A Survey of LLM-Based Applications in Programming Education: Balancing Automation and Human Oversight

Griffin Pitts* **and Anurata Prabha Hridi***
North Carolina State University
{wgpitts,aphridi}@nscu.edu

Arun-Balajiee Lekshmi-Narayanan
University of Pittsburgh
arl122@pitt.edu

Abstract

Novice programmers benefit from timely, personalized support that addresses individual learning gaps, yet the availability of instructors and teaching assistants is inherently limited. Large language models (LLMs) present opportunities to scale such support, though their effectiveness depends on how well technical capabilities are aligned with pedagogical goals. This survey synthesizes recent work on LLM applications in programming education across three focal areas: formative code feedback, assessment, and knowledge modeling. We identify recurring design patterns in how these tools are applied and find that interventions are most effective when educator expertise complements model output through human-in-the-loop oversight, scaffolding, and evaluation. Fully automated approaches are often constrained in capturing the pedagogical nuances of programming education, although human-in-the-loop designs and course-specific adaptation offer promising directions for future improvement. Future research should focus on improving transparency, strengthening alignment with pedagogy, and developing systems that flexibly adapt to the needs of varied learning contexts.

1 Introduction

Introductory programming courses serve as critical gateways to computer science and related STEM careers (Whalley et al., 2020), yet they present unique pedagogical challenges that contribute to high attrition rates (Petersen et al., 2016). Students must simultaneously master syntax, develop computational thinking skills, and learn to debug complex logical errors, creating cognitive demands that often overwhelm novices. Success in these courses often depends on access to timely, targeted interventions, e.g., feedback, explanations, and guided problem-solving, that address individual learning gaps (Marwan et al., 2020; Messer et al., 2023). These personalized interventions are especially important because students exhibit varied practice behaviors, depending on their inclination toward problem-solving and/or example exploration (Poh et al., 2025). Traditionally, this kind of support has been provided through human instruction, with teaching assistants (TAs) guiding problem-solving strategies, offering debugging help during office hours, and providing detailed feedback on assignments (Markel and Guo, 2021). However, the scalability of this human-centered model is limited. Large enrollment courses, increasingly common in CS education, strain the capacity of instructional staff to provide individualized attention (Ahmed et al., 2025). Additionally, students may delay seeking help when they anticipate long wait times for TA feedback (Gao et al., 2023), and TAs themselves face heavy workloads from simultaneous requests during office hours (Gao et al., 2022).

Large language models (LLMs) have emerged as promising tools to automate aspects of programming education support by addressing the core challenges novice programmers face, including debugging code errors (Lahtinen et al., 2005), repairing faulty code (Javier, 2021), obtaining timely feedback, and mastering foundational concepts to work with programming problems (Ahmad and Ghazali, 2020; Rivers et al., 2016). Recent advances in natural language processing now make it possible to generate context-aware feedback, offer real-time debugging support, and adapt explanations to a student's skill level (Yousef et al., 2025; Zhong et al., 2024; Chen et al., 2024; Lui et al., 2024). Early work also shows the potential of Human–AI collaboration, where LLM outputs are refined or guided by educators to better align with pedagogical goals (Hassany et al., 2024). Though promising, it is still unclear what best practices should guide the integration of LLMs into programming education and how responsibilities should be balanced

*Both authors contributed equally to this work.

Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP), pages 255–262

between automation and human expertise.

The presented paper addresses this gap by reviewing recent applications of LLMs in programming education, focusing on how they are applied, the challenges that arise in practice, and the opportunities to align technical advances more closely with pedagogy.

2 Methodology and Paper Selection

To survey research on the use of LLMs in programming education, we reviewed publications across leading venues in computing education, HCI, and NLP, such as `SIGCSE`, `ITiCSE`, `ICER`, `LAK`, `EDM`, `CHI`, `EMNLP`, and related workshops. We focused on the 2021-2025 period, when LLM-driven systems first began to appear in educational settings.

From the surveyed literature, we identified three focal areas in which LLMs are being applied: formative code feedback, assessment, and knowledge modeling. Each aligns with a central pedagogical challenge in introductory programming courses, supporting student learning at scale in contexts where one-on-one guidance is difficult to provide. Within formative code feedback, we distinguish between approaches that generate hints and explanations to help students identify and understand their errors, and those that produce corrected code as examples or candidate repairs. Assessment focuses on grading and providing evaluative feedback at scale, while knowledge modeling seeks to represent what students know and how they progress in order to give instructors actionable insights through learning analytics. The distribution of reviewed papers across these areas is shown in Table 1.

