Integrating Large Language Models into Programming Education: Opportunities, Challenges, and Learning Impacts

Fionn McGoldrick

Abstract—*This literature review looks at how Large Language Models (LLMs) are being used in programming education and what that means for students and teachers. Recent studies show that LLMs can explain code, give quick feedback, and help beginners understand difficult concepts in a more accessible way. They can reduce confusion, speed up problem-solving, and make learning feel more personalised. However, the research also points out real challenges. LLMs can produce incorrect answers, students may become too dependent on them, and there are concerns about plagiarism and reduced practice. Some studies report improved understanding when LLMs are used as support tools, while others show little benefit if students rely on them to generate full solutions. Overall, the review suggests that LLMs work best when used to guide and support learning rather than replace actual coding practice. The goal of this paper is to summarise what current research says, highlight the benefits and risks, and point out what still needs to be explored in the future.*

1 Introduction

Learning to code is one of the most important skills in today's world, but teaching it effectively to large groups of students has always been a major struggle. Teachers often can't give each student the energy and personalised help that they need, and getting instant, helpful feedback on code errors is tough. This has all changed since the sudden appearance of Large Language Models (LLMs). Tools like ChatGPT and GitHub Copilot can write, explain, and fix code with impressive skill. This new technology is shaking up programming classes, creating huge potential for improvement, but also causing confusion and concern.

This literature review is designed to take an in-depth look at the burgeoning intersection of Large Language Models and programming education. The rapid integration of these advanced AI tools into software development workflows necessitates a thorough examination of their role within the pedagogical landscape of computer science instruction. I aim to synthesise current empirical and theoretical literature to provide a balanced overview of this transformative technology's influence on how students learn to code and how educators teach it.

2 Background and Context

Before diving into the specific impact of Large Language Models (LLMs), it's crucial to understand the long-standing hurdles in programming education that these new tools promise to address. Historically, teaching introductory programming, often designated as CS1, has been notoriously challenging for both educators and students.

2.1 Traditional Obstacles in Education

Teaching programming requires students to master several complex, interconnected skills simultaneously, often leading to high rates of initial frustration and failure. Key traditional difficulties include:

- **Cognitive overload and difficult ideas:** Learning to program often requires students to change how they normally think. Instead of using everyday reasoning, they must follow strict, step-by-step computer logic. Ideas like variable scope, loops, conditionals, and data abstraction can feel confusing because beginners have to imagine a moving process by only looking at still text. Some topics, such as pointers and recursion, are known to be especially challenging for new learners. Rachel Cardell-Oliver presents a strong discussion of this issue in her paper How Students Experience Learning to Program. She notes that "in the literature ... particular aspects of pedagogy (the way computer programming is taught) lead to cognitive overload, and therefore that new, research-led approaches to instructional design will address the problem" [1].
- **Limited feedback and support:** In traditional classroom settings, students often receive feedback on their code long after submitting assignments, which slows down learning. Debugging is a skill that develops through timely and iterative practice, but when feedback is delayed or vague, students struggle to identify and correct misconceptions. Large class sizes make it difficult for instructors to provide detailed, individualised help, leaving many learners stuck on small syntax or logic errors that hinder conceptual understanding.

3 Methodology

This literature review uses a qualitative, document-based methodology focused on analysing peer-reviewed research published between 2021 and 2025. All sources were selected from recognised academic venues, including ACM

conferences, IEEE proceedings, MDPI journals, and arXiv preprints. The review concentrates on studies that examine the role of Large Language Models (LLMs) in computer science and programming education.

Source selection was guided by three criteria: (1) clear relevance to LLMs or AI-based tools in education, (2) a focus on programming, feedback, or adaptive learning, and (3) methodological soundness. This resulted in a set of representative works, including surveys on LLMs in education [11], empirical studies evaluating AI-generated feedback [5, 14], investigations into learning outcomes [6], and research on adaptive learning systems [4, 8]. Foundational papers on code generation and tutoring systems were also included to provide context, such as work on question generation [12], error-message improvement [9], and early evaluations of Codex in programming education [3].

The review applies a thematic synthesis approach. Each selected paper was read and coded for its main contribution. These contributions were then grouped into key themes commonly discussed in the literature: applications of LLMs (such as explanation, debugging, and feedback), benefits and opportunities for learners, risks and limitations, effects on student performance, and pedagogical implications for instructors.

