|| Volume 9 || Issue 11|| November 2025 || ISSN (Online) 2456-0774
INTERNATIONAL JOURNAL OF ADVANCE SCIENTIFIC RESEARCH
AND ENGINEERING TRENDS

Large Language Models as Programming Tutors in Computer Science Education: A Systematic Review of Applications, Learning Effectiveness, Error Patterns, and Academic Integrity

Smt.Gosavi Shweta Vishnu

Assistant Professor, (HOD) BCA Department, NDMVP Samaj's S.V.K.T.College of Arts,Science and Commerce, Deolali Camp, Nashik , Maharashtra- 422401

Abstract: This review synthesizes empirical evidence on how large language models (LLMs) are used as programming tutors across five key applications: code explanation, debugging, formative feedback, exercise/test generation, and assessment. Following PRISMA 2020 guidance, studies from 2020–2025 were screened for relevance to programming education, with outcomes on learning effectiveness, error patterns, and academic integrity synthesized via narrative and thematic methods. The mapping of recent studies indicates rapid uptake of LLMs, mixed but promising learning outcomes, recurring failure modes in logic and specification adherence, and emerging academic integrity risks alongside mitigation practices. Implications are provided for course design in Object-Oriented Programming (OOP) contexts, assessment practices, and departmental policy, with identified gaps and recommendations for standardized evaluation.

Keywords: *Large language models; programming education; formative feedback; exercise/test generation; assessment; academic integrity; object-oriented programming; Java; C++.*

I.INTRODUCTION:

The rapid emergence of Large Language Models (LLMs), such as ChatGPT and GitHub Copilot, is fundamentally transforming the landscape of computer science (CS) education (Prather et al., 2023; Fernandez & Cornell, 2024; Lyu et al., 2024). While these tools offer promising opportunities to function as programming tutors including the provision of personalized code explanation and debugging assistance their use presents significant pedagogical and ethical challenges (Becker et al., 2023; Cambaz & Zhang, 2024; Lau & Guo, 2023). The core challenge lies in determining how to ethically and effectively integrate generative AI when its capability to generate correct solutions threatens traditional assessment validity, particularly for take-home coding tasks and auto-gradable CS1 exercises (Lyu et al., 2024; Prather et al., 2023; Becker et al., 2023).Computing instructors urgently need clear guidance to navigate the shifting demands of the Generative AI era (Cambaz & Zhang, 2024; Prather et al., 2023).

This guidance must first focus on helping educators "clearly delineate what qualifies as acceptable use of AI and what is deemed as plagiarism" (Azoulay et al., 2025) by revising academic integrity policies (Lau & Guo, 2023; Cambaz & Zhang, 2024).Actionable frameworks are required to shift pedagogical emphasis towards higher-order skills, suggesting "a shift in emphasis towards code reading and evaluating rather than code generation" (Becker et al., 2023) and integrating "Generative AI literacy" into the curriculum (Lyu et al., 2024; Fernandez & Cornell, 2024).Educators need support in teaching students how to formulate prompts properly and critically evaluate the often-inconsistent responses LLMs produce (Fernandez & Cornell, 2024; Lyu et al., 2024).Without "specific actionable guidelines for educators to ensure safe student interaction" (Cambaz & Zhang, 2024), the computing education community risks being unprepared for this technological change. This review addresses this need by consolidating findings to answer five research questions (RQs) concerning LLM applications, learning effectiveness, error patterns, academic integrity, and contextual moderators.

Background

The ascent of LLMs is extending educational NLP capabilities to complex code generation and reasoning tasks (Raihan et al., 2025; Prather et al., 2023). Systematic reviews confirm that LLMs are being rapidly integrated as programming tutors, predominantly in university-level introductory courses (Haruto et al., 2025; Raihan et al., 2025). The models exhibit impressive performance; for example, GPT-3.5 and GPT-4 achieved a high success rate, solving 94.4% to 95.8% of introductory Python programming challenges from the CodingBat benchmark (Kiesler & Schiffner, 2023).This high performance on functional tasks belies the fact that the tools are consistently characterized by intrinsic reliability issues that impede their function as comprehensive assistants (Pirzado et al., 2024).

