IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 6 July 2024, accepted 8 August 2024, date of publication 14 August 2024, date of current version 23 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3443621

SURVEY

Navigating the Pitfalls: Analyzing the Behavior of LLMs as a Coding Assistant for Computer Science Students–A Systematic Review of the Literature

FARMAN ALI PIRZADO[1], **(Member, IEEE), AWAIS AHMED**[2], **(Member, IEEE), ROMÁN ALEJANDRO MENDOZA-URDIALES**[1], **(Member, IEEE), AND HUGO TERASHIMA-MARIN**[1], **(Senior Member, IEEE)**

[1]School of Engineering and Sciences Monterrey, Tecnológico de Monterrey, Nuevo León, Monterrey 64849, Mexico
[2]School of Computer Science and Engineering, University of Electronic Science and Technology of China—UESTC, Sichuan 611731, China

Corresponding author: Farman Ali Pirzado (a00836551@tec.mx)

This work was supported by the Tecnológico de Monterrey through the Ecosystem for Scaling Up Computational Thinking and Reasoning for Complexity Within the ''Challenge-Based Research Funding Program 2022'' under Project I003-IFE001-C2-T3-T E4C&CT.

ABSTRACT In recent years, large language models (LLMs) have been employed significantly in different domains of computing education. Nevertheless, these models have been focused on essential adherence to their integration as coding assistants in computing education. However, attention has been switched to thoroughly examining and analyzing LLM behavior, particularly in computing education for programming tasks such as code generation, code explanation, and programming error message explanation. Therefore, it becomes imperative to understand their behavior to examine potential pitfalls. This article addresses this gap systematically and details how different LLM-based coding chatbots, such as ChatGPT, Codex, Copilot, and others, react to various coding inputs within computing education. To achieve this objective, we collected and analyzed articles from 2021 to 2024, and 72 studies were thoroughly examined. These objectives include investigating the existing limitations and challenges associated with utilizing these systems for coding tasks, assessing their responses to prompts containing coding syntax, examining the impact of their output on student learning, and evaluating their performance as debugging tools. The findings of this review highlight that it is premature to incorporate these systems into computing education due to their limitations that may limit their effectiveness as comprehensive coding assistants for computer science students. These limitations include issues with handling prompts containing code snippets, potential negative impacts on student learning, limited debugging capabilities, and other ineffectiveness. The finding also reports multiple research directions that can be considered in future research related to LLMs in computing education.

INDEX TERMS Large language models, computing education, code generation, code explanation, programming error messages explanation.

I. INTRODUCTION

There is no denying the fact that the field of computer science continues to evolve, so it is time to use the tools and technologies that support learning and development within it. One such innovation that has recently gained significant attention is the emergence of Large Language Models (LLMs). It is undeniable that LLMs have become integral to computer science education due to their diverse capabilities, such as generating and explaining programming code [1]. The recent advancements have significantly enhanced machines' capacity to understand and produce content similar to humans [2]. Educators see the widespread use of these models as a significant enhancement to students and professionals in learning experiences across all levels of education, from primary to tertiary. However, these models play a role in improving reading and writing skills for various purposes, including generating practice quizzes,

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio Piccinno.

