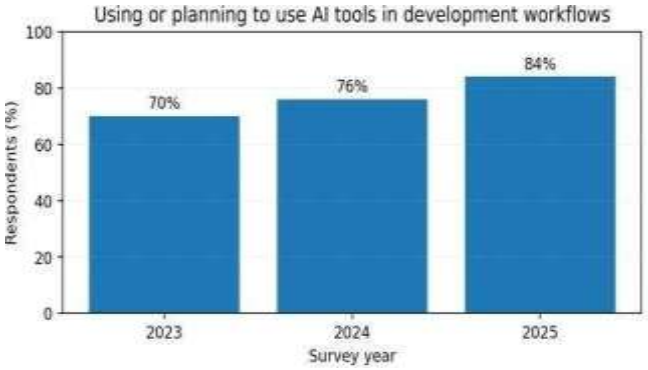


Figure 1. Reported use or planned use of AI tools in developer workflows, 2023–2025

Note. Values are the annual headline estimates reported by stack overflow (2023, 2024, 2025). They are crosssectional survey snapshots with changing sample sizes; they should not be interpreted as a panel trend or as direct evidence of student learning gains.

II. PROBLEM STATEMENT AND RATIONALE

Ai coding assistants can reduce barriers to completing programming tasks, but the relationship between assisted performance and independent competence remains uncertain. A student can complete an assignment more quickly while learning little about the underlying algorithm; conversely, carefully scaffolded assistance may help the student manage cognitive demands while still doing the reasoning needed to learn. The central problem is thus not whether ai should be used in programming education in general, but which forms of assistance support performance,

Understanding, self-regulation, and transfer without displacing meaningful student effort.The issue warrants

attention because educators may incorrectly treat speed, code correctness, user satisfaction, or confidence as direct indicators of learning. These outcomes are related but distinct. An evidence-informed approach should examine them separately and should include psychological variables that help explain why different students may respond differently to similar ai tools.

1. Objectives of the Review

- To Review Evidence On How Ai Coding Assistants Influence Programming Behavior And task performance.
- To Distinguish Immediate Task Performance From Conceptual Understanding, retention, and independent transfer.
- To Examine Psychological Variables, Especially Cognitive Load, Programming Self-efficacy, frustration, motivation, and metacognitive monitoring.
- To identify risks related to over-reliance, inaccurate output, uncritical copying, and assessment validity.
- To Recommend Practical Teaching And Assessment Strategies For Responsible integration of ai coding assistants.

2. Review Questions

- How Are Ai Coding Assistants Being Used In Computer Programming Education?
- What Does Recent Evidence Indicate About Programming-task Performance compared with learning and transfer?

How	may	AI	assistance	influence	students'	help-seeking,	debugging, verification, and independent problem-solving behaviors?
Which	psychological	variables	help	explain	students'	experiences	with AIassisted coding?

1. What Instructional Practices Can Preserve Student Agency And Support Responsible, effective ai use?

3. Review Approach and Scope

This paper uses a focused narrative-review approach. For this revision, targeted searches were made through scholarly search results, journal and publisher pages, acm proceedings, eric records, and author/project pages. Search concepts included ai coding assistant, chatgpt programming education, github copilot, programming performance, cognitive load, self-efficacy, motivation, and self-regulated learning. Priority was given to peer-reviewed empirical studies and reviews published from 2023 to october 2026, with foundational psychological theories included to interpret the findings. The selection was purposive rather than exhaustive. The review does not report a prisma-style screening process, a comprehensive count of all records retrieved, or a formal risk-of-bias assessment; it should therefore not be described as a systematic review or meta-analysis. Quantitative results in the evidence table are reported from the cited studies and reviews, not calculated from a new dataset. Differences in tools, tasks, learning conditions, outcome measures, and student experience also limit direct comparison across studies.

III. CONCEPTUAL FOUNDATIONS FROM BEHAVIORAL SCIENCE

1. Cognitive load and mental effort

Cognitive load theory proposes that working memory is limited and that learning depends in part on how the demands of a task are managed (sweller, 1988). Programming novices may need to hold syntax, control flow, variable states, problem goals, and error messages in mind simultaneously. Ai explanations or debugging suggestions may reduce unnecessary search and help students focus on the conceptual structure of a problem. However, when a tool supplies a complete solution, it may also remove some of the reasoning and practice through which programming knowledge is built.