This review will present a detailed taxonomy of these issues, including logic errors, specification drift, hallucinations, and overconfidence (Hellas et al., 2023; MacNeil et al., 2024; Kiesler & Schiffner, 2023). When acting as programming tutors, LLM-generated explanations are often rated as "significantly easier to understand and more accurate summaries of code than student-created explanations" (Leinonen et al., 2023, p. 128). Yet, this benefit is tempered by error profiles; for instance, LLMs often provide overconfident wrong answers, which novices struggle to verify (Kiesler & Schiffner, 2023; Lyu et al., 2024).Comprehensive surveys cite risks such as student overreliance and subsequent skill atrophy as major challenges (Haruto et al., 2025; Lyu et al., 2024).

II.METHODS

Protocol

The reporting methodology adheres to the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 statement, including use of the PRISMA 2020 checklist and flow diagram to transparently describe identification, screening, eligibility, and inclusion. The study selection flow is depicted in Figure 1 (PRISMA 2020), which summarizes all counts across stages for this review and underpins the final inclusion of 76 studies .Complete search strategies and screening procedures are documented consistent with established guidance, and the figure is cross-referenced in Results Section 4.1 to align with PRISMA item 16a.

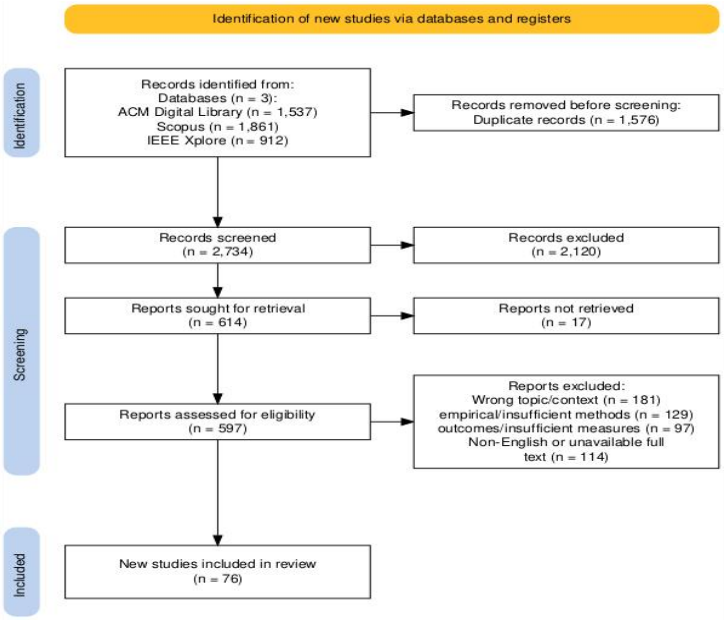

Figure 1 PRISMA flow image

Sources and Timeframe

Information sources comprised three core bibliographic databases covering computer science and education research: ACM Digital Library, Scopus, and IEEE Xplore. The time window for eligible records was 2020–2025 to capture the post-LLM era in programming education, matching prior reviews while focusing on the period of rapid classroom uptake. Per PRISMA 2020, full search strings and the last-searched dates for each database are provided in the review materials to ensure reproducibility and auditability.

Inclusion and Exclusion Criteria

Studies were included if they reported empirical or design-evaluation evidence in programming education with outcomes in at least one of five categories: explanation, debugging, formative feedback, exercise/test generation, or assessment, situated in higher education or late K-12 and published in English. Studies were excluded if they were pure systems papers without educational evaluation, opinion pieces without empirical data, or duplicate records, consistent with pre-specified eligibility criteria aligned to PRISMA 2020 Item 5.

Screening and Extraction

Screening proceeded in two stages title/abstract followed by full-text assessment conducted by multiple independent reviewers with conflicts resolved by consensus, in line with PRISMA 2020 reporting expectations for selection processes. A structured extraction schema captured study context, application category, model, outcomes, error patterns, and academic-integrity considerations to support narrative and thematic synthesis.

