International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 10 | Oct 2026 www.irjet.net p-ISSN: 2395-0072

The Impact of AI Coding Assistants on Students' Programming and Problem-Solving Abilities

A Survey-Based Study of Student Use, Perceptions, and Learning Implications

Manan Doshi[1], Meet Sanwadkar[1], Laxman Ingle[1], Azimuddin Shaikh[1], Dr. Roopali Patil[2]

[1]*Department of MCA, D. Y. Patil College of Humanities and Sciences, D. Y. Patil Deemed to be University, Navi Mumbai, Maharashtra, India*

[2]*Associate Professor, D. Y. Patil Deemed to be University, School of Humanities and Sciences, Belapur, Maharashtra, India.*

Abstract - *Generative artificial intelligence (GenAI) coding assistants are increasingly used for code generation, explanations, debugging, and feedback. This study examines student adoption, usage patterns, perceived learning benefits, and use of AI-generated code without sufficient understanding. A quantitative cross-sectional survey collected 59 responses through Google Forms. The sample was predominantly MCA students (69.5%), and 93.2% reported using AI coding assistants. ChatGPT was the most frequently selected tool (52.5%), while 69.5% reported frequent use. Agreement that AI improved programming skills and problem-solving was 64.4% for each outcome; 72.9% rated the overall learning effect positively. However, 78.0% reported using AI-generated code without full understanding at least sometimes. Exploratory exact binomial tests, Wilson confidence intervals, and Holm correction were applied to four aggregate response proportions against a 50% benchmark. After correction, the perceived programming and problem-solving benefit proportions were not statistically significant, while incomplete understanding and positive overall perception were.*

Keywords - AI coding assistants, programming education, generative AI, problem-solving, AI dependence, student learning

1. INTRODUCTION

Programming education develops computational thinking, algorithm design, debugging, code comprehension, and independent reasoning not merely the ability to produce executable code. Large language model (LLM) assistants can explain errors, generate examples, propose alternatives, and provide rapid feedback. However, generated code can be plausible but incorrect, and accepting a complete solution may shift essential reasoning away from the learner [1]–[4], [6], [7].

The educational issue is therefore how students use and verify AI output, and whether the assistance builds capability or substitutes for practice. This paper combines a survey of 120 students with a focused review of studies published during 2024–2026. It describes usage and perceptions; it does not claim that AI use causes improvement or decline in objectively measured programming ability.

2. PROBLEM STATEMENT AND RESEARCH GAP

AI coding assistants reduce the effort needed to obtain candidate code, but programming competence requires students to reason about requirements, design algorithms, identify defects, justify choices, and transfer learning to unfamiliar tasks. Uncritical acceptance can conceal gaps in understanding, while guided assistance may improve access to feedback. The specific problem is the coexistence of favorable student perceptions and reported use of AI-generated code without complete understanding, alongside the absence of direct performance measures in the present survey.

Recent studies and reviews cover tutoring systems, code generation, debugging, assessment, and AI literacy, but differ in tools, course settings, scaffolding, samples, and outcome metrics [6], [7], [11]–[13]. The present survey adds a compact student-centered profile combining tool preference, use frequency, task purpose, perceived benefits, overall learning effect, and incomplete understanding. Its contribution is descriptive: the aggregate data cannot establish whether use frequency, tool choice, or prior experience predicts learning outcomes.

3. LITERATURE REVIEW

Empirical evidence suggests that AI changes programming behavior, but learning gains depend on how help is structured. Sun et al. [1] compared ChatGPT-facilitated and self-directed programming among 82 college students. AI-assisted learners engaged more with debugging feedback and copy-paste workflows, but the performance difference was not statistically significant. Jing et al. [2] found that programming knowledge, AI literacy, and awareness influenced how effectively 53 participants used ChatGPT to solve programming problems; access alone was insufficient.

In an 11-week study, Yang et al. [4] compared a ChatGPT-based Python tutor with a comparison group (35 and 36 students). Structured tutoring supported engagement and task completion, including for learners with weaker foundations, while raising concern about over-reliance. Kohen-Vacs et al. [5] found that positive perceptions do not remove the difficulty of detecting and correcting errors in generated code. These studies distinguish getting an answer from evaluating whether it is correct and understanding why it works.