Cognitive load should not be treated as a single simple measure. Intrinsic load relates to the complexity of the material in relation to prior knowledge; extraneous load concerns demands imposed by poor instructions or inefficient presentation; and germane processing refers to effort invested in constructing and organizing

Knowledge. In practice, these components are difficult to separate perfectly, and findings depend on the instrument and task. Research using guided coding hints suggests that scaffold design can influence cognitive load and selfregulated learning differently from unrestricted solution generation (huang et al., 2025; Bassner et al., 2026).

2. Programming self-efficacy

Self-efficacy is a person's belief in their capability to organize and carry out actions required for a particular task (bandura, 1997). In programming, selfefficacy may influence whether a student starts a difficult problem, persists after an error, seeks help, and attempts a new challenge. Ai support can provide mastery experiences by making a task more manageable, but confidence based mainly on accepting a generated answer may be poorly calibrated. A useful

educational goal is therefore not simply to make students feel more confident; it is to help them develop evidence-based confidence through practice, feedback, explanation, and independent success.

The distinction between perceived competence and demonstrated competence is important. Gardella et al. (2024) Studied 17 novice programmers using github copilot in a within-subjects design. Ai support increased programming efficiency and reduced effort and mental workload, but it did not produce a significant immediate change in self-efficacy or emotion. The small sample and timepressured setting limit generalization, but the study cautions against assuming that performance assistance automatically changes learners' confidence.

3. Self-regulated learning and metacognition

Self-regulated learning involves setting goals, planning strategies, monitoring progress, and evaluating outcomes (zimmerman, 2000). Metacognition is especially important in ai-assisted coding because a fluent explanation can appear convincing even when the generated code is incorrect or unsuitable. Students need to monitor whether they understand each step, test whether the code works under relevant conditions, and judge whether they could reproduce the solution without the assistant.An ai assistant can support self-regulation when it offers graduated hints, prompts the learner to explain their reasoning, and provides feedback that encouragesRevision. It may weaken self-regulation when the interaction becomes a sequence of broad prompts followed by unexamined copying. The design of the task and the learner's interaction strategy are therefore central variables, not incidental details.

4. Technology acceptance and perceived usefulnes

The technology acceptance model proposes that perceived usefulness and perceived ease of use help shape attitudes and intentions toward using a technology (davis, 1989). Sun et al. (2024) Found that experience with chatgptassisted programming changed students' perceptions, including perceived usefulness, ease of use, and intention to use. Acceptance is relevant to adoption, but it is not itself evidence of learning effectiveness. A tool can be easy to use and highly preferred while still needing instructional safeguards to support conceptual understanding.

IV. Findings from the Literature

AI assistants change programming behavior

In a quasi-experimental study involving 82 college students, sun et al. (2024) Compared chatgpt-facilitated programming with self-directed programming. Students in the chatgpt condition more frequently engaged in debugging-related actions, received and read feedback, and copied code to and from chatgpt. These behavioral differences show that ai changes the workflow around programming. They do not, by themselves, indicate that students learned more or less; interpretation depends on whether the actions involved active checking and reasoning or passive acceptance.

Help-seeking should therefore be understood in terms of quality, not only frequency. A student who asks for a hint after testing an initial idea may be using support productively. A student who requests full code without attempting the problem may bypass a useful learning opportunity. Classroom activities can make this distinction visible by asking students to record their initial attempt, explain the prompt they used, identify changes they made, and describe how they verified the outcome.

Immediate performance does not equal durable learning

The strongest quantitative summary in the current literature is the meta-analysis by alanazi et al. (2025), Which examined 35 controlled studies of ai tools in computer-programming learning. It reported a positive standardized mean difference for programming performance (smd = 0.86, 95% Ci [0.36, 1.37], P = .0008; I^2 = 54%). In contrast, the pooled estimate for learning success or ease of understanding was small and statistically non-significant (smd = 0.16, 95% Ci [0.23, 0.55], P = .41; I^2 = 55%). These outcomes suggest a distinction between doing better on a supported task and developing deeper understanding, while moderate heterogeneity indicates that effects differ across studies.

A 2026 randomized controlled trial by bassner et al. Compared a scaffolded ai tutor, unrestricted chatgpt, and a no-ai condition in an introductory programming course (n = 275 after data-quality filters). Mean exercise performance was 71.84% With chatgpt, 57.50% With the scaffolded tutor, and 29.85% In the no-ai group. Despite the performance differences, the groups did not differ significantly in short-term knowledge gains or code-comprehension performance. The study also found reduced frustration and changes in reported cognitive load among ai-supported learners, while intrinsic motivation increased significantly only in the scaffolded-tutor condition. This is one specific course and one 90-minute exercise; it should not be interpreted as proof that ai never improves learning. It does, however, demonstrate why task scores and learning measures should be reported separately.