Topic	Selected Papers
Formative Code Feedback	20
Assessment	14
Knowledge Modeling	8

Table 1: Number of papers reviewed across three focal areas.

3 LLM Usage in Formative Code Feedback

Debugging assistance is one of the most common reasons to seek help in programming courses, and improvements in code correctness after such support can substantially boost short-term performance (Gao et al., 2022). However, the scalability limits of human-led help sessions have prompted researchers to explore how LLMs can extend this support in programming education. LLM-based feedback on student code is typically delivered in two complementary forms. In some cases, models generate explanations, hints, or scaffolding that help students locate and reason about their own programming mistakes. In others, models produce corrected code directly, offering candidate repairs or examples that students can study and compare against their own solutions. Both approaches aim to reduce the bottleneck of human-provided feedback, although they differ in the amount of agency they leave with the learner.

A large body of work has explored how LLMs can generate hints and explanations to guide student reasoning. Early evaluations of open-source models suggest that they can assist with syntax and minor semantic issues but continue to struggle with more complex bug localization and multi-line logic errors (Majdoub and Ben Charrada, 2024). To address these challenges, researchers have developed systems that focus on scaffolding student reasoning. *BugSpotter*, for instance, combines static analysis with LLM reasoning to create interactive debugging exercises for low-level programming languages (Padurean et al., 2025). Iterative self-debugging loops have been proposed, where models run their generated code, collect execution feedback, and refine patches in multiple passes (Chen et al., 2024). Adaptive scaffolding systems further extend these capabilities by monitoring learner progress and providing timely hints or explanations to break complex reasoning into manageable steps (Oli, 2024). Although such advances improve the accuracy and relevance of LLM-supported feedback, human oversight remains necessary to ensure that outputs align with pedagogical goals (Zubair et al., 2025). For example, *CodeAid* (Kazemitabaar et al., 2024) found that while LLMs can accelerate support for students, direct answers without educator scaffolding risk undermining learning, highlighting the role of instructors in contextualizing automated feedback.

Another cluster of research investigates improving the clarity of error explanations to support student comprehension. Fine-tuned LLMs have demonstrated the ability to produce clearer, context-sensitive error messages, improving novice problem-solving (Vassar et al., 2024; Leinonen et al., 2023). Comparative analyses of human and model debugging strategies reveal differences in reasoning patterns, pointing to opportunities for de-

signing AI-assisted tools that nudge learners toward more expert-like approaches (MacNeil et al., 2024). Related work on prompt engineering and explainability techniques, such as step-by-step runtime verification, has also shown promise for improving the readability of error messages and fostering trust in human–AI collaboration (Zhong et al., 2024; Hoq et al., 2025; Kang et al., 2025). Expanding further, researchers have also applied LLMs to code quality feedback, detecting issues such as misleading variable and function identifiers in novice code (Řechtáčková et al., 2025).

Beyond generating hints and explanations, a growing body of work explores producing corrected code and worked examples that students can study alongside their own solutions. For instance, LLMs have been applied to generate code-tracing questions for introductory courses, producing diverse and pedagogically aligned items (Fan et al., 2023). Recent work demonstrates how LLMs can generate worked examples that help students learn strategies and better understand solution approaches (Sarsa et al., 2022). Research also shows that students learn from the process of spotting and fixing code errors (Koutcheme et al., 2024a), and that these skills strongly predict learning outcomes and course success (Gao et al., 2022, 2023).

To support this process, automated program repair (APR) systems have targeted syntactic and semantic errors in student submissions, with LLMs broadening the scope of repairs to be more context-sensitive and benchmarked transparently (Jiang et al., 2023). Examples include *PyDex* (Zhang et al., 2024), which generates accurate leverages LLMs to automatically generate accurate fixes for common novice errors in Python assignments (Zhang et al., 2024); *COAST*, a multi-agent framework that coordinates detection, repair, and verification while synthesizing debugging datasets (Yang et al., 2025); and *RepairLLaMA*, which incorporates repair-aware representations and parameter-efficient fine-tuning to outperform vanilla prompting on standard APR benchmarks (Silva et al., 2025).