Synthesis Approach

Synthesis employed narrative and thematic integration with structured vote counting by outcome direction, reflecting heterogeneity that precluded meta-analysis.Pre-specified subgroup analyses compared outcomes by OOP focus, programming language, course level, model version, and prompting strategy, with the full selection flow summarized in Figure 1 for PRISMA completeness.

III.RESULTS

.Study Selection (PRISMA Flow)

The study selection process followed PRISMA 2020 guidelines (Haruto et al., 2025). Initial identification of records across multiple reviews ranged from 154 to over 6,000 (Pereira & Mello, 2025; Pirzado et al., 2024). After removing duplicates, title and abstract screening further narrowed the corpus (Raihan et al., 2025; Cambaz & Zhang, 2024). Following a full-text assessment, a final set of studies was included for synthesis, ranging from 21 to 125 studies across the reviews (Cambaz & Zhang, 2024; Raihan et al., 2025).

RQ1: Applications of LLMs as Programming Tutors

LLMs are reshaping programming instruction across five key application categories. For students, the primary uses are explanation and real-time debugging (Haruto et al., 2025; Lyu et al., 2024). LLM-generated explanations are often rated as "significantly easier to understand and more accurate" than student-created ones (Leinonen et al., 2023, p. 128). As debugging assistants, tools like CodeTutor are heavily used for syntax comprehension and error correction (Lyu et al., 2024).

For instructors, LLMs are used to automate formative feedback and assessment (Azaiz et al., 2024; Cambaz & Zhang, 2024). Models can produce personalized feedback, though with mixed accuracy.One study of GPT-4 Turbo found only 52% of feedback was fully correct (Azaiz et al., 2024, p. 30). For assessment, high success rates on introductory tasks have led to pedagogical shifts like "Prompt Problems," where the task is to write the prompt itself (Denny et al., 2024). Instructors use LLMs for exercise/test generation to create varied assignments (Sarsa et al., 2022).

RQ2: Learning Effectiveness and Student Outcomes

Evidence on learning effectiveness is mixed. A semester-long study found that students using an LLM tutor (CodeTutor) showed significant improvements in final scores compared to a

control group (Lyu et al., 2024). Another study in an OOP course that redesigned assignments to be more complex found no statistically significant difference in grades between groups with and without LLM access, suggesting that pedagogical interventions can mitigate certain impacts (Kosar et al., 2024, as cited in Azoulay et al., 2025, p. 24).

Effectiveness is strongly moderated by course level and task type. In introductory courses, LLMs excel at self-contained problems (Kiesler & Schiffner, 2023), novices themselves struggle to write effective prompts (Nguyen et al., 2024). In advanced courses, performance declines sharply,ChatGPT's correctness rate dropped from 75.66% in introductory data science to just 22.9% in advanced courses (Shen et al., 2024, p. 140).

RQ3: Recurring Error Patterns and Failure Modes

The utility of LLMs as programming tutors is undermined by recurring error patterns (Pirzado et al., 2024). A primary model limitation is the generation of logic errors (MacNeil et al., 2024). This is particularly acute in OOP contexts, where LLMs violate best practices, such as using instanceof for type checking (Pereira Cipriano & Alves, 2023, p. 119). Another critical error is hallucination.One study found that in 48% of responses to student help requests, the LLM "reported on issues that did not actually exist" (Hellas et al., 2023, p. 115).

Interaction failures are also common. Specification drift occurs when novices write unclear prompts (Nguyen et al., 2024). LLMs exhibit an over-completion bias, providing a full solution even when explicitly asked for a hint (Hellas et al., 2023, p. 96). Mitigation strategies center on prompt engineering and teaching critical evaluation of AI output (Shen et al., 2024; Fernandez &

RQ4: Academic Integrity and Assessment Practices

The ability of LLMs to solve introductory assignments poses a fundamental threat to academic integrity (Lau & Guo, 2023; Lyu et al., 2024). Current detection methods are largely ineffective.Traditional plagiarism software is easily circumvented (Lau & Guo, 2023, p. 6) and specialized AI detectors are unreliable (Azoulay et al., 2025).

