Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review

Doga Cambaz
dogacambaz@gmail.com
Delft University of Technology
Delft, The Netherlands

Xiaoling Zhang
X.Zhang-14@tudelft.nl
Delft University of Technology
Delft, The Netherlands

ABSTRACT

The recent emergence of LLM-based code generation models can potentially transform programming education. To pinpoint the current state of research on using LLM-based code generators to support the teaching and learning of programming, we conducted a systematic literature review of 21 papers published since 2018. The review focuses on (1) the teaching and learning practices in programming education that utilized LLM-based code generation models, (2) characteristics and (3) performance indicators of the models, and (4) aspects to consider when utilizing the models in programming education, including the risks and challenges. We found that the most commonly reported uses of LLM-based code generation models for teachers are generating assignments and evaluating student work, while for students, the models function as virtual tutors. We identified that the models exhibit accuracy limitations; generated content often contains minor errors that are manageable by instructors but pose risks for novice learners. Moreover, risks such as academic misconduct and over-reliance on the models are critical when considering integrating these models into education. Overall, LLM-based code generation models can be an assistive tool for both learners and instructors if the risks are mitigated.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Computing methodologies** → **Natural language generation**.

KEYWORDS

Systematic review, Artificial intelligence in education, Programming education, Code generation models, Large language models

ACM Reference Format:
Doga Cambaz and Xiaoling Zhang. 2024. Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630958

1 INTRODUCTION

Large Language Models (LLMs) such as OpenAI's Generative Pretrained Transformer (GPT) and the Codex, can function as AI-driven code generation models to generate code from natural language descriptions. These models enable natural language programming and can perform code-to-code operations such as code completion, translation, and repair, as well as language-to-code operations such as code explanation [11]. Given their availability and accessibility, AI-driven code generation models can potentially transform the learning and teaching of programming.

These AI-driven code generation models offer opportunities and challenges in programming education. One of the opportunities is using code generators to automatically correct syntax, allowing students to concentrate more on the problem-solving components of computational thinking [1]. Moreover, another study shows that AI-driven code generators allowed novice programmers to perform better and faster with less frustration while keeping their performance on manual code modification or writing code without the code generator [11]. Meanwhile, by producing programming exercises and explanations for the solutions, these tools could help educators develop curricula [1]. However, auto-generated code raises concerns about academic integrity and the risk of users' over-reliance on generated outputs [11].

AI-driven code generation models are growing as a part of the education landscape. However, the rapid emergence of these tools may catch educators off-guard, leaving them unprepared for the significant impact of code generation models on education [1]. Yet there is limited understanding of how best to adapt our teaching practices to manage the challenges and benefits associated with their use effectively. Hence, it is critical to review and adapt our educational practices to incorporate these new technologies.

This study aims to provide a comprehensive review of the current state of the use of AI-driven code generation models, specifically LLM-based code generation models, in teaching and learning programming. We formulated the following research questions (RQs) to guide our systematic literature review.

- RQ1 What are the teaching and learning practices involving LLM-based code generation models used in programming education?
- RQ2 What are the characteristics of the LLM-based code generation models used in teaching and learning programming?
- RQ3 Which indicators are used for evaluating the performance of LLM-based code generation models in teaching and learning programming?
- RQ4 Which aspects should be considered when utilizing LLM-based code generation models in programming education?

2 METHODS

We conducted a systematic literature review according to the guidelines proposed by Kitchenham and Charters [12]. Details are presented in the following subsections.

2.1 Search Process

Three databases were used for searching relevant papers: ACM Digital Library, Scopus, and Google Scholar. ACM Digital Library is selected as it covers a wide range of computer science topics. Scopus is included to access papers that are not presented in the ACM database as it is a multidisciplinary database with articles from various academic fields. Furthermore, since the topic of this review is rapidly evolving, we also choose to include Google Scholar, which enabled expanding the search to include unpublished work and identify additional resources to address publication bias.