Recent reviews broaden the evidence base. Manorat et al. [6] reviewed 119 papers and described growing use of AI for classroom learning support and assessment. Nathaniel et al. [12] synthesized 40 empirical studies and emphasized curriculum alignment, self-regulated learning, educator preparation, and iterative feedback. Chen and Cheung [13], in a meta-analysis of 57 studies and 97 estimates across higher education, reported positive pooled learning effects but no significant metacognition effect; the results are not specific to programming courses. Two 2026 reviews also report diverse tools and outcomes alongside concerns about learning depth and dependence [7], [11].

A 2026 short-course study of a guidance-first programming chatbot reported stronger pre/post gains than unrestricted chatbot use [8]. By contrast, a mixed-method study of 855 programming students found that perceived usefulness and ChatGPT use did not significantly predict course marks (R^2 = 0.006) [9]. A further study links computational-thinking level to the way students interact with coding assistants [10]. Taken together, these findings support guided assistance and direct assessment of reasoning rather than judging learning from satisfaction or task completion alone.

TABLE I. COMPARISON OF SELECTED STUDIES (2024–2026)

Study / method and sample	Measures and principal findings	Limitations / relevance
Sun et al. [1], 2024; quasi-experiment, n=82; ChatGPT vs self-directed programming.	Programming behavior, performance and perceptions; more feedback/copy-paste behavior, but no significant performance difference.	Course-specific, short intervention; illustrates that altered workflow is not proof of learning gain.
Jing et al. [2], 2024; quasi-experiment, n=53; ChatGPT problem solving.	Problem-solving outcomes and learner factors; programming knowledge and AI literacy influenced effective use.	Limited tasks and setting; access alone was not enough.
Yang et al. [4], 2024; 11-week tutor study, n=71; PyTutor for Python.	Engagement, task completion and learning; structured tutoring supported learners with weaker foundations.	Specific introductory course and tutoring design; over-reliance remains a concern.
Kohen-Vacs et al. [5], 2025; two quantitative studies.	Perceptions and correction of LLM code; students valued AI but found error correction challenging.	Assessment and course context may affect transferability.
Nathaniel et al. [12], 2025; review of 40 empirical studies; Chen & Cheung [13], 2025; meta-analysis of 57 studies.	Pedagogical integration, learning effect sizes and moderators; findings favor deliberate scaffolding but vary by outcome.	Reviews combine heterogeneous designs; broad higher-education results are not coding-specific.
Noraset et al. [8], 2026; controlled short course, n=42; Gambo et al. [9], 2026; mixed methods, n=855.	Pre/post Python tests and interaction logs in [8]; course marks and perceptions in [9]. Guided help improved gains; perceptions did not predict marks in [9].	Small short-course sample in [8]; single-institution cross-sectional study in [9].

4. RESEARCH QUESTIONS, OBJECTIVES AND HYPOTHESES

The study addresses five questions: (RQ1) which assistants students use and how frequently; (RQ2) which programming activities they support; (RQ3) how students perceive effects on programming skills and problem-solving; (RQ4) how often generated code is used without full understanding; and (RQ5) what overall learning effect students report.

Objectives are to summarize adoption and usage, quantify perceived programming and problem-solving benefits, estimate the frequency of incomplete understanding, describe the overall learning assessment, and interpret these results in light of recent literature.

Four exploratory hypotheses compare response proportions with a 50% benchmark. For each outcome, H_0: $p = 0.50$ and H_1: $p \neq 0.50$. Outcomes are: (H1) agreement that programming skills improved; (H2) agreement that problem-solving improved; (H3) using AI-generated code without full understanding at least sometimes; and (H4) a positive/very positive overall learning effect. The benchmark is not a control group and does not represent objective competence.

5. RESEARCH METHODOLOGY

5.1 Design, sample and instrument

A quantitative, cross-sectional descriptive survey was conducted using a 10-question Google Form. The convenience/non-probability sample contained 59 students from computing-related programmes: 41 MCA (69.5%), six BCA (10.2%), and 12 in "Others" (20.3%). Programming experience was less than one year for 19 respondents (32.2%), one to two years for 15 (25.4%), two to three years for 11 (18.6%), and more than three years for 14 (23.7%).