While these advances demonstrate the technical potential of LLMs for formative programming support, their educational value depends on when and how the feedback is delivered. Automated fixes that come too early, solve too much of the problem, or present complete answers can short-circuit the learning process by removing opportunities for students to reason through their own errors. In contrast, tools that generate hints, scaffold reasoning, and explain errors without directly supplying solutions are better aligned with pedagogical goals. The challenge is therefore not only improving accuracy on complex bugs but also designing systems that adapt the level of support to the learner's needs. Emerging best practices point toward hybrid approaches, where LLMs address routine or surface-level issues and generate scalable practice materials, while human educators provide context, address deeper misconceptions, and guide students toward lasting debugging strategies.

4 LLM Usage in Assessment

With the increasing availability of LLMs in education, there are now provisions for the use of automated teaching assistants (TAs) in assessments, particularly in programming courses where grading is frequent and labor-intensive (Mehta et al., 2023). Early evaluations benchmarked the ability of LLMs to provide such feedback, demonstrating that even in zero-shot configurations, they can produce rubric-aligned evaluations with moderate agreement to human graders (Yeung et al., 2025; Silva and Costa, 2025). These findings position LLMs as viable tools for scalable deployment, reducing the need for extensive rule-based assessment design. For example, *ABScribe* (Reza et al., 2024) demonstrates how LLMs can support human-AI co-writing by generating and organizing multiple text variations, easing TA workload and improving revision efficiency.

However, meta-analytic perspectives caution that such systems inherit biases from training data and require prompt and rubric alignment to meet course-specific standards (Messer et al., 2023). In classroom settings, results have been mixed. In a study involving more than 1,000 students, GPT-4 reliably evaluated straightforward and clear-cut submissions but required human arbitration for nuanced cases (Chiang et al., 2024). Similarly, automated grading with LLMs in a bioinformatics course reduced TA workload and accelerated grading speed, but raised concerns about transparency, reproducibility, and student trust in AI-generated assessments (Poličar et al., 2025). To address this, frameworks like *BeGrading* (Yousef et al., 2025) have integrated LLMs into a multi-stage feedback pipeline, combining initial automated grading with targeted suggestions for improvement, while *CodEv* (Tseng et al., 2024) applied

chain-of-thought prompting, ensemble reasoning, and consistency checks to produce accurate and constructive feedback. Other work, such as the AI-augmented TA feasibility study (Ahmed et al., 2025), examined how LLMs can fit into human TA workflows in CS1 courses, focusing on providing timely, individualized support while preserving grading quality.

Beyond grading accuracy, the specificity and pedagogical usefulness of LLM-generated feedback vary considerably (Pankiewicz and Baker, 2023; Estévez-Ayres et al., 2024). Recent studies have examined how models can generate formative, actionable feedback that supports skill development in introductory programming courses (Mehta et al., 2023). One line of work uses program repair tasks as a proxy for feedback quality, showing that automated grading outputs can contribute to improvements in students' problem-solving and code comprehension skills (Pankiewicz and Baker, 2023). Others highlight the need for careful prompt engineering, rubric alignment, and iterative evaluation to ensure that feedback remains contextually relevant and educationally valuable. Therefore, human oversight remains a central design principle in this space, with LLMs serving as collaborators that augment TA capacity rather than replacements.

5 LLM Usage in Knowledge Modeling

Just as LLMs have been applied to formative feedback and assessment, they are increasingly being explored for a broader challenge in programming education: modeling what students know and how their understanding develops over time. Knowledge modeling supports instructional design by making student learning more visible, helping educators monitor progress, identify misconceptions, and create targeted interventions at scale. One common approach is knowledge component (KC) extraction, where student work is mapped to the concepts they need to master, such as variables, loops, and conditionals. While this process helps educators monitor progress and create targeted interventions, performing it manually is time-consuming and limits scalability.

Recent advances have demonstrated how LLMs can automate this extraction process with promising results. Researchers used GPT-4 to generate and tag KCs from multiple-choice questions, with human evaluators preferring the LLM-generated tags over instructor-assigned ones in about two-thirds of cases (Moore et al., 2024). *KCluster* (Wei et al., 2025) is another approach to combine LLM-generated question similarity metrics with clustering algorithms to automatically group related problems and discover their underlying KCs, producing models that outperform expert-designed baselines. Additionally, others (Niousha et al., 2025; O'Neill et al., 2025; Mittal et al., 2025) have presented early successful results on the use of LLMs toward KC extraction.