The search query was created by pinpointing search terms from the research questions, listing their synonyms, and including specific LLMs or code generation tools like ChatGPT and Github Copilot. For Google Scholar, a more strict query is used to search in the full text of publications as it offers limited options to combine multiple search terms and does not allow limiting the search to title, abstract, and keywords. The search queries are listed as follows:

- **ACM Digital Library and Scopus**: ("code generation model*" OR Codex OR "Github Copilot" OR ChatGPT OR "AI coding assistants" OR "AI code-generators" OR "code generation") AND (teaching OR education OR learning OR educat* OR learn* OR instructor*) AND ("computer science" OR "computing education" OR programming OR "coding practices" OR "software engineering") AND (assessment* OR curriculum OR curricula OR practices OR proposal OR tools)
- **Google Scholar**: education learning teaching Codex OR "Github Copilot" OR ChatGPT OR "AI coding assistants" OR "code generation models" "computing education" OR "programming education" curricula OR practices OR proposal OR tools OR strategies "large language models"

The search on 08/05/2023 was limited to papers published in the last 5 years (2018-2023), considering the novelty of LLMs for code generation. Search results were recorded in a spreadsheet[1] to identify duplicates and record subsequent screening.

2.2 Criteria

The following inclusion and exclusion criteria were applied to filter articles to match the scope of our research.
Inclusion Criteria:

- Published or unpublished full papers in English
- Papers that present or discuss the use of LLMs for code generation for educational purposes
- Papers published in the last five years (2018-2023)
- Papers that focus on the impact of AI code generation on Computer Science and programming education

Exclusion Criteria:

- Papers irrelevant to LLM-based code generation models
- Papers irrelevant to use of code generation models in programming education
- Papers that are inaccessible

[1]bit.ly/code-gen-models-reviewresults

2.3 Selection Process

Figure 1 displays the steps and the results of the selection process, including *Identification*, *Screening*, and *Eligibility*. We identified 162 records from the ACM database (n=47) and Scopus (n=115) and 79 records from Google Scholar. Excluding duplicated results, 217 records remained for the screening process. For *Screening*, following the criteria in Section 2.2, we screened the title and abstract of articles and it yielded 36 records. Regarding *Eligibility*, we reviewed the methodology, discussion, and conclusion sections of the articles. This step ensured that the selected papers conform to the criteria in Section 2.2 and address at least one of the research questions. This step resulted in 21 articles for further analysis.

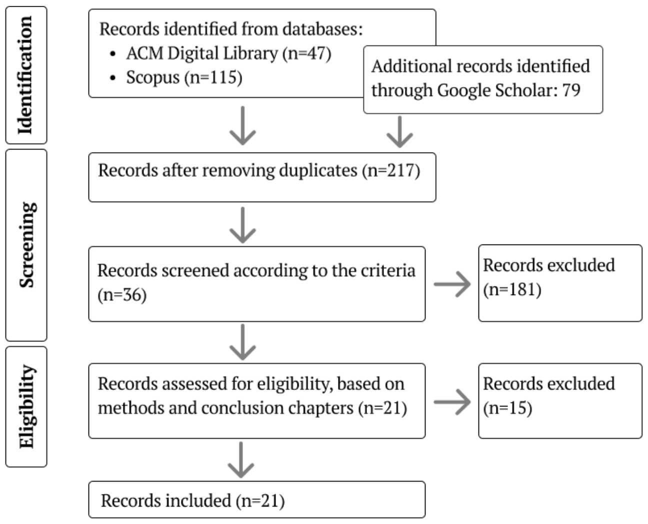
Identification
Records identified from databases:
• ACM Digital Library (n=47)
• Scopus (n=115)
Additional records identified through Google Scholar: 79
Screening
Records after removing duplicates (n=217)
Records screened according to the criteria (n=36)
Records excluded (n=181)
Eligibility
Records assessed for eligibility, based on methods and conclusion chapters (n=21)
Records excluded (n=15)
Records included (n=21)

Figure 1: PRISMA flow diagram

2.4 Coding and Information Extraction

Using the *Atlas.ti*[2] software, we extracted information by coding deductively and inductively. Initially, themes and codes were derived deductively from the research questions; subsequently, new codes were generated by iteratively reviewing the articles.

During the coding process, we matched quotes to relevant codes or themes. For RQ1, the theme was educational practices, including teaching, learning practices, and tools. Additional codes like assessment and content generation were added during the iterative process. For RQ2, the theme was characteristics of code generation models, with subcategories like performance, and limitations. For RQ3, the theme was performance indicators; we further categorized relevant quotes as quantitative and qualitative metrics. For RQ4, we included categories of risks, ethical use, and alignment with learning objectives, which were refined upon reading the articles.