Educators are pursuing assessment redesign (Lau & Guo, 2023). Key patterns include:

- **Authentic Tasks:** Using bespoke starter code or local context that LLMs lack (Lau & Guo, 2023, p. 5).
- **Process-Based Assessment:** Grading development history and design logs rather than just the final product (Lau & Guo, 2023, p. 7).
- **Oral Defenses:** Requiring students to verbally explain their code to verify understanding, a strategy shown to be effective (Azoulay et al., 2025, p. 41; Kosar et al., 2024, as cited in Azoulay et al., 2025).

RQ5: Contextual Moderators Influencing Outcomes

LLM performance is influenced by several moderators. Model version is critical.Newer models like GPT-4 consistently outperform earlier versions in accuracy and consistency (Azaiz et al., 2024; MacNeil et al., 2024). Prompt scaffolding is equally important, as performance improves with detailed prompts (Shen et al., 2024), but novices "struggle to write natural language prompts for LLMs" (Nguyen et al., 2024, p. 10). Instructor oversight is essential, as the pedagogical value depends on "thoughtful integration strategies that combine automation with human oversight" (Haruto et al., 2025, p. 305) , performance weakens with non-English languages (Pirzado et al., 2024) and non-textual inputs (Ooh et al., 2025, as cited in Savelka et al., 2024).

IV.DISCUSSION

The findings indicate that while LLMs provide scalable support as programming tutors, their integration into OOP contexts requires careful scaffolding to translate this support into durable learning gains (Haruto et al., 2025; Prather et al., 2023). LLMs' strong performance on CS1-style Python problems coexists with brittleness on multi-class OOP tasks, where adherence to encapsulation and design constraints remains inconsistent (Kiesler & Schiffner, 2023; Pereira Cipriano & Alves, 2023). This performance duality is a central challenge.

To counter the observed patterns of logic errors and specification drift, the most critical adaptations are the enforcement of rigorous design principles: scaffolding, prompt literacy, and verification workflows (Liffiton et al., 2023; Shen & Ai, 2024; Lau & Guo, 2023). Scaffolding requires integrating LLMs into tools with "guardrails" to prevent them from outputting complete solutions (Liffiton et al., 2023). Prompt literacy must be explicitly taught, helping students decompose complex OOP tasks into structured steps (Shen & Ai, 2024) Instructors must enforce verification workflows, redesigning assignments to focus on code reading and critique, thereby reinforcing the cognitive skills necessary to audit AI output (Lau & Guo, 2023; Fernandez & Cornell, 2024).

Quality Appraisal and Limitations

The research landscape concerning LLMs in CS education is marked by significant methodological and contextual heterogeneity (Raihan et al., 2025). While systematic reviews categorize most studies as having "High" or "Moderate" rigor (Pereira & Mello, 2025). The diversity in designs makes direct comparison difficult (Haruto et al., 2025). The research is also heavily skewed toward introductory university courses in the Global North, raising concerns about generalizability (Haruto et al., 2025, p. 336).

Several common threats to validity are present. The rapid evolution of LLMs poses a major threat to temporal validity (Azaiz et al., 2024, p. 33). The inherent stochasticity of LLMs challenges reproducibility (Azaiz et al., 2024, p. 33; MacNeil et al., 2024, p. 188). Many qualitative studies also face threats from rater bias (Hellas et al., 2023, p. 96). These factors combined with a lack of standardized metrics, make a statistical meta-analysis largely infeasible (Gaitantzi & Kazanidis, 2025).

Implications and Recommendations

For Instructors:

- **Teach Prompt Scaffolding and Literacy:** Incorporate dedicated activities to teach students how to formulate effective prompts by breaking down problems and providing context (Lyu et al., 2024; Fernandez & Cornell, 2024).
- **Establish Verification Workflows:** Design assignments that require students to critically evaluate LLM-generated output for errors or poor design (e.g., requiring students to submit a report analyzing an LLM's solution for adherence to SOLID principles) (Cambaz & Zhang, 2024; Fernandez & Cornell, 2024).
- **Implement Timed Formative Feedback:** Use LLMs to provide immediate feedback, but integrate them within tools that have "guardrails" to prevent the generation of complete solutions (Azaiz et al., 2024; Liffiton et al., 2023).