Researchers have also explored how LLMs can perform KC extraction during real-time learning interactions. LLMs can annotate student-tutor dialogues with KC tags during conversations, achieving close to human-level accuracy (Scarlatos et al., 2025). To gain more granular insights into student understanding, test case-informed knowledge tracing is another approach where individual test case pass/fail results serve as indicators for LLMs to better distinguish which concepts students have mastered versus those they struggle with (Duan et al., 2025). Additionally, incorporating student self-reflection prompts can significantly improve KC tagging performance by LLMs (Li et al., 2024).

While these advances highlight the potential of LLMs to scale knowledge modeling, their value depends on expert validation of concept mappings and alignment with course objectives. If not carefully validated, knowledge models that are inaccurate or poorly contextualized can misguide instructors and weaken their ability to design effective interventions. When integrated responsibly, however, LLM-generated models can strengthen learning analytics by giving educators actionable insights into student progress, revealing common misconceptions, and informing the design of targeted supports at scale. Future research can focus on improving the accuracy and reliability of these approaches across varied datasets and on developing methods that ensure valid and useful representations of student knowledge.

6 Discussion

Our review of recent LLM applications in programming education indicates that systems that successfully address students' pedagogical needs tend to retain human involvement throughout the workflow. Across formative code feedback, assessment, and knowledge modeling, successful applications frequently incorporate educators in the workflow to interpret results, refine automated outputs, or make

Topic	Best Practices
Formative Code Feedback	Use LLMs to generate hints, explanations, and error messages that scaffold in pedagogically-sound ways; balance automation with formative scaffolding to prevent over-complete fixes and overreliance.
Assessment	Ensure LLM grading aligns with rubrics and course standards; use human arbitration for nuanced cases.
Knowledge Modeling	When using LLMs for KC extraction and clustering, validate outputs against expert review of topics and subtopics.

Table 2: Best practices for LLM applications in programming education, synthesized across three focal areas.

instructional decisions. In contrast, fully automated systems often focus on narrower or more technical tasks where less contextual judgment is required. Best practices are summarized in Table 2.

Altogether, the systems surveyed demonstrate several shared strengths. They address scalability by automating tasks that would otherwise demand substantial instructor time, such as grading large cohorts or generating individualized debugging hints. Open-source language models have also been incorporated into APR pipelines, where evaluation frameworks use GPT-4-as-a-judge to approximate expert review at scale. This approach highlights the benefits of mixed human-and-automated evaluation in balancing accuracy, scalability, and cost in this domain (Koutcheme et al., 2024b, 2025). Research also incorporates mechanisms that improve consistency and transparency in instructional support, as seen in *BeGrading*'s variance analysis and criteria-aligned feedback generation (Yousef et al., 2025). Increasingly, these tools integrate pedagogical considerations into their design, from scaffolding strategies in debugging systems to feedback phrased in ways that guide student reflection and self-correction.

However, at the same time, performance varies considerably depending on factors such as prompt design, availability of course-specific training data, and evaluation practices. Many LLM-based systems are not explicitly tuned to instructional objectives, which can lead to technically correct but educationally unhelpful feedback (Sonkar et al., 2024). Performance often drops when moving from controlled benchmarks to authentic, noisy student code, and the opacity of model reasoning can reduce trust among both students and instructors. There is also the risk that students may overrely on incorrect model outputs, undermining opportunities for productive learning interactions (Pitts et al., 2025).

7 Limitations and Future Work

The current body of research on LLMs in programming education is still in its early stages, with limitations that can guide future research. While the focal areas covered in this survey reflect active areas of research, other domains, such as collaborative coding and accessibility support, remain underexplored. Addressing these gaps can take place in a larger-scale systematic literature review. Additionally, relating to the maturity of the current work reviewed, many of the systems surveyed are early-stage prototypes or evaluated only in small-scale settings, with limited evidence on scalability or long-term learning outcomes.