3 RESULTS

All reviewed papers employed or evaluated OpenAI's GPT models for code generation. Six studies [2, 5, 7, 11, 20, 25] used OpenAI

[2]https://atlasti.com/

Activity	Activity Detail	References
Automatic generation of assignments	Exercise descriptions, sample answers and explanations	[10, 13, 14, 20, 24]
	Test cases for the exercises	[3, 10, 20]
	Personalized problems	[3, 14, 20]
	Variations of questions	[8, 20]
Assessment and evaluation	Grading Assignment	[10, 18, 19]
	Identifying areas students are struggling	[10]
	Feedback Generation	[10, 19, 20, 24]

Table 1: Teaching practices using Code Generation Models

Activity	References
Generate practice exercises	[8, 10, 13, 19, 20]
Generate exemplar solutions	[1, 2, 13, 19, 20]
Generate alternative solutions	[1, 8, 16, 20]
Improve student code	[19, 25]
Clarify error messages and provide suggestions	[1, 8, 14]
Support conceptual understanding	[1, 8, 14, 18, 19, 21]
Provide syntax tips	[8, 16]
Code explanations	[1, 3, 14, 15, 18, 21]

Table 2: Learning practices using Code Generation Models

Codex, while three studies [14, 16, 17] focused on analyzing Github Copilot powered by Codex. One study [15] used GPT-3, while [21] studied both GPT-3 and GPT-3.5. Six studies [6, 8, 10, 13, 18, 19] explored the use of ChatGPT on various educational tasks, while only one [6] used its latest version, GPT-4. Lastly, [1, 3, 4, 24] focused on the educational opportunities and challenges of AI code generation without specifying an LLM-based code generation model.

3.1 RQ1: Educational Practices That Use Code Generation Models

We identified three subcategories: teaching practices, learning practices, and educational tools that use LLM-based code generation models. Findings indicate that teachers can mainly use the models to generate assignments and evaluate student work, while for students, the models function as virtual tutors. Tables 1 and 2 list teaching and learning activities that use code generation models, with references to the corresponding papers.

3.1.1 Teaching Practices. LLM-based code generation models are mainly used in two kinds of teaching activities: automatic generation of assignments, and assessment and evaluation.

Regarding the automatic generation of assignments, we identified four subcategories. First, five papers [10, 13, 14, 20, 24] discuss the generation of exercise descriptions, sample answers, and explanations, while three papers [3, 10, 20] propose the generation of test cases for the exercises. Among these papers, while the others suggest or propose such teaching activities, only one paper [20] presents an empirical study that uses Codex to generate exercises and code explanations and evaluates the quality of the generated exercises. This paper [20] demonstrated that the majority of the automatically generated content is novel, sensible, including an appropriate sample solution, and in some cases, is ready to use. Moreover, according to three papers [3, 14, 20], models such as ChatGPT and Codex possess the capability to contextualize problem statements and generate personalized questions tailored to students' interests, resulting in more engaging questions. For instance, [3] emphasizes the potential of generating personalized Parsons problems based on students' incorrect solutions. Finally, according to [8, 20], these models also enable instructors to create new exercise variations from existing ones.

Furthermore, regarding using LLM-based code generation models for assessment and evaluation, we identified three subcategories. First, three papers [10, 18, 19] argue that the models can be used to grade assignments and quizzes, which can save teachers a significant amount of time. [10, 19] also underline the possibility of using the models to check for plagiarism in student work. Moreover, [10] mentions that ChatGPT can semi-automate grading by highlighting the potential strengths and weaknesses of the work in question. Lastly, four papers [10, 19, 20, 24] discuss ChatGPT's and Codex's capability of providing individualized feedback on programming assignments as well as writing assignments.

Finally, all of the papers referenced in Table 1 acknowledge the need for manual review of AI-generated materials to ensure accuracy and clarity due to the unreliability of LLMs. Nevertheless, researchers agree that using these models reduces teachers' workload compared to creating exercises from scratch. For example, Sarsa et al. [20] mentions that code generation tools help instructors overcome writer's block and generate ideas, even if the resulting exercises are not used directly. Additionally, Geng et al. [8] suggests that instructors and teaching assistants can manually review the generated material to ensure correctness and appropriateness.