This method allows the findings of different studies to be compared and combined to produce an integrated understanding of how LLMs are currently used in programming education. Papers raising ethical or pedagogical concerns [7, 2] were also included to ensure a balanced perspective. Historical work on traditional programming difficulties [1] was used to contextualise the challenges that LLMs aim to address.

Only peer-reviewed or formally archived research was included. Informal articles, editorials, and non-academic commentary were excluded to maintain academic quality. All citations follow BibLaTeX formatting for consistency throughout the review.

4 Applications of LLMs in Programming Education

LLMs are being used in several practical ways in programming education, from guiding learners to debugging and automating feedback. The most common applications include:

4.1 Code Generation

LLMs like Codex and GPT-4 are capable of generating code based on natural language prompts. This has been used in classrooms to help students prototype solutions and experiment with different problem-solving strategies [10, 3]. Instructors have also used these tools to generate template code or variations of problems, helping reduce content creation time. However, it's emphasized that students benefit most when they treat generated code as a starting point and actively analyse it [14].

4.2 Code Explanation

One of the most appreciated applications is using LLMs to explain what code does in plain language. Tools like ChatGPT have been praised for providing helpful breakdowns of logic, variables, and structures, especially for beginners [2, 9]. This lowers the cognitive barrier for those struggling to interpret syntax or understand how lines of code interact.

4.3 Debugging Support

Debugging is a core skill in programming, and LLMs have been applied to support this by analysing buggy code and suggesting fixes [9, 5]. This includes identifying likely mistakes, generating corrected versions, or walking students through possible causes of errors. Some systems are even integrated into IDEs to offer real-time debugging feedback as students code.

4.4 Feedback Generation

LLMs are also being tested and deployed in automated grading and feedback systems. For instance, the BeGrading framework uses LLMs to evaluate student submissions and generate personalised feedback on errors and code quality [14]. These applications can scale feedback delivery, making it immediate and detailed, especially in large classes.

These applications are transforming how instructors manage workload and how students interact with code. However, the effectiveness of each depends on the context of use and the level of guidance provided to learners [7, 2].

5 Benefits and Opportunities

The integration of Large Language Models (LLMs) into programming education has opened up a wide range of new possibilities for both learners and educators. These models are not just tools for code generation—they are reshaping how feedback, support, and instructional content can be delivered at scale and in real time.

5.1 On-Demand Support and Feedback

One of the most prominent benefits of LLMs is their ability to offer students immediate support. Students can ask questions in natural language and receive explanations, code suggestions, or hints in real-time, making them feel more supported outside of traditional classroom hours [7]. This round-the-clock access improves flexibility and encourages self-paced learning [2].

5.2 Personalised Learning Pathways

LLMs can adjust the complexity and content of their responses based on student input, allowing for more personalised education. Adaptive feedback systems powered by LLMs can tailor content, explanations, and challenges to each student's ability level [4, 8]. This supports differentiation in classrooms and allows students to progress at their own pace.

5.3 Enhanced Engagement and Confidence

Research shows that students feel more confident and engaged when they can interact with LLMs to clarify misunderstandings or test ideas [13]. This autonomy can increase motivation and reduce frustration, especially among beginners who may hesitate to ask for help from instructors [2].

5.4 Assistance in Curriculum Development

Instructors benefit from LLMs by using them to design exercises, generate example code, and assess the difficulty of programming tasks [11]. This saves time and enhances the variety and richness of instructional material without compromising on quality.

5.5 Improved Accessibility

LLMs can act as an equaliser for students with limited access to traditional tutoring or support services. By integrating LLMs into e-learning platforms, institutions can provide high-quality explanations and debugging support at scale, potentially reducing educational gaps [8, 7].

Together, these opportunities suggest that when used responsibly, LLMs have the potential to enhance both teaching and learning experiences in programming education across diverse contexts and learner populations.

6 Challenges and Limitations

While LLMs present many promising applications in programming education, they also introduce notable challenges that must be addressed to ensure effective and ethical use.

6.1 Over-reliance and Reduced Problem-Solving Skills

A recurring concern in the literature is that students may become overly dependent on LLMs, using them to bypass rather than develop problem-solving abilities. Jošt et al. found that increased LLM usage for tasks involving code generation and debugging correlated with lower final grades, especially on assessments requiring critical thinking [6]. This suggests that students relying heavily on AI may miss opportunities to engage deeply with programming concepts.