Questions covered course, experience, AI adoption, preferred assistant, use frequency, primary activity, perceived programming-skill improvement, perceived problem-solving improvement, code use without full understanding, and overall learning effect. The tools reported included ChatGPT, Claude, Gemini, GitHub Copilot, Cursor, and other systems. No AI model was trained or benchmarked in this study.

5.2 Tools, analysis and ethics

Frequencies and percentages were summarized from the aggregate chart counts. For four binary outcomes, two-sided exact binomial tests compared each observed proportion with 0.50; Holm correction controlled family-wise error across the four tests. Wilson 95% confidence intervals were calculated. The workflow used Google Forms for collection, Python with SciPy for exact tests, statsmodels for confidence intervals, and Microsoft Word for manuscript preparation. No regression, chi-square association, t-test, ANOVA, reliability statistic, or between-group test was run because respondent-level data were unavailable. The supplied materials do not state an ethics approval number or reproduce the exact consent text; these details should be verified by the researcher before formal submission.

5.3 Variables and operational measures

The questionnaire operationalized AI use through categorical responses and learning perceptions through individual items. Because only aggregate charts were available, the variables were summarized separately; moderation by prior knowledge, correlations among use patterns, and multi-item scale reliability were not estimable.

TABLE II. OPERATIONALIZATION OF STUDY VARIABLES

Variable	Measure used in the questionnaire
Adoption and preferred tool	Q3: AI use (Yes/No); Q4: most-used assistant category.
Use intensity and purpose	Q5: Rarely to Always; Q6: primary programming activity.
Perceived programming benefit	Q7: agreement with "AI coding assistants have improved my programming skills" on a 1–5 scale.
Perceived problem-solving benefit	Q8: agreement with "AI coding assistants have improved my problem-solving ability" on a 1–5 scale.
Comprehension risk	Q9: frequency of using generated code without fully understanding it.
Overall learning effect	Q10: five ordered categories from Very Negative to Very Positive.

The available materials do not identify an ethics approval number or preserve the exact consent wording. Before formal submission, the researcher should confirm whether participation was voluntary, how respondents were informed about the purpose of the survey, and how data were stored. No attempt is made here to identify individual respondents.

The analysis assumes mutually exclusive response categories and one response per question. Chart-derived small categories should be checked against the original spreadsheet. Q4–Q6 display responses for all 59 students although Q3 shows that four respondents did not use AI; the original form routing and denominators require verification.

6. DATA ANALYSIS AND INTERPRETATION

6.1 Adoption, tools and use cases

AI coding assistants were used by 55 of 59 respondents (93.2%). ChatGPT was most frequently selected (31; 52.5%), followed by Claude (12; 20.3%), Gemini (9; 15.3%), GitHub Copilot (4; 6.8%), Cursor (2; 3.4%), and other tools (1; 1.7%). Frequent use Often, Very Often or

Always—was reported by 41 students (69.5%); 18 (30.5%) selected Always. For primary use, 26 (44.1%) chose "All of the above"; writing code was selected by 11 (18.6%), learning concepts by seven (11.9%), debugging by six (10.2%), understanding code by five (8.5%), and solving programming problems by four (6.8%).

TABLE III. MAIN SURVEY FINDINGS (n = 59)

Indicator	Count (%)	Interpretation
AI assistant adoption	55 (93.2%)	Adoption is common.
Frequent use	41 (69.5%)	Often, very often or always.
ChatGPT most-used	31 (52.5%)	Largest single tool category.
Perceived programming benefit	38 (64.4%)	Agree / strongly agree.
Perceived problem-solving benefit	38 (64.4%)	Agree / strongly agree.
Code used without full understanding	46 (78.0%)	At least sometimes.
Positive overall learning effect	43 (72.9%)	Positive / very positive.