3.1.2 Learning Practices. Overall, code generation models can help students study and practice programming by generating personalized learning materials and functioning as a tutor, especially for students who do not have access to tutoring. Table 2 shows eight activities that use code generation models in learning practices.

LLM-based code generation models can create additional learning resources in programming education, such as practice exercises [8, 10, 13, 19, 20] and sample answers [1, 2, 13, 19, 20]. According to Geng et al. [8], these models can generate personalized exercises tailored to the student's proficiency level, improving the learning experience. The models can also offer multiple alternative solutions to a given programming problem, introducing students to different problem-solving approaches [1, 8, 16, 20]. Furthermore, tools like Github Copilot and ChatGPT offer valuable assistance to learners by providing immediate feedback on their code. They achieve this by explaining the code [1, 3, 14, 15, 18, 21], suggesting optimizations [19, 25], providing syntax tips [8, 16], clarifying error messages and suggesting ways to fix the errors [1, 8, 14] . They also support conceptual understand by ,for instance, explaining understand algorithmic concepts [1, 8, 14, 18, 19, 21]. Moreover, these tools are capable of various types of code explanations. Given a code snippet, GPT-3 can analyze time complexity, identify common mistakes, summarize code, trace execution, fix bugs and explain how they were fixed, create real-world analogies, list relevant programming concepts, and predict console output [15].

Furthermore, [10, 19] noted ChatGPT's advantage of generating conversational dialogues, enabling learners to ask questions as if they ask their tutors. This makes the learning process more intuitive, interactive, and beginner-friendly. Additionally, [8] highlights that since students can input their own prompts, they can learn at their preferred pace and in alignment with their learning style.

3.1.3 Educational Tools. Three papers [5, 11, 25] extensively explain and evaluate educational tools that use LLM-based code generation models. These tools are *The Coding Steps* web app for learning basic Python programming [11], the *Robosourcing model* for scalable practice programming question generation [5], and the *MMAPR model* [25] for repairing bugs in student code. All three tools use OpenAI Codex. *The Coding Steps* [11] stands apart by offering a user interface tailored for novice programmers to interact with an AI code generator. To ensure the generated code is beginner-level and contextually relevant, the system customizes prompt messages for Codex by combining six predefined examples, existing code in the editor, and the user's requested behavior. Their research demonstrates improved task completion and correctness scores with reduced errors and completion times when using the AI code generator in *The Coding Steps*. Similarly, the *Robosourcing model* [5] tailors prompts by combining problem statements, sample solutions, and relevant themes to generate a pool of exercises. The model successfully produces coherent programming exercises with sample solutions and automated tests, despite minor accuracy concerns that were manageable through manual adjustments [5].

3.2 RQ2: Characteristics of the Code Generation Models

In total, ten papers [5–8, 17–21, 25] feature empirical studies examining the performance and characteristics of LLM-based code generation models for programming education. Among these, six papers [6–8, 17–19, 21] argue that while the models can generally produce well-structured and accurate solutions for programming assignments, they have several limitations. Furthermore, [5, 20] discuss the limitations of Codex in generating programming assignments, while [20] delves into the characteristics of generated code explanations.

Regarding the performance of LLM-based code generation models on question types, [21] found that GPT models perform better in handling questions involving code generation or explanation than multiple-choice questions (MCQs). MCQs with code snippets were particularly challenging for GPT models compared to those without. While fill-in-the-blank questions and completing natural language statements were handled relatively well, questions requiring analysis and reasoning about code, such as true/false questions or predicting output, posed the most difficulties. [6] supported these findings, demonstrating that even the latest GPT model, GPT-4, struggled with various question types, frequently selecting only some of the correct options in MCQs. GPT-4 also faced challenges with questions involving graph traversal and executing search algorithms only based on textual descriptions.

Regarding the performance of LLM-based code generation models on programming tasks, five papers [7, 8, 17–19] evaluated the performance of ChatGPT, Codex, and Github CoPilot on solving programming assignments, and all emphasized that adjustments were often needed for error-free compilation despite achieving decent code accuracy.