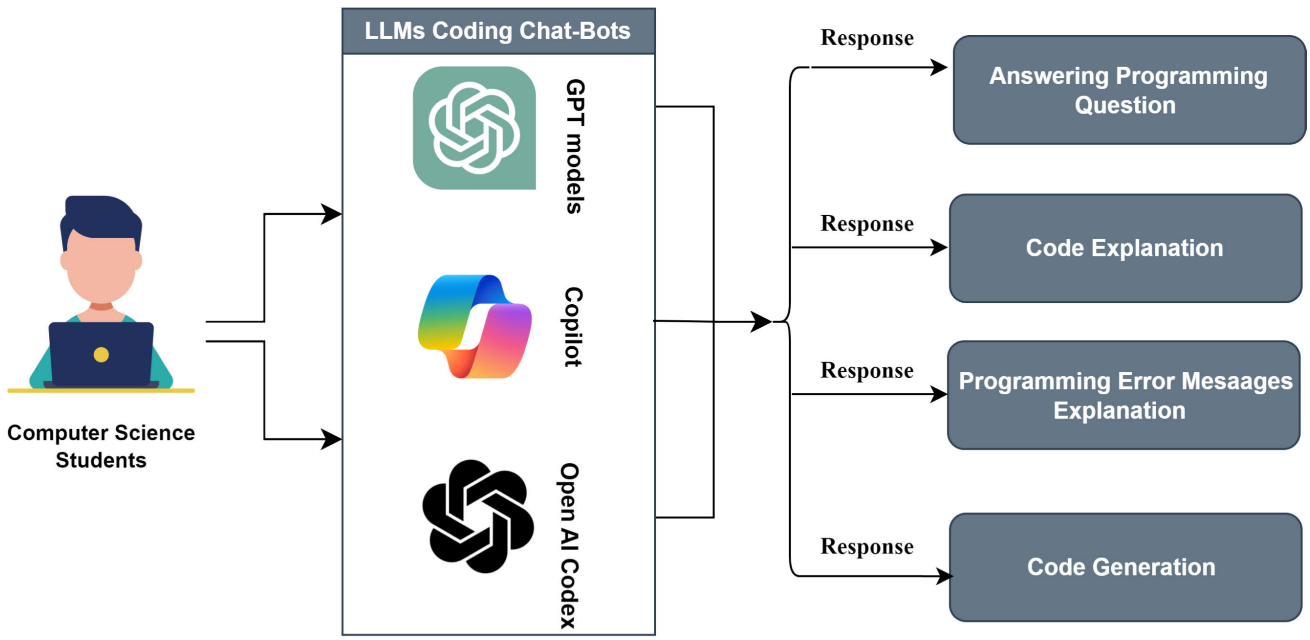

FIGURE 1. Large language models in computing education as a coding chatbots.

producing programming-related content, and conducting research tasks at an advanced level [3]. The interest in intelligent code generators and AI-powered coding tools has increased significantly in recent years. These are driven by their potential to transform the programming landscape. One of the examples that has captured widespread attention is "GitHub Copilot," a collaboration between GitHub and OpenAI [4], [5].

The several coding chatbots, including OpenAI Codex [6], Microsoft CodeBERT [7], Google Palm [8], and DeepMindAlphaCode [9] are trained on an extensive collection of source code to produce quality output from natural language description. This capability puts them at the center of AI coding assistants such as GitHub Copilot and Amazon's Code Whisperer, offering context-aware code suggestions to speed up coding tasks. These tools are widely desired for programming tasks such as code generation, code explanation, and generating programming-based solutions for assignments [10]. According to a recent study, LLM-generated content can be used as a supplement to students-generated content [11]. As shown in Figure 1, LLMs and related coding chatbots are a focal point in computing education. Several articles over the last five years have discussed different opportunities and challenges given by these coding chatbots, particularly in providing code generation, code explanation, and programming error messages (PEMs) explanation for both computer science educators and learners [1], [12], [13].

Further, output generated by LLMs on a given programming prompt and usability concerns indicate that there is still a great scope for enhancing and advancing these systems. For example, a study of 24 'programmers' discovered that [14], despite preferring Copilot over the intelligent plugin, errors remained in Copilot-generated code. Furthermore, programmers found larger code snippets challenging to understand, change, and troubleshoot. Similarly, novices using Copilot identified issues related to the software's design [15]. The studies have also been examining coding content generated by LLMs since 2020. It has been discussed that LLMs have been grappling with essential issues that could influence various domains of computer science education, such as producing generic outputs, lack of diversity, and exhibiting inconsistency in their outputs [16], [17]. This emerging role of these systems in computing education suggests that the literature needs to be reviewed to see the capabilities of these different coding chatbots based on LLMs for different programming capabilities, such as generating code in response to coding prompts from students, how well these chatbots can explain complex codes, and what their strengths and weaknesses are in debugging a given code. Therefore, this systematic literature review (SLR) focuses on the behavioral issues of LLMs while dealing with programming queries.

Through this investigation, we focused on current limitations and challenges that might affect the effectiveness of LLMSs in assisting students with programming tasks (RQ-1). We also examined whether LLMs exhibit similar responsiveness and adaptability to code-related inputs similar to general English inputs (RQ-2). This systematic review also investigates how LLMs impact computer science students' learning experience and outcomes, focusing on the quality, comprehensibility, and correctness of the outputs provided by these models (RQ-3). Finally, the eligibility and capability of LLMs as a debugging tool within computing education have been evaluated (RQ-4). Additionally, this study investigates the future gaps that still need to be addressed in the research related to the analysis of the current role of LLMs in programming tasks and explores the problems and challenges of adopting these tools as a programming assistant in computer science education. Further, this review provides findings that can help computer education researchers and stakeholders observe the use of these tools for programming activities.