Results from huang et al. (2025) Point to the importance of scaffold design. Their codeflow assistant offered different levels of guidance, from flowcharts and incomplete code to more complete solutions. The ai-assisted group achieved higher coding scores than a teaching-assistant-supported comparison group, showed comparable improvements in conceptual understanding, and improved in aspects of self-regulated learning. Taken together with bassner et al. (2026), The findings support testing how hints, staged support, and

learner explanation can be built into ai-assisted programming rather than assuming that more complete answers are always better.

Psychological variables that may shape outcomes

Cognitive load, self-efficacy, frustration, motivation, and metacognitive monitoring are useful psychological variables for explaining student responses to ai coding assistants. They are related, but they should not be used interchangeably.

Table 2. Psychological variables relevant to AI-assisted programming

Psychological variable	Psychological variable	How it may operate during AI-assisted coding	What should be measured
Cognitive load / mental effort	Cognitive load / mental effort	AI may reduce unproductive search and syntax burden; complete answers may also reduce active problem-solving.	Perceived mental effort and, where feasible, separate intrinsic, extraneous, and germane load measures.
Programming self-efficacy	Programming self-efficacy	Confidence may improve after successful practice, but confidence can be inflated if the student cannot reproduce the solution independently.	Task-specific confidence before and after coding, alongside actual independent performance.
Frustration and affect	Frustration and affect	Hints and error explanations may reduce frustration; unhelpful or inconsistent outputs may increase it.	Brief validated affect/frustration ratings tied to a defined task and time point.
Intrinsic motivation	Intrinsic motivation	Scaffolded assistance may support competence while preserving autonomy; full-solution dependence may make activity feel easier without increasing interest.	Interest/enjoyment ratings and optional persistence on a new, unsupported task.
Psychological variable	Psychological variable	How it may operate during AI-assisted coding	What should be measured
Metacognitive monitoring / regulation	self-	Students must judge what they understand, test outputs, plan next steps, and decide when to seek help.	Prompt logs, verification behaviors, explanation quality, independent transfer, and self-regulation scales.

The direct evidence for these variables is still uneven. Gardella et al. (2024) Found lower mental workload but no significant immediate self-efficacy effect in a small study. Bassner et al. (2026) Found lower frustration for ai-supported groups and a motivational advantage for hint-based support, while huang et al. (2025) Reported improvements in self-regulated learning with structured coding hints. These findings are better treated as evidence that psychological outcomes depend on tool design and context than as a universal pattern.

What broad reviews add

Manorat et al. (2025) Reviewed 119 papers on ai and machine learning in computer-programming education, grouping applications around course design, classroom implementation, assessment and feedback, and performance monitoring. This broader field includes more than generative coding assistants, so its paper count should not be interpreted as 119 controlled trials of chatgpt or copilot. The review is nevertheless useful for showing that ai in programming education includes a wider instructional system, including feedback, assessment design, and identification of learning needs.

Alanazi et al. (2025) Provide a more directly quantitative synthesis of controlled evidence, but their pooled estimates also combine different ai tools, tasks, and outcome measures. The appropriate conclusion is not that all ai assistants have the same effect. Rather, effect estimates should be read alongside the support offered, the learner population, prior knowledge, the duration of the intervention, and how learning was measured.

International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 5, Sep-Oct-2026, PP:01-10

V. QUANTITATIVE EVIDENCE SUMMARY

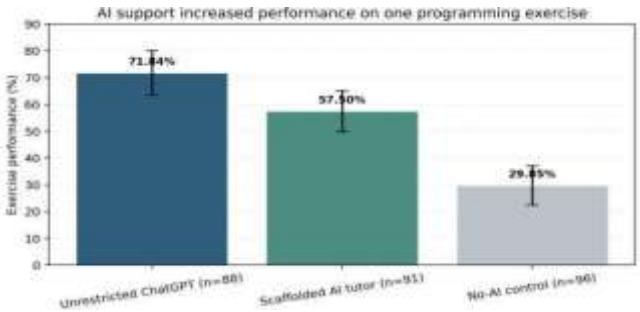


Figure 2. Mean exercise performance in the 2026 randomized trial

Note. Bars Show Mean Scores; Whiskers Show Approximate 95% Confidence Intervals calculated From The published Means, standard Deviations, And Group sizes for One

90-Minute programming exercise: chatgpt, m = 71.84, Sd = 39.65 (N = 88); scaffolded ai tutor (iris), m = 57.50, Sd = 37.36 (N = 91); no-ai control, m = 29.85, Sd = 36.17 (N = 96). The Chart Represents Task Performance In One Study, Not General programming competence or long-term learning. Source: bassner et al. (2026).