Regarding the generation of programming exercises, [5] found that approximately one-third of the exercises generated by Codex were immediately usable for teaching purposes and served as starting points for learners to evaluate and modify. Likewise, [20] also noted that Codex-generated programming exercises often required adjustments before using them in a course, as problem statements frequently did not address corner cases, and many exercises either lacked tests or had flawed ones.

Finally, regarding code explanations provided by LLM-based code generation models, [20] found that Codex covers around 90% of the code but contains inaccuracies in about 67.2% of the explanation lines. These errors are generally minor and can be quickly addressed by instructors or teaching assistants. The study highlights Codex's limitations in generating high-level code descriptions, as it tends to provide line-by-line explanations despite using various explicit priming statements.

3.3 RQ3: Indicators for Evaluating the Performance of Code Generation Models

Three papers [5, 11, 20] systematically analyze the performance of LLM-based code generation models with a list of performance indicators. Denny et al. [5] and Sarsa et al. [20] used a list of qualitative and quantitative metrics to evaluate the performance of the models for automatically generating exercises, sample solutions, and code explanations. The metrics and their definitions are provided in Table 3. On the other hand, Kazemitabaar et al. [11] focused on evaluating the impact of AI code generators on learner behavior rather than directly examining the performance of the models, which are categorized into three groups: (i) overall training metrics which include indicators like completion rate and the amount of received feedback, (ii) per-task performance which involves correctness score, completion time, and encountered errors, and (iii) AI code generator usage that includes metrics such as the percentage of code written by Codex, and the Jaccard text similarity between final submission and Codex-generated code.

3.4 RQ4: Aspects to Be Considered When Using Code Generation Models

We identified six aspects to be considered when using code generation models in teaching and learning programming, including academic integrity, over-reliance on the models, accuracy and reliability of the models, appropriateness for beginners, ethical concerns, and the models' role in programming education.

3.4.1 Academic Integrity. Eleven papers [1, 5, 7, 8, 10, 13, 16–19, 21] discuss the risk of AI code generators facilitating plagiarism and compromising academic integrity. Introducing LLM-based code generation models may increase plagiarism [5] and burden the detection of AI-generated answers [7, 16, 19]. Moreover, these tools may undermine the validity of academic assessments [8]. To mitigate these risks, [18] suggests revising academic integrity policies and honor codes to address the use of AI tools, providing clear and simple guidelines for the proper use of LLMs in education, and

Qualitative Metrics for Programming Exercises	Definition
Sensibleness	whether the problem statement describes a practical problem that could be given to students to solve [5, 20]
Novelty	whether the copy of the programming exercise or a similar programming exercise already exists and can be found online [5, 20]
Topicality	whether the generated problem incorporates the provided theme and concepts from the required sets (e.g. matches the CS concepts provided in the prompt) [5]
Readiness for Use	the amount of manual work a teacher would have to make for the exercises and the associated sample solution and tests [5, 20]
Quantitative Indicators for Programming Exercises	**Definition**
Executability of Sample Solutions	whether the sample solutions could be run [5, 20]
Automated Tests	whether the sample solution passed the automated tests [5, 20]
Statement Coverage	the statement coverage of the automated tests when the code runs [5, 20]
Metrics for Code Explanations	**Definition**
Presence and Frequency of Mistakes	the types of mistakes present and determining and how common they were in the explanations for the different priming programs [20]
Completeness of Code Explanations	whether all parts of the code were explained in the generated explanations (Yes/No) [20]
Accuracy of Explained Lines	the proportion of correctly explained lines out of all the generated explanation lines, indicating the accuracy of the code explanations [20]

Table 3: Metrics used for evaluating automatically generated programming exercises and code explanations. The qualitative metrics were first assessed by Yes / No / Maybe statements and then were quantitatively analyzed by counting Yes / No / Maybes.

training students on academic integrity to ensure they fully understand the importance of maintaining ethical standards. additionally, papers encourage research on new analysis techniques to distinguish AI-generated text [10, 19], and incentives to develop curricula and assessments that require the creative and complementary use of code generation models [10, 18].