For Assessment Leads:

- **Design Authentic, AI-Proof Tasks:** Revamp traditional assignments by incorporating local context or multi-modal inputs (e.g., UML diagrams) that challenge the LLM's general training data (Lau & Guo, 2023; Azoulay et al., 2025).
- **Emphasize Process Logs and Documentation:** Shift grading focus toward process-based assessment, requiring students to submit iterative documentation like version control history (Lau & Guo, 2023).
- **Incorporate Oral Defenses and Presentations:** Mandate oral examinations where students must verbally explain their code logic to ensure genuine comprehension (Azoulay et al., 2025; Lau & Guo, 2023).
- **Calibrate and Articulate Policies:** Develop clear, subject-specific academic integrity policies that define permitted and prohibited uses of LLMs (Lau & Guo, 2023; Azoulay et al., 2025).

For Departments:

- **Mandate Generative AI Literacy Training:** Integrate AI literacy as a core component of the CS curriculum for both students and faculty (Lyu et al., 2024; Haruto et al., 2025).
- **Promote Hybrid LLM-Human Systems:** Invest in developing localized LLM tools that combine the scalability of AI with essential human oversight (Pirzado et al., 2024; Haruto et al., 2025).
- **Advocate for Watermarking Technology:** Collaborate with LLM developers to mandate the use of invisible watermarks or digital fingerprints in AI-generated code for future traceability (Azoulay et al., 2025).

For Researchers:

- **Develop Standardized Reporting Templates:** Adopt common templates that explicitly detail the model version, parameters, and date of data collection to ensure reproducibility (Azaiz et al., 2024; Pereira Cipriano & Alves, 2023).
- **Create Shared and Diverse Benchmarks:** Develop public benchmark datasets that test LLM capabilities on advanced topics like OOP best practices and complex algorithmic reasoning (Shen et al., 2024; Pereira Cipriano & Alves, 2023).
- **Focus on Long-Term Impact:** Prioritize longitudinal studies that assess the long-term effects of LLM usage on critical thinking and problem-solving skills (Haruto et al., 2025; Lyu et al., 2024).

V.CONCLUSION

Across code explanation, debugging, formative feedback, exercise/test generation, and assessment, LLMs demonstrate substantial potential as programming tutors but require deliberate pedagogy and integrity safeguards to realize consistent learning benefits. While newer models show improved accuracy, effectiveness is undermined by persistent error patterns particularly when moving beyond simple syntactic tasks to complex design paradigms like OOP. LLMs may pass functional tests but often fail to adhere to fundamental OOP best practices. This gap fuels academic integrity risks, as students may become over-reliant on tools that provide functional but pedagogically poor solutions.Instructors must shift toward integrity-aware assessment in OOP-focused courses by designing complex assignments and integrating process-based evaluation. For the research community, standardized evaluation practices and shared benchmarks are pivotal for responsible and effective integration.

VI.REFERENCES

1. Azaiz, I., Kiesler, N., & Strickroth, S. (2024). Feedback-Generation for Programming Exercises With GPT-4. *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, 31–37. https://doi.org/10.1145/3649217.3653594
2. Azoulay, R., Hirst, T., & Reches, S. (2025). Large Language Models in Computer Science Classrooms: Ethical Challenges and Strategic Solutions. *Applied Sciences*, *15*(4), 1793. https://doi.org/10.3390/app15041793
3. Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 500–506. https://doi.org/10.1145/3545945.3569759