Table 3. Key studies informing the review

Study	Design and evidence base	Main findings relevant to this review	Interpretive caution
Sun et al. (2024)	Quasi-experiment; 82 college students; ChatGPT-assisted vs selfdirected programming.	AI group showed more debugging, feedback-reading, and code copy/paste interactions. Perceptions of usefulness, ease of use, and intention to use changed; between-group performance difference was not statistically significant.	Single instructional context; more interaction does not automatically mean deeper learning.
Gardella et al. (2024)	Within-subjects study; 17 novice programmers using GitHub Copilot under time pressure.	AI support improved efficiency and reduced effort/mental workload; immediate emotion and self-efficacy did not significantly change.	Small sample and time-pressured context; treat as preliminary evidence.
Alanazi et al. (2025)	Systematic review and meta-analysis of 35 controlled studies.	Programming performance: SMD = 0.86, 95% CI [0.36, 1.37], p = .0008, I^2 = 54%. Learning success/ease of understanding: SMD = 0.16, 95% CI [-0.23, 0.55], p = .41, I^2 = 55%.	Pooled effects varied across tool types, tasks, and measures; performance and understanding were distinct outcomes.
Manorat et al. (2025)	Systematic literature review of 119 papers published from 2012 to 2024.	Mapped AI/ML applications in course design, classroom implementation, assessment/feedback, and performance monitoring.	Scope includes AI/ML broadly, not only generative coding assistants or controlled effects on learning.
Huang et al. (2025)	Controlled study of a structured, self-regulated-learning coding assistant versus teaching-assistantsupported programming.	Coding scores were higher with staged GenAI hints; conceptual understanding improved comparably; some selfregulation indicators improved and cognitive-load components shifted favorably.	The specific tool and instructional design matter; findings should not be generalized to unrestricted answer generation.
Bassner et al. (2026)	Randomized controlled trial; 275 students in three conditions after data-quality filters.	Exercise scores: ChatGPT 71.84%, Iris 57.50%, no AI 29.85%. Neither AI condition had greater short-term knowledge gains or code-comprehension performance; AI groups reported lower frustration/load, and only Iris increased intrinsic motivation.	One course, one 90minute exercise, short-term learning measures, and a postfilter sample; delayed retention and transfer remain open questions.

International Journal for Novel Research in Economics , Finance and Management
www.ijnrefm.com
ISSN:3048-7722
Volume 4, Issue 5, Sep-Oct-2026, PP:01-10

VI. PROPOSED CONCEPTUAL FRAMEWORK

Based on the reviewed literature, this paper proposes a conceptual framework in which ai support design influences student behavior, which in turn interacts with psychological processes and learning outcomes. The framework is conceptual and has not been empirically tested by the present paper. It distinguishes immediate performance (for example, task completion and code correctness) from durable learning (for example, conceptual understanding, transfer, retention, and independent coding). Prior knowledge, task difficulty, prompt literacy, and course design may moderate these relationships.

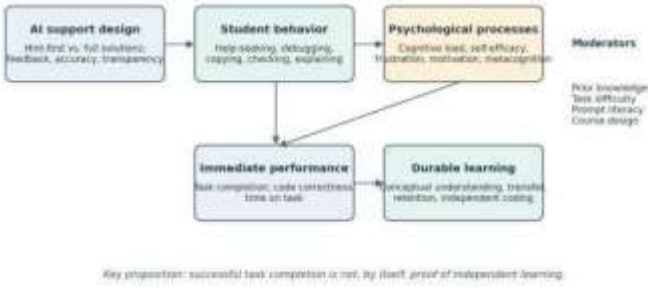


Figure 3. A proposed framework linking AI support, student behavior, psychological processes, and outcomes

Note. This Is A Proposed Synthesis Of Constructs Discussed In The Cited Literature, Not a tested causal model.

1. Discussion

The reviewed evidence supports a balanced conclusion. Ai coding assistants can alter student behavior and improve performance or efficiency on supported tasks. Meta-analytic and experimental results also indicate that those performance

Benefits do not consistently translate into measurable short-term gains in conceptual understanding. This is not evidence that ai is inherently harmful to learning; it is evidence that the task, design of assistance, and outcome measure matter.