3.4.2 Over-reliance. Eight papers [1, 2, 8, 10, 16, 18, 19, 21] highlight the risk of over-reliance on code generation tools. [19] and [10] accentuated that the use of these tools can be a barrier to improving learners' critical thinking and problem-solving skills. Over-reliance on the models can lead to the loss of creativity [18], and amplify laziness [19]. Likewise, [16] hypothesized that "over-reliance on tools like Copilot could possibly worsen a novice's metacognitive programming skills and behaviors." Moreover, [1] stated that novices using models like Github Copilot may become reliant on auto-suggested solutions, potentially resulting in careless reading of problem statements and missing critical thinking for problem-solving. [8] argues that this dependency on AI-generated code could gradually diminish the quality of education and devalue computer science degrees. [19] adds to that by appealing for additional research to develop academic curricula, question-and-answer formats, and exams to effectively tackle the challenges.

3.4.3 Accuracy and Reliability of the Models. Five papers [1, 6, 10, 13, 18] highlight the accuracy and reliability concerns of Codex and ChatGPT. ChatGPT has a tendency to generate solutions with non-existent rules or equations and provides unreliable, untraceable, and unverifiable answers [18]. It can also make errors in arithmetic and deduction but supports them with excellent explanations [6]. This makes it challenging for students to distinguish between errors and verified information, leading them to accept false or misleading information as true [10]. Likewise, according to [1], Codex can recommend syntactically incorrect code, including undefined variables, functions, and attributes, which may seem correct at first glance. To mitigate the risks, [10] stresses educating students on the critical evaluation of information and teaching strategies for exploration, investigation, and verification.

3.4.4 Appropriateness for Beginners. Papers [1, 7, 10] raise concerns about the appropriateness of these code generation models for beginners. According to [7], students using Codex to generate model solutions for exercises may hinder their learning if the generated solutions are incorrect or of poor style, resulting in the adoption of inappropriate conventions and poor coding style. While this is a risk with any crowd-sourced solution, the customized nature of Codex's solutions may lead students to perceive them as more credible. Furthermore, [1] states that the coding styles of publicly available code are potentially more advanced compared to those of novice programmers. Given that these models are trained on publicly available code, the style of the generated code may differ from those of typical novice programmers and their instructors.

3.4.5 Ethical Implications. Five papers [1, 5, 10, 16, 19] highlight the issue of harmful bias, claiming that code generation models are not immune to the bias in AI, and can possibly reflect stereotypes, represent only certain groups of people, etc. Furthermore, two papers [1, 16] mention licensing and attribution challenges. Publicly available codes that are used to train these models may have various licenses. However, AI-generated code often lacks clear attribution, leading to potential license violations. Thus, instructors should educate their students about how the models are trained and their responsibilities when reusing code.

3.4.6 The Position of Code Generation Models in Programming Education. Two papers [6, 7] propose contextualized, specific, and applied assessments that enable students to utilize code generation tools while still engaging in problem-solving. Likewise, [4] advocates shifting the focus from detecting and preventing the use of these tools to embracing and integrating them.

Furthermore, [1, 14] suggested a shift in the focus of programming courses from syntax and basic programming principles to higher-level algorithms, as code generation models can handle low-level implementation tasks. This could lead to a teaching approach that prioritizes algorithmic problem-solving, utilizing AI code generation for implementation and delaying syntax discussions until later stages [1]. Similarly, software engineering courses might pivot towards prompt engineering, code evaluation, and debugging [14].

Finally, Dobslaw and Bergh [6], Geng et al. [8] discuss the need for a transitional period for novice programmers. Understanding and assessing AI-generated code may be challenging for new programmers; thus, programming education is still crucial for individuals to effectively use code generation tools [8]. Instead of introducing students to these technologies from day one, it may be more beneficial for them to prioritize building a strong foundation in core computing concepts [6].

4 DISCUSSION

The findings suggested that LLM-based code generators can potentially improve teaching and learning as a teacher's assistant or a student's virtual tutor. However, their performance is unguaranteed, and the risks of using them should always be considered.