TABLE IV. RESPONSE DISTRIBUTIONS FOR SURVEY ITEMS

Survey item	Response distribution (n, %)
Course	BCA 6 (10.2%); MCA 41 (69.5%); Others 12 (20.3%).
Programming experience	<1 year 19 (32.2%); 1–2 years 15 (25.4%); 2–3 years 11 (18.6%); >3 years 14 (23.7%).
AI tool used most	ChatGPT 31 (52.5%); Claude 12 (20.3%); Gemini 9 (15.3%); GitHub Copilot 4 (6.8%); Cursor 2 (3.4%); Other 1 (1.7%).
Use frequency	Rarely 3 (5.1%); Sometimes 15 (25.4%); Often 12 (20.3%); Very often 11 (18.6%); Always 18 (30.5%).
Main use	Writing code 11 (18.6%); debugging 6 (10.2%); understanding code 5 (8.5%); learning concepts 7 (11.9%); problem solving 4 (6.8%); all listed uses 26 (44.1%).
Programming skill item	Strongly disagree 4 (6.8%); disagree 3 (5.1%); neutral 14 (23.7%); agree 28 (47.5%); strongly agree 10 (16.9%).
Problem-solving item	Strongly disagree 0 (0%); disagree 3 (5.1%); neutral 18 (30.5%); agree 27 (45.8%); strongly agree 11 (18.6%).
Code without full understanding	Never 6 (10.2%); rarely 7 (11.9%); sometimes 28 (47.5%); often 14 (23.7%); very often 4 (6.8%).
Overall learning effect	Very negative 0 (0%); negative 3 (5.1%); no significant effect 13 (22.0%); positive 35 (59.3%); very positive 8 (13.6%).

6.2 Perceived outcomes and comprehension risk

For programming skills, 38 respondents (64.4%) agreed or strongly agreed that AI improved their skills; 14 (23.7%) were neutral and seven (11.9%) disagreed or strongly disagreed. For problem-solving, 38 (64.4%) agreed or strongly agreed, 18 (30.5%) were neutral and three (5.1%) disagreed. The reconstructed single-item means were 3.63/5 for programming skills, 3.78/5 for problem-solving, and 3.81/5 for overall learning effect; these are descriptive summaries, not validated composite scales. Overall, 43 (72.9%) rated the learning effect positive or very positive, 13 (22.0%) reported no significant effect, and three (5.1%) reported a negative effect. At the same time, 46 students (78.0%) used AI-generated code without full understanding at least sometimes, while 13 (22.0%) selected rarely or never.

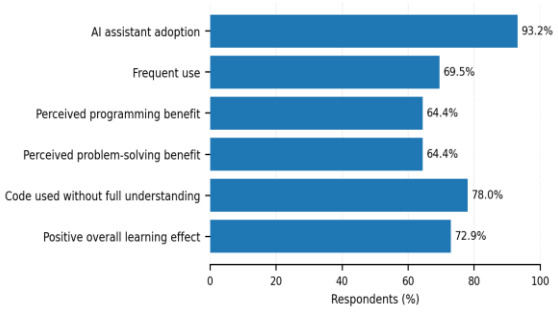


Fig. 1. Key survey proportions; outcomes are self-reported, not objective performance measures.

6.3 Statistical hypothesis testing

Exact two-sided binomial tests were used because the available evidence was aggregate count data. Holm-adjusted p-values are reported for the four pre-specified exploratory comparisons. The tests estimate whether each response proportion differs from 50%; they do not test causation, compare AI users with non-users, or measure

coding competence. The four hypotheses were assessed at $\alpha = 0.05$.

TABLE V. EXPLORATORY HYPOTHESIS TESTS AGAINST p = 0.50

Outcome	Observed	95% CI	Exact p	Holm p / decision
H1 Programming benefit	38/59 (64.4%)	51.7–75.4%	0.0363	0.0727; not sig.
H2 Problem-solving benefit	38/59 (64.4%)	51.7–75.4%	0.0363	0.0727; not sig.
H3 Incomplete understanding	46/59 (78.0%)	65.9–86.6%	0.000019	0.000077; sig.
H4 Positive overall effect	43/59 (72.9%)	60.4–82.6%	0.000584	0.001753; sig.

After Holm correction, H1 and H2 were not statistically significant at $\alpha = 0.05$, despite observed agreement of 64.4%. H3 and H4 remained significant: the proportion reporting incomplete understanding and the proportion reporting positive overall effects were both greater than the 50% benchmark under this test. The result highlights that a positive view of AI can coexist with incomplete comprehension. Sampling bias and measurement limitations remain, and the tests should be treated as exploratory.