The psychological perspective helps explain why a simple benefit-versus-risk framing is inadequate. Reducing cognitive burden can help students engage with material that otherwise feels inaccessible. However, learning also requires effortful processes such as tracing code, predicting outputs, diagnosing errors, explaining decisions, and retrieving knowledge without assistance. Some difficulty is unproductive and can be reduced by clear hints; other difficulty represents the very reasoning students need to practise. Instructional design should aim to remove unnecessary friction without removing all opportunities for thinking.

Self-efficacy and perceived usefulness also require careful interpretation. Ai can make students feel more capable because it provides fast, successful outputs, and this perceived usefulness can support continued use (davis, 1989). Yet confidence should be checked against independent performance. Students should be asked to explain generated code, predict its behavior, test edge cases, and solve a related problem without ai. Such assessment gives a more direct indication of whether the student has internalized the relevant knowledge.

The more promising educational model is structured assistance. Hint-first systems, staged prompts, explanation requests, and code verification can support progress while leaving cognitive work for the student. Educators can also make the learning process visible by asking students to submit their initial attempt, final solution, tests, concise rationale, and a reflection on which ai suggestions were accepted or rejected. These strategies do not eliminate misuse, but they align assessment more closely with understanding rather than mere production of code.

2. Recommendations for Students and Educators

Recommended workflow for students

- Attempt The Problem First: Restate The Task, Identify Inputs And Outputs, And Draft An algorithm or pseudocode before requesting a full solution.
- Ask For The Smallest Useful Support: Request A Hint, An Explanation Of An Error, Or a comparison of approaches before requesting complete code.
- Verify rather than trust: run tests, inspect edge cases, compare the response with official documentation, and check security or data-handling implications where relevant.
- Explain the solution in your own words: describe the algorithm, important variables, control flow, and why the approach works.
- Remove The Assistant And Retry: Solve A Parallel Problem Or Reproduce Key Parts from memory to test independent understanding.
- Follow Academic Integrity Rules: Disclose Ai Assistance When Required And Do Not submit generated work in violation of course policy.

Recommendations for instructors

- Design ai use around staged scaffolding, with hints first and complete solutions only when pedagogically justified.

- Assess process as well as product through code explanation, debugging exercises, oral questioning, test design, and independent transfer tasks.
- Use low-stakes, no-ai activities at intervals so students and instructors can assess unaided competence.
- Teach verification explicitly, including how to test output, inspect edge cases, detect hallucinated apis, and compare code against trusted references.
- Provide clear rules distinguishing permitted brainstorming, debugging help, code generation, collaboration, and disclosure expectations.
- Evaluate educational outcomes using multiple measures: task correctness, code quality, conceptual tests, delayed assessments, confidence calibration, cognitive load, and self-regulation.

3. Directions for Future Research

Future studies should examine not only whether ai is available, but how it is configured and how students are prompted to use it. Research designs can compare unrestricted full-code generation with hint-first scaffolding, staged disclosure, and feedback that asks students to explain a proposed solution. Where feasible, studies should include baseline programming knowledge, delayed tests, and novel transfer tasks. The use of multiple outcomes would help identify cases in which ai improves task completion without producing durable understanding.

The following hypotheses are suitable for future empirical study; they are proposed questions, not findings of this review:

- H1: hint-first ai assistance will be associated with stronger independent transfer performance than unrestricted full-solution generation, after controlling for prior programming knowledge.
- H2: the association between ai assistance and immediate task performance will be stronger than its association with delayed conceptual retention.
- H3: higher verification and metacognitive-monitoring behavior will be associated with better independent performance, regardless of frequency of ai use.
- H4: changes in cognitive load and frustration will vary by the design of ai assistance; lower reported effort alone will not necessarily predict stronger learning gains.
- H5: programming self-efficacy will predict persistence and help-seeking, but its relationship with performance will be stronger when confidence is calibrated against independent task results.

Limitations

This paper has several limitations. It is a focused narrative review rather than a systematic review, so the selection of studies is not exhaustive and no formal riskof-bias assessment was conducted. The included studies vary in participant characteristics, ai tools, course design, task duration, outcome definitions, and measurement methods. Some have small samples or short interventions. Quantitative findings from a single study should not be generalized to all programming settings, and the review cannot establish causality beyond what is supported by the design of each cited study. Finally, ai systems change rapidly; results obtained with one model or interface may not replicate exactly with later versions. The proposed conceptual framework and hypotheses require direct empirical testing.