Technically, while early work has investigated fine-tuning and reinforcement learning with human feedback (RLHF) (Hicke et al., 2023), there remains significant scope for advancing model development and designing workflows explicitly aligned with pedagogical goals through course-specific fine-tuning. Research could also investigate adaptive collaboration frameworks where the degree of automation varies according to task complexity, user proficiency, and the model's own confidence in its output. Further priorities include identifying and mitigating biases in model outputs, especially in grading and feedback, and expanding the use of multi-modal, context-aware interaction that can adapt feedback to the learner's current state. While LLMs can offer an improved learning experience for programming education, their greatest potential lies in augmenting rather than replacing human expertise. Systems that remain adaptable, transparent, and closely aligned with pedagogical best practices are most likely to deliver meaningful and sustainable benefits for learners.

References

S Noor Ahmad and Juzlinda Ghazali. 2020. Programming teaching and learning: issues and challenges. *Fstm. Kuis. Edu. My*, 16(1):724–398.

Umair Z. Ahmed, Shubham Sahai, Ben Leong, and Amey Karkare. 2025. Feasibility study of augmenting teaching assistants with ai for cs1 programming feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 11–17, New York, NY, USA. Association for Computing Machinery.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Cheng-Han Chiang, Wei-Chih Chen, Chun-Yi Kuan, Chienchou Yang, and Hung-yi Lee. 2024. Large language model as an assignment evaluator: Insights, feedback, and challenges in a 1000+ student course. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2489–2513, Miami, Florida, USA. Association for Computational Linguistics.

Zhangqi Duan, Nigel Fernandez, Alexander Hicks, and Andrew Lan. 2025. Test case-informed knowledge tracing for open-ended coding tasks. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 238–248.

Iria Estévez-Ayres, Patricia Callejo, Miguel Ángel Hombrados-Herrera, Carlos Alario-Hoyos, and Carlos Delgado Kloos. 2024. Evaluation of llm tools for feedback generation in a course on concurrent programming. *International journal of artificial intelligence in education*, pages 1–17.

Aysa Xuemo Fan, Ranran Haoran Zhang, Luc Paquette, and Rui Zhang. 2023. Exploring the potential of large language models in generating code-tracing questions for introductory programming courses. *arXiv preprint arXiv:2310.15317*.

Zhikai Gao, Bradley Erickson, Yiqiao Xu, Collin Lynch, Sarah Heckman, and Tiffany Barnes. 2022. You asked, now what? modeling students' help-seeking and coding actions from request to resolution. *Journal of Educational Data Mining*, 14(3):109–131.

Zhikai Gao, Collin Lynch, and Sarah Heckman. 2023. Too long to wait and not much to do: Modeling student behaviors while waiting for help in online office hours. *Proceedings of the 7th Educational Data Mining in Computer Science Education (CSEDM) Workshop*.

Mohammad Hassany, Peter Brusilovsky, Jiaze Ke, Kamil Akhuseyinoglu, and Arun Balajiee Lekshmi Narayanan. 2024. Human-ai co-creation of worked examples for programming classes. *arXiv preprint arXiv:2402.16235*.

Yann Hicke, Anmol Agarwal, Qianou Ma, and Paul Denny. 2023. Ai-ta: Towards an intelligent question-answer teaching assistant using open-source llms. *arXiv preprint arXiv:2311.02775*.

Muntasir Hoq, Jessica Vandenberg, Shuyin Jiao, Seung Lee, Bradford Mott, Narges Norouzi, James Lester, and Bita Akram. 2025. Facilitating instructors-llm collaboration for problem design in introductory programming classrooms. (arXiv:2504.01259). ArXiv:2504.01259.

Billy Javier. 2021. Understanding their voices from within: difficulties and code comprehension of lifelong novice programmers. *International Journal of Arts, Sciences and Education*, 1(1):53–73.

Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of code language models on automated program repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1430–1442. IEEE.

Sungmin Kang, Bei Chen, Shin Yoo, and Jian-Guang Lou. 2025. Explainable automated debugging via large language model-driven scientific debugging. *Empirical Software Engineering*, 30(2):45.

Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, page 1–20. ACM.