Regarding RQ1, we noted fewer papers on educational tools compared to those that propose learning and teaching practices, suggesting limited examples of active use of code generation models in education. Therefore, further research is crucial to address knowledge gaps and uncertainties among educators for effective integration of these models. Furthermore, results show overlapping use of LLM-based code generation models in teaching and learning activities, particularly in exercise creation. The models only act as assistive tools for teachers but are more valuable to learners since they can provide immediate feedback on student work. Studies have noted immediate feedback on code significantly improves student learning outcomes and engagement [9, 22, 23]. Yet, using these models for learning poses higher risks than for teaching, as novice programmers may overlook inaccuracies in the outputs. Therefore, we encourage further research to investigate ways to mitigate the risks for beginner programmers. The Coding Steps web app [11] exemplifies how an AI coding assistant powered by Codex can support complete beginners while incorporating control mechanisms to prevent over-utilization. Considering this, we believe that especially for novices, prompts can be tailored to limit the generation of huge chunks of code and constraints can be imposed to prevent direct use of generated code.

Answering RQ2 proved challenging due to the diversity of code generation models examined in the papers and their varying versions, making it difficult to generalize their characteristics. Additionally, the empirical studies differed significantly in their experimental designs, ranging from large-scale evaluations using hundreds of programming questions to smaller-scale evaluations involving twenty students. Nevertheless, common limitations emerged. While excelling in code and explanations, models face challenges with reasoning-based questions like MCQs. We believe it is crucial to clearly communicate the limitations of these models to students so that they do not consider the generated answers as the hard truth.

In response to RQ3, while evaluating the accuracy of the generated material is crucial, the qualitative metrics proposed by [8, 11] - sensibleness, novelty, topicality, and readiness for use - offer valuable insights for other researchers and educators to evaluate the AI-generated code and teaching material.

Regarding RQ4, the reviewed papers discussed the challenges of using models in education but lacked specific actionable guidelines for educators to ensure safe student interaction. Further research in developing new technologies, such as ways to obstruct over-utilization or AI-based plagiarism detectors to safeguard the integrity of education, is necessary. Furthermore, we suggest developing novice-friendly user interfaces and tools to promote the safe and appropriate use of code generation models in education, addressing academic integrity, over-reliance, and reliability concerns.

Finally, unequal access to education persists due to the dominance of English-based innovations. Only 4 out of 21 reviewed papers [10, 11, 20, 24] address the research and innovation gap in this area for non-English languages. While LLMs hold promise for enhancing programming education in English, they can lead to unfair access to educational technologies for non-English speakers. Unfortunately, the impact of code generation models on non-English programming education remains unexplored.

5 THREATS TO VALIDITY

The study faces several threats to its validity due to limitations in its design and execution. Firstly, the research area is new and still developing, underscoring the possibility of omitting recent significant developments. To mitigate this, we chose to include unpublished work [4, 6, 8, 16, 18, 21, 24, 25]. Consequently, the review might be reporting results that have not been filtered through a scientific review process. Moreover, the dynamic nature of LLMs means that results based on current and older versions may not hold for future versions, altering the educational practices and recommendations provided in the study. Another limitation stems from focusing solely on English literature; this decision may have excluded relevant studies in other languages, potentially introducing bias and limiting generalisability. Despite these limitations, the study aims to contribute to a better understanding of the use of LLM-based code generation models in programming education and stimulate further research.

6 CONCLUSION

This systematic review of 21 articles emphasized the use of LLM-based code generation models in programming education and its potential advantages as a teacher's assistant in the creation and evaluation of assessments and as a student's virtual tutor. However, generated content often contains minor errors that are manageable by instructors but pose risks for novice learners. LLM-based code generation models offer transformative teaching and learning possibilities, but addressing accuracy limitations and risks like misconduct and over-reliance is vital. Future studies should explore integrating AI code generators in classrooms and designing programming assessments that encourage critical thinking rather than relying on these tools as answer generators.

REFERENCES

[1] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 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

[2] Robert W. Brennan and Jonathan Lesage. 2023. Exploring the Implications of OpenAI Codex on Education for Industry 4.0. , 254-266 pages. https://doi.org/10.1007/978-3-031-24291-5_20

[3] Peter Brusilovsky, Barbara J. Ericson, Christian Servin, Frank Vahid, and Craig Zilles. 2023. The Future of Computing Education Materials. CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula, Curricula Practices.