VII. CONCLUSION

Ai coding assistants are changing how students seek support, debug programs, and approach programming tasks. Current evidence suggests that they can improve assisted task performance and reduce some immediate difficulties, but the transfer

From supported performance to independent programming knowledge is not guaranteed. Behavioral science concepts clarify why this distinction matters: cognitive load, self-efficacy, motivation, frustration, helpseeking, and metacognitive monitoring influence how students use ai and what they learn from it.

The most defensible educational response is neither uncritical adoption nor blanket reliance on prohibition. Ai should be integrated through clear expectations, hint-first scaffolding, active verification, student explanation, independent practice, and assessments of transfer and retention. Ultimately, the success of ai in programming education should be judged not just by whether students can produce working code with a tool, but by whether they become more capable of understanding, evaluating, and creating code when they must think for themselves.

REFERENCES

1. Alanazi, M., Soh, B., Samra, H., & Li, A. (2025). The influence of artificial intelligence tools on learning outcomes in computer programming: A systematic review and meta-analysis. Computers, 14(5), Article https://doi.org/10.3390/computers14050185
2. Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S., Ringenberg, M., & Koedinger, K. R. (2016). Example-tracing tutors: Intelligent tutor development for
3. nonprogrammers. International Journal of Artificial Intelligence in Education, 26(1), 224–
4. https://doi.org/10.1007/s40593-015-0088-2
5. Bandura, A. (1997). Self-efficacy: The exercise of control. W. H. Freeman.

6. Bassner, P., Lenk-Ostendorf, B., Beinstingel, R., Wasner, T., & Krusche, S. (2026). Less stress, better scores, same learning: The dissociation of performance and learning in AIsupported programming education. Computers and Education: Artificial Intelligence, 10, Article 100537. https://doi.org/10.1016/j.caeai.2025.100537
7. Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 13(3), 319– https://doi.org/10.2307/249008
8. Gardella, N., Pettit, R., & Riggs, S. L. (2024). Performance, workload, emotion, and self-efficacy of novice programmers using AI code generation. In Proceedings of the 2024 ACM Conference on Innovation and Technology in Computer Science Education (Vol. (pp. 290–296). Association for Computing Machinery. https://doi.org/10.1145/3649217.3653615
9. Gerdes, A., Heeren, B., Jeuring, J., & van Binsbergen, L. T. (2017). Ask-Elle: An adaptable programming tutor for Haskell giving automated feedback. International Journal of Artificial Intelligence in Education, 27(1), 65– https://doi.org/10.1007/s40593-0150080-x
10. Huang, A. Y. Q., Lin, C.-Y., Su, S.-Y., & Yang, S. J. H. (2025). The impact of GenAI-enabled coding hints on students' programming performance and cognitive load in an SRL-based Python course. British Journal of Educational Technology, 56(5), 1942–1972. https://doi.org/10.1111/bjet.13589
11. Keuning, H., Jeuring, J., & Heeren, B. (2019). A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education, 19(1), Article https://doi.org/10.1145/3231711
12. Manorat, P., Tuarob, S., & Pongpaichet, S. (2025). Artificial intelligence in computer programming education: A systematic literature review. Computers and Education: Artificial Intelligence, 8, Article 100403. https://doi.org/10.1016/j.caeai.2025.100403
13. Messer, M., Brown, N. C. C., Kölling, M., & Shi, M. (2023). Automated grading and feedback tools for programming education: A systematic review [Preprint]. arXiv. https://arxiv.org/abs/2306.11722
14. Stack Overflow. (2023). Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/
15. Stack Overflow. (2024). Stack Overflow Developer Survey 2024: AI. https://survey.stackoverflow.co/2024/ai
16. Stack Overflow. (2025). Stack Overflow Developer Survey 2025: AI. https://survey.stackoverflow.co/2025/ai
17. Sun, D., Boudouaia, A., Zhu, C., & Li, Y. (2024). Would ChatGPT-facilitated programming mode impact college students' programming behaviors, performances, and perceptions? An empirical study. International Journal of Educational Technology in Higher Education, 21, Article https://doi.org/10.1186/s41239-024-00446-5
18. Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Science, 12(2), 257– https://doi.org/10.1207/s15516709cog1202_4
19. Zimmerman, B. J. (2000). Attaining self-regulation: A social cognitive perspective. In M. Boekaerts, P. R. Pintrich, & M. Zeidner (Eds.), Handbook of self-regulation (pp. 13–39). Academic Press. https://doi.org/10.1016/B978-012109890-2/50031-7