6.4 Interpretation of uncertainty

The Wilson intervals show the uncertainty associated with each sample proportion under a simple binomial-response model. For programming and problem-solving benefits, the intervals run from 51.7% to 75.4%; they include values only modestly above one half, which is consistent with the lack of significance after correcting four tests. For incomplete understanding, the interval is 65.9%–86.6%, while the overall positive-effect interval is 60.4%–82.6%. These ranges support the descriptive conclusion that incomplete understanding was common in this sample, but they do not repair non-probability sampling or self-report bias. The 50% comparison is a transparent statistical reference point chosen for this exploratory analysis; it should not be interpreted as the expected result in a no-AI comparison group.

The reconstructed means of 3.63, 3.78 and 3.81 suggest that responses leaned positive across the three learning-perception items. However, averaging a single ordinal item provides only a rough descriptive summary, and an identical average may hide different response distributions. The full category counts should therefore remain available alongside grouped agreement rates. Future analysis should predefine primary outcomes and directional hypotheses where justified, report effect estimates with confidence intervals, and correct for multiple comparisons when testing several outcomes.

7. MAJOR FINDINGS AND DISCUSSION

Five findings stand out. First, adoption is high (93.2%) and frequent use is common (69.5%). Second, ChatGPT is the leading tool in the sample (52.5%), while 44.1% report using AI across multiple programming activities. Third, 64.4% perceive improvements in both programming and problem-solving, but these proportions did not remain significant against the benchmark after multiple-testing correction. Fourth, incomplete understanding is the clearest risk signal: 78.0% reported using generated code without fully understanding it at least sometimes. Fifth, 72.9% rated AI's overall learning effect positively.

The results align with research showing that AI can provide timely assistance, explanations and task support, but do not ensure independent learning [1], [5], [6], [8], [12]. They also echo evidence that student perceptions need not translate into improved course performance [9]. Task efficiency and durable learning are different outcomes: a working solution does not prove that a student can explain, reproduce, debug, or transfer it without help. Prior knowledge and computational thinking may influence whether a learner uses AI to understand an approach or simply to retrieve an answer [2], [10].

The pattern in this dataset is not evidence that AI assistants are inherently harmful or beneficial. Rather, the survey shows that perceived usefulness and possible comprehension risk can occur together. Because the questionnaire did not directly measure verification behavior, the relationship between understanding and testing cannot be quantified. Likewise, the small number of non-users prevents a meaningful user-versus-non-user comparison. A follow-up design should distinguish tool access, type of help requested, verification actions, and independent performance so that the mechanism behind any observed learning difference can be examined.

A useful interpretation for educational practice is to define success at two levels: assisted task success and unaided learning. The first measures whether the student can complete a coding task with AI; the second measures whether the student can explain the logic, find a defect, adapt the code, or solve a related task without AI. Both can be valuable, but they should not be conflated. This distinction is consistent with recent studies of tutoring design and code-correction difficulty [4], [5], [8].

8. RECOMMENDATIONS

8.1 Students and educators

Use a "think first, ask second" workflow: outline the algorithm and attempt a solution before requesting hints or full code. Require learners to trace, test, explain and modify generated code; assess reasoning, error diagnosis, test-case design and justified changes rather than only program output. Use staged tasks independent plan, AI-assisted iteration, verification evidence, and reflection and maintain AI-free exercises or oral code-tracing tasks to evaluate retained independent skills. Teach AI literacy, including incorrect logic, hallucinated APIs, security and privacy risks, attribution, and disclosure expectations.

8.2 Education technology and industry

Learning-oriented assistants should offer progressive hints, ask students to justify an approach, communicate uncertainty, and prompt testing before revealing a complete solution. IDE and learning-platform integrations should support diff inspection, test execution, alternative comparison, and transparent use records with appropriate privacy safeguards. Product evaluation should measure task completion alongside delayed tests, debugging accuracy, code comprehension and unaided transfer. Institutions should distinguish permitted AI help from the competency being assessed, define disclosure requirements, and avoid relying on AI-content detectors alone.