Charles Koutcheme, Nicola Dainese, and Arto Hellas. 2024a. Using program repair as a proxy for language models' feedback ability in programming education. In *Workshop on Innovative Use of NLP for Building Educational Applications*, pages 165–181. Association for Computational Linguistics.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, Syed Ashraf, and Paul Denny. 2025. Evaluating language models for generating and judging programming feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSETS 2025, page 624–630, New York, NY, USA. Association for Computing Machinery.

Charles Koutcheme, Nicola Dainese, Sami Sarsa, Arto Hellas, Juho Leinonen, and Paul Denny. 2024b. Open source language models can provide feedback: Evaluating llms' ability to help students using gpt-4-as-a-judge. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 52–58.

Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. *Acm sigcse bulletin*, 37(3):14–18.

Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A Becker. 2023. Using large language models to enhance programming error messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 563–569.

Hang Li, Tianlong Xu, Jiliang Tang, and Qingsong Wen. 2024. Automate knowledge concept tagging on math questions with llms. *arXiv preprint arXiv:2403.17281*.

Richard Wing Cheung Lui, Haoran Bai, Aiden Wen Yi Zhang, and Elvin Tsun Him Chu. 2024. Gptutor: A generative ai-powered intelligent tutoring system to support interactive learning with knowledge-grounded question answering. In *2024 International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pages 702–707. IEEE.

Stephen MacNeil, Paul Denny, Andrew Tran, Juho Leinonen, Seth Bernstein, Arto Hellas, Sami Sarsa, and Joanne Kim. 2024. Decoding logic errors: a comparative study on bug detection by students and large language models. In *Proceedings of the 26th Australasian Computing Education Conference*, pages 11–18.

Yacine Majdoub and Eya Ben Charrada. 2024. Debugging with open-source large language models: An evaluation. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 510–516.

Julia M Markel and Philip J Guo. 2021. Inside the mind of a cs undergraduate ta: A firsthand account of undergraduate peer tutoring in computer labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 502–508.

Samiha Marwan, Thomas W Price, Min Chi, and Tiffany Barnes. 2020. Immediate data-driven positive feedback increases engagement on programming homework for novices. In *CSEDM@ EDM*.

Atharva Mehta, Nipun Gupta, Aarav Balachandran, Dhruv Kumar, Pankaj Jalote, and 1 others. 2023. Can chatgpt play the role of a teaching assistant in an introductory programming course? *arXiv preprint arXiv:2312.07343*.

Marcus Messer, Neil CC Brown, Michael Kölling, and Miaojing Shi. 2023. Machine learning-based automated grading and feedback tools for programming: A meta-analysis. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pages 491–497.

Kanav Mittal, Abigail O'Neill, Hanna Schlegel, Gireeja Ranade, and Narges Norouzi. 2025. Modeling student knowledge progression across concepts in intelligent tutoring interactions. In *International Conference on Artificial Intelligence in Education*, pages 105–118. Springer.

Steven Moore, Robin Schmucker, Tom Mitchell, and John Stamper. 2024. Automated generation and tagging of knowledge components from multiple-choice questions. In *Proceedings of the eleventh ACM conference on learning@ scale*, pages 122–133.

Rose Niousha, Abigail O'Neill, Ethan Chen, Vedansh Malhotra, Bita Akram, and Narges Norouzi. 2025. Llm-kci: Leveraging large language models to identify programming knowledge components. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, pages 1557–1558.

Priti Oli. 2024. *Towards automated scaffolding of learners' code comprehension process*. The University of Memphis.

Abby O'Neill, Samantha Smith, Aneesh Durai, John DeNero, JD Zamfirescu-Pereira, and Narges Norouzi. 2025. From code to concepts: Textbook-driven knowledge tracing with llms in cs1. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, pages 1565–1566.

Victor-Alexandru Padurean, Paul Denny, and Adish Singla. 2025. Bugspotter: Automated generation of code debugging exercises. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 896–902.

Maciej Pankiewicz and Ryan S. Baker. 2023. Large language models (gpt) for automating feedback on programming assignments. (arXiv:2307.00150). ArXiv:2307.00150.

Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. 2016. Revisiting why students drop cs1. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, pages 71–80.

Griffin Pitts, Neha Rani, Weedguet Mildort, and Eva-Marie Cook. 2025. Students' reliance on ai in higher education: Identifying contributing factors. *arXiv preprint arXiv:2506.13845*.