6.2 Accuracy and Hallucination

Despite their capabilities, LLMs are not always reliable. They can generate plausible-looking but incorrect code or explanations—a phenomenon often referred to as "hallucination" [7]. When learners are inexperienced, they may fail to detect such errors, leading to confusion and reinforcement of misconceptions [2].

6.3 Academic Integrity

The ability of LLMs to produce entire assignments raises ethical concerns around plagiarism and originality. Without clear guidelines, students may submit AI-generated solutions as their own work [11]. This has prompted educators to reconsider how assessments are structured and to implement new forms of AI usage policies and detection methods.

6.4 Bias and Ethical Concerns

LLMs trained on large-scale datasets may inadvertently reproduce societal biases or stereotypes in their output [7]. This introduces the risk of exposing students to inappropriate content or biased advice, especially if these systems are used widely in educational settings without content filtering.

6.5 Lack of Transparency and Explainability

LLMs often provide answers without a transparent reasoning process, making it difficult for both students and instructors to understand the basis of a given output. This lack of explainability may limit their usefulness in contexts where interpretability is essential [13].

6.6 Technical and Pedagogical Constraints

There are practical barriers to using LLMs effectively. These include access to computing infrastructure, integration with learning management systems, and the need for teacher training [4]. Pedagogically, it is still unclear how best to scaffold student interaction with LLMs to ensure learning gains rather than shortcuts.

Despite their promise, LLMs must be thoughtfully integrated into education systems to avoid undermining foundational skills and academic integrity. Many of these limitations can be addressed through appropriate policy design, instructional framing, and continuous evaluation of their educational impact.

7 Impact on Learning Outcomes

Understanding how LLMs affect student performance and long-term learning is critical to evaluating their place in programming education. Current research presents a mixed but insightful picture regarding their influence on outcomes such as grades, engagement, and skill development.

7.1 Performance Metrics

Several studies have examined whether students using LLMs perform better in assessments and coursework. A meta-analysis by Wang and Fan found that AI tools like ChatGPT had a significant positive impact on students' immediate performance, particularly in lower-order tasks such as syntax correction and basic algorithm implementation [13]. However, Jošt et al. reported a negative correlation between frequent LLM use and final grades in programming courses when students relied on LLMs to complete entire tasks rather than understand the material [6]. These findings suggest that the performance benefits of LLMs depend heavily on how they are used.

7.2 Engagement and Retention

LLMs have shown promise in increasing student engagement. Tools like ChatGPT can respond to questions in natural language, which helps lower the barrier to asking for help [2]. This has been linked to improved retention rates, especially in online or self-paced courses where students often struggle in isolation [4]. The adaptive and conversational nature of LLMs contributes to a more interactive learning environment, which can motivate students to stay engaged with programming tasks.

7.3 Skill Development

While LLMs can help students complete assignments, there are concerns about whether they foster deeper skill acquisition. Research by Jacobs and Jaschke shows that AI-generated feedback can help students identify specific weaknesses in their code and improve iteratively [5]. On the other hand, Raihan et al. caution that LLMs may inhibit the development of independent debugging and problem decomposition skills if students rely on AI-generated solutions without critical reflection [11]. Thus, LLMs appear most beneficial when used as scaffolding tools, not as substitutes for learning-by-doing.

Overall, LLMs can improve performance and engagement under guided conditions, but overuse or misuse can hinder the development of essential problem-solving and coding skills. The impact on learning outcomes is therefore not fixed, but highly dependent on instructional design and student behavior.

8 Adaptive Learning and Personalization

One of the most compelling advantages of LLMs in programming education is their potential to support adaptive learning environments that respond dynamically to student needs. Unlike static resources, LLMs can adjust explanations, challenges, and feedback based on a learner's current progress or query complexity.

8.1 Individualized Feedback and Scaffolding

LLMs can tailor their responses to each student's level of understanding. For example, a beginner might receive a basic explanation of a loop, while an advanced learner could be challenged with optimization techniques. This adaptability creates a more personalized learning experience and allows students to move at their own pace [4]. The BeGrading framework exemplifies this by generating individualized feedback based on common error patterns and student input [14].

8.2 Adaptive Assessment and Content Generation

Some systems now integrate LLMs to generate practice problems or quizzes tailored to the learner's strengths and weaknesses. These systems can increase task difficulty incrementally or repeat misunderstood concepts until mastery is achieved [8]. This type of adaptive curriculum has been especially effective in supporting students from disadvantaged backgrounds by filling knowledge gaps without requiring intensive instructor oversight.