4. Cambaz, D., & Zhang, X. (2024). Use of AI-driven Code Generation Models in Teaching and Learning Programming: A Systematic Literature Review. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 172–178. https://doi.org/10.1145/3626252.3630958
5. Cipriano, B. P., & Alves, P. (2023). GPT-3 vs Object Oriented Programming Assignments: An Experience Report. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 61–67. https://doi.org/10.1145/3587102.3588814
6. Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., & Reeves, B. N. (2024). Prompt Problems: A New Programming Exercise for the Generative AI Era. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 296–302. https://doi.org/10.1145/3626252.3630909
7. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139
8. Fernandez, A. S., & Cornell, K. A. (2024). CS1 with a Side of AI: Teaching Software Verification for Secure Code in the Era of Generative AI. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 345–351. https://doi.org/10.1145/3626252.3630817
9. Gaitantzi, A., & Kazanidis, I. (2025). The Role of Artificial Intelligence in Computer Science Education: A Systematic Review with a Focus on Database Instruction. *Applied Sciences*, *15*(7), 3960. https://doi.org/10.3390/app15073960
10. Haindl, P., & Weinberger, G. (2024). Students' Experiences of Using ChatGPT in an Undergraduate Programming Course. *IEEE Access*, *12*, 43519–43529. https://doi.org/10.1109/ACCESS.2024.3380909
11. Haruto, S., Nnadi, L. C., & Yutaka, W. (2025a). Systematic Review of Large Language Model Applications in Programming Education. In H. Fujita, A. Hernandez-Matamoros, & Y. Watanobe (Eds.), *Frontiers in Artificial Intelligence and Applications*. IOS Press. https://doi.org/10.3233/FAIA250512
12. Haruto, S., Nnadi, L. C., & Yutaka, W. (2025b). Systematic Review of Large Language Model Applications in Programming Education. In H. Fujita, A. Hernandez-Matamoros, & Y. Watanobe (Eds.), *Frontiers in Artificial Intelligence and Applications*. IOS Press. https://doi.org/10.3233/FAIA250512
13. Hellas, A., Leinonen, J., Sarsa, S., Koutcheme, C., Kujanpää, L., & Sorva, J. (2023). Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*, 93–105. https://doi.org/10.1145/3568813.3600139
14. Jacobs, S., & Jaschke, S. (2024). Evaluating the Application of Large Language Models to Generate Feedback in Programming Education. *2024 IEEE Global Engineering Education Conference (EDUCON)*, 1–5. https://doi.org/10.1109/EDUCON60312.2024.10578838
15. Joshi, I., Budhiraja, R., Dev, H., Kadia, J., Ataullah, M. O., Mitra, S., Akolekar, H. D., & Kumar, D. (2024). ChatGPT in the Classroom: An Analysis of Its Strengths and Weaknesses for Solving Undergraduate Computer Science Questions. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 625–631. https://doi.org/10.1145/3626252.3630803
16. Kiesler, N., & Schiffner, D. (2023). *Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments* (No. arXiv:2308.08572). arXiv. https://doi.org/10.48550/arXiv.2308.08572
17. Koutcheme, C., Dainese, N., Sarsa, S., Hellas, A., Leinonen, J., Ashraf, S., & Denny, P. (2025). Evaluating Language Models for Generating and Judging Programming Feedback. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 624–630. https://doi.org/10.1145/3641554.3701791
18. Kulangara, K. J. (n.d.). *Designing and Building a Platform for Teaching Introductory Programming Supported by Large Language Models*.
19. Kumar, N. A., & Lan, A. (2024). *Using Large Language Models for Student-Code Guided Test Case Generation in Computer Science Education* (No. arXiv:2402.07081). arXiv. https://doi.org/10.48550/arXiv.2402.07081
20. Lau, S., & Guo, P. (2023). From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. *Proceedings of the 2023 ACM Conference on International Computing Education Research V.1*, 106–121. https://doi.org/10.1145/3568813.3600138
21. Leinonen, J., Denny, P., MacNeil, S., Sarsa, S., Bernstein, S., Kim, J., Tran, A., & Hellas, A. (2023).

Comparing Code Explanations Created by Students and Large Language Models. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 124–130. https://doi.org/10.1145/3587102.3588785