Allison Poh, Anurata Hridi, Jordan Barria-Pineda, Peter Brusilovsky, and Bita Akram. 2025. Example explorers and persistent finishers: Exploring student practice behaviors in a python practice system. Proceedings of 9th Educational Data Mining in Computer Science Education.

Pavlin G Poličar, Martin Špendl, Tomaž Curk, and Blaž Zupan. 2025. Automated assignment grading with large language models: insights from a bioinformatics course. *Bioinformatics*, 41(Supplement_1):i21–i29.

Anna Řechtáčková, Alexandra Maximova, and Griffin Pitts. 2025. Finding misleading identifiers in novice code using llms. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, pages 1595–1596.

Mohi Reza, Nathan M Laundry, Ilya Musabirov, Peter Dushniku, Zhi Yuan "Michael" Yu, Kashish Mittal, Tovi Grossman, Michael Liut, Anastasia Kuzminykh, and Joseph Jay Williams. 2024. Abscribe: Rapid exploration & organization of multiple writing variations in human-ai co-writing tasks using large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–18.

Kelly Rivers, Erik Harpstead, and Kenneth R Koedinger. 2016. Learning curve analysis for programming: Which concepts do students struggle with? In *ICER*, volume 16, pages 143–151. ACM.

Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM conference on international computing education research-volume 1*, pages 27–43.

Alexander Scarlatos, Ryan S. Baker, and Andrew Lan. 2025. Exploring knowledge tracing in tutor-student dialogues using llms. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, page 249–259, Dublin Ireland. ACM.

Andre Silva, Sen Fang, and Martin Monperrus. 2025. Repairllama: Efficient representations and fine-tuned adapters for program repair. *IEEE Transactions on Software Engineering*, (01):1–16.

Priscylla Silva and Evandro Costa. 2025. Assessing large language models for automated feedback generation in learning programming problem solving. *arXiv preprint arXiv:2503.14630*.

Shashank Sonkar, Kangqi Ni, Sapana Chaudhary, and Richard G Baraniuk. 2024. Pedagogical alignment of large language models. *arXiv preprint arXiv:2402.05000*.

En-Qi Tseng, Pei-Cing Huang, Chan Hsu, Peng-Yi Wu, Chan-Tung Ku, and Yihuang Kang. 2024. Codev: An automated grading framework leveraging large language models for consistent and constructive feedback. In *2024 IEEE International Conference on Big Data (BigData)*, pages 5442–5449. IEEE.

Alexandra Vassar, Jake Renzella, Emily Ross, and Andrew Taylor. 2024. Fine-tuning large language models for better programming error explanations. In *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*, pages 1–2.

Yumou Wei, Paulo Carvalho, and John Stamper. 2025. Kcluster: An llm-based clustering approach to knowledge component discovery. (arXiv:2505.06469). ArXiv:2505.06469.

Jacqueline Whalley, Andrew Petersen, and Paul Denny. 2020. Mathematics, computer science and career inclinations—a multi-institutional exploration. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–10.

Weiqing Yang, Hanbin Wang, Zhenghao Liu, Xinze Li, Yukun Yan, Shuo Wang, Yu Gu, Minghe Yu, Zhiyuan Liu, and Ge Yu. 2025. COAST: Enhancing the code debugging ability of LLMs through communicative agent based data synthesis. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2570–2585, Albuquerque, New Mexico. Association for Computational Linguistics.

Calvin Yeung, Jeff Yu, King Chau Cheung, Tat Wing Wong, Chun Man Chan, Kin Chi Wong, and Keisuke Fujii. 2025. A zero-shot llm framework for automatic assignment grading in higher education. (arXiv:2501.14305). ArXiv:2501.14305.

Mina Yousef, Kareem Mohamed, Walaa Medhat, Ensaf Hussein Mohamed, Ghada Khoriba, and Tamer Arafa. 2025. Begrading: large language models for enhanced feedback in programming education. *Neural Computing and Applications*, 37(2):1027–1040.

Jialu Zhang, José Pablo Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2024. Pydex: Repairing bugs in introductory python assignments using llms. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):1100–1124.

Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 851–870, Bangkok, Thailand. Association for Computational Linguistics.

Fida Zubair, Maryam Al-Hitmi, and Cagatay Catal. 2025. The use of large language models for program repair. *Computer Standards & Interfaces*, 93:103951.