8.3 Conversational Tutoring and Dynamic Interaction

LLMs facilitate natural language interaction, which makes them suitable for conversational tutoring. Students can engage in back-and-forth dialogue, asking follow-up questions or requesting simpler explanations. This interaction mimics the scaffolding typically offered by human tutors [12]. Chat-based tutoring can be particularly beneficial in asynchronous learning environments where real-time instructor support is not available.

8.4 Personalization at Scale

Perhaps the most transformative aspect is the ability to deliver personalized education at scale. Unlike one-on-one tutoring, which is resource-intensive, LLMs can provide custom support to hundreds or thousands of students simultaneously. This scalability enables institutions to offer individualized learning experiences even in large introductory programming courses [7].

In summary, LLMs have strong potential to support adaptive learning and personalization by offering tailored feedback, dynamic assessment, and conversational guidance. These capabilities can help close achievement gaps and empower learners to take control of their educational journey.

9 Pedagogical Implications

The introduction of LLMs into programming education requires educators to rethink traditional teaching strategies and classroom practices. While these tools offer significant instructional support, their integration raises important questions around curriculum design, assessment methods, and teacher roles.

9.1 Shifting Instructional Roles

As LLMs become more capable of providing immediate feedback and explanations, instructors may need to shift their focus from delivering content to facilitating critical thinking and metacognitive strategies. Teachers are no longer the sole source of information but become guides who help students evaluate and apply AI-generated content appropriately [7].

9.2 Designing AI-Resilient Assessments

To uphold academic integrity and ensure genuine skill development, assessments must evolve. Traditional programming tasks that LLMs can easily solve may not effectively measure student understanding. As a result, educators are increasingly incorporating process-oriented assessments, such as code walkthroughs, oral exams, or reflective writing, which emphasize reasoning over results [11, 6].

9.3 Scaffolding and Responsible Use

Students must be taught not only how to program, but also how to interact productively and ethically with LLMs. This includes understanding when and how to use AI tools, verifying outputs, and documenting their usage transparently. Institutions are beginning to introduce AI literacy components into the curriculum to address this need [13].

9.4 Teacher Training and Policy Development

Effective integration of LLMs also depends on equipping educators with the skills and resources to use these tools effectively. Many instructors remain unfamiliar with prompt engineering or unsure of how to support responsible use in the classroom. Professional development programs and clear institutional guidelines are essential to address these gaps [2, 7].

9.5 Equity and Access

While LLMs can help democratize access to quality support, they can also widen digital divides if access is uneven. Pedagogical planning should account for infrastructure disparities and ensure all students have fair opportunities to use these tools [8].

In short, the pedagogical implications of LLMs are broad and significant. To fully realize their educational potential, educators must proactively adapt teaching practices, develop ethical AI policies, and support learners in becoming thoughtful, critical users of AI technologies.

10 Future Research Directions

Although LLMs are becoming more common in programming education, many aspects of their impact remain underexplored. Most current studies focus on short-term academic performance, leaving a gap in understanding their long-term effects on student learning, retention, and problem-solving development [6, 5].

Further research should also explore how LLMs can be best integrated into classroom practices. Questions remain about the right balance between human instruction and AI assistance, how to scaffold LLM usage effectively, and how to teach students to engage critically with AI-generated content [7].

10.1 Equity and Ethics

One particularly important area for future research is the ethical and equitable use of LLMs. While these tools offer the potential to personalize learning and improve accessibility, they also risk reinforcing existing inequalities if not implemented carefully [8]. Studies should examine how LLMs perform across diverse student populations and how to ensure all learners benefit fairly. Additionally, further work is needed to address the risks of bias, misinformation, and inappropriate content in AI responses [13, 7].

In short, the next wave of research should prioritize sustainable, inclusive, and pedagogically sound uses of LLMs to maximize their educational impact.

11 Conclusion

Large Language Models are rapidly reshaping how programming is taught and learned. Their ability to generate code, explain concepts, offer instant feedback, and support personalized learning has introduced new opportunities in computing education. Studies suggest that when used responsibly, LLMs can increase engagement, improve access to support, and enhance learning outcomes—particularly for students who benefit from adaptive or self-paced instruction.

However, these advantages are accompanied by real challenges. Concerns around over-reliance, academic integrity, and the accuracy of AI-generated outputs must be taken seriously. Without clear guidance and thoughtful integration, LLMs risk becoming shortcuts rather than support systems, potentially undermining the development of core problem-solving skills.