For industry teams, the same approach can be applied to onboarding and code review: require developers to inspect generated diffs, run automated tests, review security-sensitive changes, and record why a suggested change was accepted or rejected. The objective is not to treat every AI interaction as a risk, but to ensure that code entering a shared system is understood, tested, and maintainable. Product teams should evaluate whether convenience features improve downstream quality and user skill, not just prompt response time or code-generation volume.

Organizations should also set rules for confidential source code, credentials, customer data, and student information before enabling external AI services. Training should include examples where generated code compiles but still contains logical, privacy, or security defects. A policy that combines approved tools, disclosure, access controls, code review, and independent testing is more actionable than a blanket instruction to "use AI responsibly."

9. LIMITATIONS

The sample is small, dominated by MCA students, and recruited by convenience sampling; it is not representative of all programming learners. Outcomes are self-reported and cross-sectional, so causal claims are not justified. Only aggregate charts were available, preventing association analysis between use frequency, experience, tool choice and outcomes. Programming and problem-solving constructs each rely on a single Likert item, preventing reliability assessment. Survey routing for Q4–Q6 and chart-reconstructed small categories must be checked against the original spreadsheet. The tests use a 50% benchmark rather than a control group and quantify response proportions, not actual ability.

10. FUTURE SCOPE

Future studies should collect de-identified respondent-level data and recruit larger, more diverse samples across BCA, MCA, B.Tech/BE and other computing programmes, using conditional routing for non-users. A stronger design should combine validated multi-item scales with pre/post coding, debugging, code-tracing and algorithm-decomposition tests. Longitudinal or randomized experiments could compare unrestricted code generation with hint-based tutoring while controlling for prior knowledge and computational thinking. Additional work should assess delayed retention, correction of AI errors, collaboration, academic integrity, data privacy and secure coding. Pre-registration and transparent reporting of effect sizes, confidence intervals, missing data and adjusted tests would improve reproducibility.

A practical evaluation protocol should report more than one score. Recommended measures include coding-test accuracy, debugging success, code-tracing accuracy, the ability to repair an AI-generated defect, delayed retention, performance on a related no-AI transfer problem, time-on-task, frequency of unverified acceptance, and student confidence. Researchers should also document model/version, prompting rules, whether full code was allowed, and how students verified outputs. These details would help distinguish gains caused by structured learning support from simple task-completion assistance.

11. CONCLUSION

This survey provides a preliminary snapshot of AI coding assistant use among 59 students. Use was widespread and perceptions were generally favorable, but incomplete understanding of generated code was also common. Exploratory tests found that incomplete-understanding and positive-overall-perception proportions differed from a 50% benchmark after correction; perceived programming and problem-solving benefits did not remain significant. The findings support learning-centered integration rather than unrestricted code substitution. AI assistance should be paired with independent reasoning, code explanation, testing, reflection and performance-based assessment. Larger studies with respondent-level

data and objective learning measures are required before drawing general conclusions about effectiveness.

APPENDIX A. SURVEY INSTRUMENT

The ten-item Google Forms instrument below is summarized from the supplied research report. It records categories and response formats only; no respondent-level answers are reproduced. This appendix supports transparency and replication of the survey procedure.

TABLE VI. SURVEY ITEMS AND RESPONSE FORMATS

Item	Question / response format
Q1	What is your current course? BCA / MCA / Others.
Q2	How much programming experience do you have? Less than 1 year / 1–2 years / 2–3 years / more than 3 years.
Q3	Do you use AI coding assistants for programming? Yes / No.
Q4	Which AI coding assistant do you use most? ChatGPT / GitHub Copilot / Gemini / Claude / Cursor / Other.
Q5	How often do you use AI coding assistants? Rarely / Sometimes / Often / Very often / Always.
Q6	What do you mainly use AI coding assistants for? Writing code / debugging errors / understanding code / learning concepts / solving programming problems / all of the above.
Q7	AI coding assistants have improved my programming skills. Five-point agreement scale.
Q8	AI coding assistants have improved my problem-solving ability. Five-point agreement scale.
Q9	How often do you use AI-generated code without fully understanding it? Never / Rarely / Sometimes / Often / Very often.
Q10	Overall, what has been the effect of AI coding assistants on your programming learning? Very negative / Negative / No significant effect / Positive / Very positive.