[4] Christopher Bull and Ahmed Kharrufa. 2023. Generative AI Assistants in Software Development Education. arXiv:2303.13936 [cs.SE]

[5] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. 2022. Robosourcing Educational Resources – Leveraging Large Language Models for Learnersourcing. arXiv:2211.04715 [cs.HC]

[6] Felix Dobslaw and Peter Bergh. 2023. Experiences with Remote Examination Formats in Light of GPT-4. arXiv:2305.02198 [cs.CY]

[7] James Finnie-Ansley, Paul Denny, Brett A Becker, Andrew Luxton-Reilly, and James Prather. 2022. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. *Proceedings of the 24th Australasian Computing Education Conference*, 10–19. https://doi.org/10.1145/3511861.3511863

[8] Chuqin Geng, Yihan Zhang, Brigitte Pientka, and Xujie Si. 2023. Can ChatGPT Pass An Introductory Level Functional Language Programming Course? arXiv:2305.02230 [cs.CY]

[9] Vincent Gramoli, Michael Charleston, Bryn Jeffries, Irena Koprinska, Martin McGrane, Alex Radu, Anastasios Viglas, and Kalina Yacef. 2016. Mining Autograding Data in Computer Science Education. In *Proceedings of the Australasian Computer Science Week Multiconference* (Canberra, Australia) *(ACSW '16)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/2843043.2843070

[10] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, J. Kuhn, and G. Kasneci. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023). https://doi.org/10.1016/j.lindif.2023.102274

[11] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3544548.3580919

[12] B Kitchenham and S Charters. 2007. *Guidelines for performing systematic literature reviews in software engineering.* Technical Report EBSE-2007-01. Keele University and Durham University.

[13] Chung Kwan Lo. 2023. What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature. *Education Sciences* 13 (4 2023), 410. Issue 4. https://doi.org/10.3390/educsci13040410

[14] Stephen MacNeil, Andrew Tran, Juho Leinonen, Paul Denny, Joanne Kim, Arto Hellas, Seth Bernstein, and Sami Sarsa. 2022. Automatically Generating CS Learning Materials with Large Language Models. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, 1176–1176. https://doi.org/10.1145/3545947.3569630

[15] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating Diverse Code Explanations using the GPT-3 Large Language Model. *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, 37–39. https://doi.org/10.1145/3501709.3544280

[16] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. arXiv:2304.02491 [cs.HC]

[17] Ben Puryear and Gina Sprint. 2022. Github Copilot in the Classroom: Learning to Code with AI Assistance. *J. Comput. Sci. Coll.* 38 (11 2022), 37–47. Issue 1.

[18] Basit Qureshi. 2023. Exploring the Use of ChatGPT as a Tool for Learning and Assessment in Undergraduate Computer Science Curriculum: Opportunities and Challenges. arXiv:2304.11214 [cs.CY]

[19] Md. Mostafizer Rahman and Yutaka Watanobe. 2023. ChatGPT for Education and Research: Opportunities, Threats, and Strategies. *Applied Sciences* 13 (2023). Issue 9. https://doi.org/10.3390/app13095783

[20] 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* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. Association for Computing Machinery, New York, NY, USA, 27–43. https://doi.org/10.1145/3501385.3543957

[21] Jaromir Savelka, Arav Agarwal, Christopher Bogart, and Majd Sakr. 2023. Large Language Models (GPT) Struggle to Answer Multiple-Choice Questions about Code. arXiv:2303.08033 [cs.CL]

[22] Mark Sherman, Sarita Bassil, Derrell Lipman, Nat Tuck, and Fred Martin. 2013. Impact of Auto-Grading on an Introductory Computing Course. *J. Comput. Sci. Coll.* 28, 6 (jun 2013), 69–75.

[23] Chris Wilcox. 2015. The Role of Automation in Undergraduate Computer Science Education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 90–95. https://doi.org/10.1145/2676723.2677226

[24] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. 2023. Practical and Ethical Challenges of Large Language Models in Education: A Systematic Literature Review. arXiv:2303.13379 [cs.CL]

[25] Jialu Zhang, José Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2022. Repairing Bugs in Python Assignments Using Large Language Models. arXiv:2209.14876 [cs.SE]