Educators must rethink assessment practices, equip students with AI literacy, and implement policies that ensure fair and ethical use. Meanwhile, researchers must continue investigating how to maximize the benefits of LLMs while minimizing harm—particularly for underserved learners.

In summary, LLMs offer powerful tools that, if embedded wisely, can transform programming education. Their effectiveness ultimately depends on how they are used, who they are accessible to, and how we as educators, researchers, and students choose to engage with them.

References

[1] Rachel Cardell-Oliver. "How Students Experience Learning to Program". In: *Education Research and Perspectives* 41 (2014), pp. 196–216. ISSN: 0311-2543.

[2] Fitsum Gizachew Deriba, Ismaila Temitayo Sanusi, and Amos Oyelere. "Enhancing Computer Programming Education Using ChatGPT: A Mini Review". In: *Proceedings of the 23rd Koli Calling Conference* (2023). DOI: 10.1145/3631802.3631848. URL: https://doi.org/10.1145/3631802.3631848.

[3] James Finnie-Ansley et al. "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming". In: *Australasian Computing Education Conference (ACE)*. 2022, pp. 10–19. DOI: 10.1145/3511861.3511863. URL: https://doi.org/10.1145/3511861.3511863.

[4] Ilie Gligorea et al. "Adaptive Learning Using Artificial Intelligence in e-Learning: A Literature Review". In: *Education Sciences* 13.12 (2023). DOI: 10.3390/educsci13121216. URL: https://www.mdpi.com/2227-7102/13/12/1216.

[5] Sven Jacobs and Steffen Jaschke. "Evaluating the Application of Large Language Models to Generate Feedback in Programming Education". In: *2024 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2024, pp. 1–5. DOI: 10.1109/EDUCON60312.2024.10578838. URL: http://dx.doi.org/10.1109/EDUCON60312.2024.10578838.

[6] Gregor Jošt, Viktor Taneski, and Sašo Karakatič. "The Impact of Large Language Models on Programming Education and Student Learning Outcomes". In: *Applied Sciences* 14.10 (2024). DOI: 10.3390/app14104115. URL: https://www.mdpi.com/2076-3417/14/10/4115.

[7] Enkelejda Kasneci et al. "ChatGPT for Good? On Opportunities and Challenges of Large Language Models in Education". In: *Learning and Individual Differences* 103 (2023), p. 102274. DOI: 10.1016/j.lindif.2023.102274. URL: https://doi.org/10.1016/j.lindif.2023.102274.

[8] Jozsef Katona and Klara Ida Katonane Gyonyoru. "AI-based Adaptive Programming Education for Socially Disadvantaged Students: Bridging the Digital Divide". In: *TechTrends* 69.5 (2025), pp. 925–942. DOI: 10.1007/s11528-025-01088-8. URL: https://doi.org/10.1007/s11528-025-01088-8.

[9] Juho Leinonen et al. "Using Large Language Models to Enhance Programming Error Messages". In: *arXiv preprint* (2022). eprint: 2210.11630. URL: https://arxiv.org/abs/2210.11630.

[10] Tung Phung et al. “Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors”. In: *Proceedings of the 2023 ACM Conference on International Computing Education Research - ICER ’23*. 2023, pp. 41–42. DOI: 10.1145/3568812.3603476. URL: https://doi.org/10.1145/3568812.3603476.

[11] Nishat Raihan et al. “Large Language Models in Computer Science Education: A Systematic Literature Review”. In: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 938–944. DOI: 10.1145/3641554.3701863. URL: https://doi.org/10.1145/3641554.3701863.

[12] Megha Srivastava and Noah Goodman. “Question Generation for Adaptive Education”. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, 2021, pp. 692–701. DOI: 10.18653/v1/2021.acl-short.88. URL: https://aclanthology.org/2021.acl-short.88/.

[13] Shen Wang et al. *Large Language Models for Education: A Survey and Outlook*. 2024. arXiv: 2403.18105 [cs.CL]. URL: https://arxiv.org/abs/2403.18105.

[14] Mina Yousef et al. “BeGrading: Large Language Models for Enhanced Feedback in Programming Education”. In: *Neural Computing and Applications* 37 (2025), pp. 1027–1040. DOI: 10.1007/s00521-024-10449-y. URL: https://doi.org/10.1007/s00521-024-10449-y.