Reproducibility note: Before formal submission, compare the category counts in Tables III–IV with the original Google Forms export. In particular, confirm the counts reconstructed from small chart slices and determine why the tool and task questions display 59 responses although four respondents reported not using AI. Preserve the original export in a secure location, document any cleaning decisions, and avoid inferring individual-level relationships from aggregate charts.

REFERENCES

[1] 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, Art. no. 14, 2024, doi: 10.1186/s41239-024-00446-5.

[2] Y. Jing, H. Wang, X. Chen, and C. Wang, "What factors will affect the effectiveness of using ChatGPT to solve programming problems? A quasi-experimental study," Humanities and Social Sciences Communications, vol. 11, Art. no. 319, 2024, doi: 10.1057/s41599-024-02751-w.

[3] S. Groothuijsen, A. A. J. van den Beemt, J. C. C. Remmers, and L. W. van Meeuwen, "AI chatbots in programming education: Students' use in a scientific computing course and consequences for learning," Computers and Education: Artificial Intelligence, vol. 7, Art. no. 100290, 2024, doi: 10.1016/j.caeai.2024.100290.

[4] A. C. M. Yang, J.-Y. Lin, C.-Y. Lin, and H. Ogata, "Enhancing Python learning with PyTutor: Efficacy of a ChatGPT-based intelligent tutoring system in programming education," Computers and Education: Artificial Intelligence, vol. 7, Art. no. 100309, 2024, doi: 10.1016/j.caeai.2024.100309.

[5] D. Kohen-Vacs, M. Usher, and M. Jansen, "Integrating generative AI into programming education: Student perceptions and the challenge of correcting AI errors," International Journal of Artificial Intelligence in Education, vol. 35, pp. 3166–3184, 2025, doi: 10.1007/s40593-025-00496-4.

[6] P. Manorat, S. Tuarob, and S. Pongpaichet, "Artificial intelligence in computer programming education: A systematic literature review," Computers and Education: Artificial Intelligence, vol. 8, Art. no. 100403, 2025, doi: 10.1016/j.caeai.2025.100403.

[7] T.-C. Huang and H.-P. Tseng, "Learning, behavior, and pedagogy: A systematic review of generative AI use in programming education," International Journal of Information and Education Technology, vol. 16, no. 1, pp. 102–116, 2026, doi: 10.18178/ijiet.2026.16.1.2487.

[8] T. Noraset, A. Supratak, C. Ragkhitwetsagul, N. Worathong, and S. Tuarob, "Evaluating lab assistant chatbot on student learning and behaviors in a programming short course," Computers and Education: Artificial Intelligence, vol. 10, Art. no. 100527, 2026, doi: 10.1016/j.caeai.2025.100527.

[9] O. Gambo, O. J. Onifade, W. Olanrewaju-Smart, et al., "Impact of ChatGPT on learning outcomes and performance of students in computer programming courses: A mixed-method approach," Discover Education, vol. 5, Art. no. 164, 2026, doi: 10.1007/s44217-026-01142-4.

[10] S. Zhao, Z. Wang, T. Chen, et al., "Exploring the effect of computational thinking levels on students' learning performance, cognition, and behavior when using AI coding assistants," Educational Technology Research and Development, 2026, doi: 10.1007/s11423-026-10663-6.

[11] Y. Yalçın, E. Bakır-Yalçın, H. İslamoğlu, and Ö. F. Ursavaş, "Mapping research on generative artificial intelligence for programming education: A systematic review," Journal of Computer Assisted Learning, vol. 42, no. 5, Art. no. e70341, 2026, doi: 10.1002/jcal.70341.

[12] J. Nathaniel, S. S. Oyelere, J. Suhonen, and M. Tedre, "Literature review on the integration of generative AI in programming education," International Journal of Artificial Intelligence in Education, vol. 35, pp. 2724–2755, 2025, doi: 10.1007/s40593-025-00524-3.

[13] S. Chen and A. C. K. Cheung, "Effect of generative artificial intelligence on university students' learning outcomes: A systematic review and meta-analysis," Educational Research Review, vol. 49, Art. no. 100737, 2025, doi: 10.1016/j.edurev.2025.100737.