22. Lyu, W., Wang, Y., Chung, T. (Rachel), Sun, Y., & Zhang, Y. (2024). Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study. *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, 63–74. https://doi.org/10.1145/3657604.3662036
23. Lyu, W., Wang, Y., Tingting, Chung, Sun, Y., & Zhang, Y. (2024). Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study. *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, 63–74. https://doi.org/10.1145/3657604.3662036
24. Macneil, S., Denny, P., Tran, A., Leinonen, J., Bernstein, S., Hellas, A., Sarsa, S., & Kim, J. (2024). Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models. *Proceedings of the 26th Australasian Computing Education Conference*, 11–18. https://doi.org/10.1145/3636243.3636245
25. Nguyen, S., Babe, H. M., Zi, Y., Guha, A., Anderson, C. J., & Feldman, M. Q. (2024). How Beginning Programmers and Code LLMs (Mis)read Each Other. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–26. https://doi.org/10.1145/3613904.3642706
26. Pereira, A. F., & Ferreira Mello, R. (2025). A Systematic Literature Review on Large Language Models Applications in Computer Programming Teaching Evaluation Process. *IEEE Access*, *13*, 113449–113460. https://doi.org/10.1109/ACCESS.2025.3584060
27. Petrovska, O., Clift, L., Moller, F., & Pearsall, R. (2024). Incorporating Generative AI into Software Development Education. *Proceedings of the 8th Conference on Computing Education Practice*, 37–40. https://doi.org/10.1145/3633053.3633057
28. Piccolo, S. R., Denny, P., Luxton-Reilly, A., Payne, S. H., & Ridge, P. G. (2023). Evaluating a large language model's ability to solve programming exercises from an introductory bioinformatics course. *PLOS Computational Biology*, *19*(9), e1011511. https://doi.org/10.1371/journal.pcbi.1011511
29. Pirzado, F. A., Ahmed, A., Mendoza-Urdiales, R. A., & Terashima-Marin, H. (2024). Navigating the Pitfalls: Analyzing the Behavior of LLMs as a Coding Assistant for Computer Science Students—A Systematic Review of the Literature. *IEEE Access*, *12*, 112605–112625. https://doi.org/10.1109/ACCESS.2024.3443621
30. Raihan, N., Goswami, D., Puspo, S. S. C., Siddiq, M. L., Newman, C., Ranasinghe, T., Santos, J. C. S., & Zampieri, M. (2025). On the performance of large language models on introductory programming assignments. *Journal of Intelligent Information Systems*. https://doi.org/10.1007/s10844-025-00968-y
31. Raihan, N., Siddiq, M. L., Santos, J. C. S., & Zampieri, M. (2024a). *Large Language Models in Computer Science Education: A Systematic Literature Review* (No. arXiv:2410.16349). arXiv. https://doi.org/10.48550/arXiv.2410.16349
32. Raihan, N., Siddiq, M. L., Santos, J. C. S., & Zampieri, M. (2024b). *Large Language Models in Computer Science Education: A Systematic Literature Review* (No. arXiv:2410.16349). arXiv. https://doi.org/10.48550/arXiv.2410.16349
33. Raihan, N., Siddiq, M. L., Santos, J. C. S., & Zampieri, M. (2025a). Large Language Models in Computer Science Education: A Systematic Literature Review. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 938–944. https://doi.org/10.1145/3641554.3701863
34. Raihan, N., Siddiq, M. L., Santos, J. C. S., & Zampieri, M. (2025b). Large Language Models in Computer Science Education: A Systematic Literature Review. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 938–944. https://doi.org/10.1145/3641554.3701863
35. Rajala, J., Hukkanen, J., Hartikainen, M., & Niemelä, P. (2023). "\"Call me Kiran\" – ChatGPT as a Tutoring Chatbot in a Computer Science Course". *26th International Academic Mindtrek Conference*, 83–94. https://doi.org/10.1145/3616961.3616974
36. Shahzad, T., Mazhar, T., Tariq, M. U., Ahmad, W., Ouahada, K., & Hamam, H. (2025). A comprehensive review of large language models: Issues and solutions in learning environments. *Discover Sustainability*, *6*(1), 27. https://doi.org/10.1007/s43621-025-00815-8
37. Shen, Y., Ai, X., Soosai Raj, A. G., Leo John, R. J., & Syamkumar, M. (2024). Implications of ChatGPT for Data Science Education. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 1230–1236. https://doi.org/10.1145/3626252.3630874
38. Tanay, B., Arinze, L., Joshi, S., Davis, K., & Davis, J. (2024). An Exploratory Study on Upper-Level Computing Students' Use of Large Language Models as Tools in a Semester-Long Project. *2024 ASEE Annual Conference & Exposition Proceedings*, 46557. https://doi.org/10.18260/1-2--46557