A. MOTIVATION AND SCOPE

Based on current research, LLMs such as ChatGPT have the potential to revolutionize many fields, such as education, medicine, and science. Researchers have shown keen interest in investigating and discussing the role of LLMs and how they impact computing education and students [18]. Recent evaluations have dug into specific applications of LLMs, such as their utility in healthcare [19], banking [20], and code generation and explanation abilities in computing education [6]. Research also has addressed the potential opportunities and challenges provided by ChatGPT in education and its ability to enhance the programming learning experience [21]. During the workshops and studies, the researchers exhibit the capabilities of LLMs like ChatGPT to assist teachers and researchers in many academic activities, such as helping in generating quality programming assignments, providing good explanations of complex codes, and generating code from scratch [22]. Additionally, LLMs can be beneficial in creating educational content and improving the students' engagement while also describing there is the need for a particular set of skills among educators and learners to adopt these tools for their academic tasks regularly [3]. An article provides a detailed exploration of potential opportunities and threats introduced by LLMs, such as ChatGPt for education overall, highlighting their role in enhancing programming skills among students [21]. As a result of this discrepancy, there is an urgent need for an in-depth review of the state of the art that synthesizes existing LLMs coding chatbots and explores their implications, notably in programming tasks in computing education. Therefore, the work aimed to analyze the behavior of LLM chatbots from three perspectives: code generation, code explanation, and PEMs explanation. This enables computer education researchers and students to make informed judgments and apply these models to everyday tasks. Furthermore, identify future gaps to aid stakeholders in understanding the possible benefits and limitations of utilizing LLMs as a coding assistant in computing education. Therefore, this SLR aims to provide valuable insights and advance the field by identifying successful applications and assessing their impact. This will help researchers drive further development and exploration.

B. KEY CONTRIBUTION OF THE REVIEW

The main contributions of this review article are as follows:

- Providing a comprehensive overview of LLMs on three different programming tasks.
- Following programming tasks, a detailed analysis of different LLMs' behavioral issues on receiving students' programming prompts in the state of the art.
- Following behavioral issues, we also highlight different challenges and limitations of using LLMs for coding tasks, including code generation, code explanation, and programming error explanation tasks.
- This paper presents unique insights into the behavior and effectiveness of LLMs as coding assistants for computer science students. This review represents the first SLR focusing on this topic.
- In the last, the article also contributes by providing possible areas for future research in coding chatbots within computing education. These suggestions can guide readers in enhancing their research in this field and advancing it further.

TABLE 1. List of acronyms.

Acronyms	Description
SAR	Suitability Assessment Rubrics
SLR	Systematic Literature Review
LLM	Large Language Model
PEMs	Programming Error Messages
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analysis
RQs	Research Questions
IDEs	Integrated Development Environments

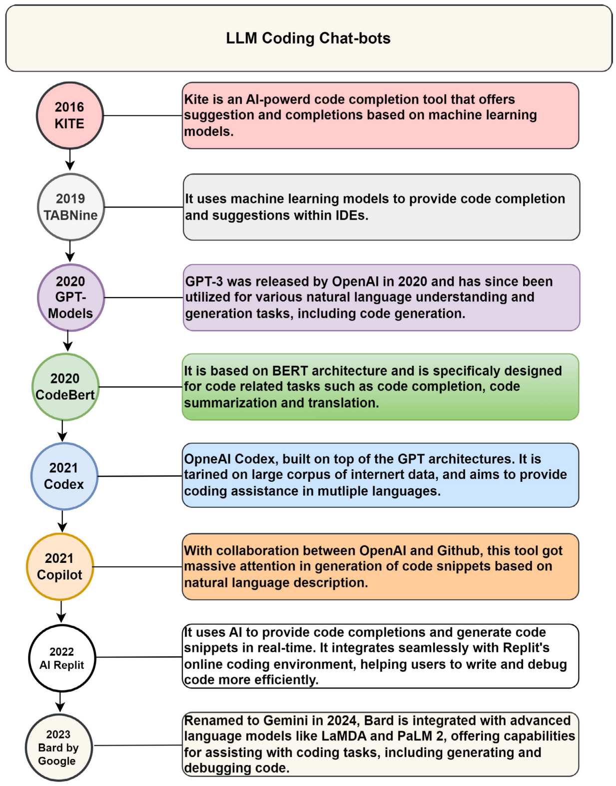

FIGURE 2. Evolution of LLM-based coding chatbots.

C. REVIEW ORGANIZATION

The rest of the study is organized as follows: Section II presents the history of coding chatbots and summarizes previous surveys on the same topic to contextualize the study. Section III describes the current study's methodology

TABLE 2. Existing review articles. [CG stands for code generation, while ce is code explanation, and PEMs is used for programming error messages].

Reference	Objectives	CG	CE	PEMs
[23]	The review focuses explicitly on utilizing LLMs in multiple software engineering applications. The team designed RQs on integrating LLMs in software engineering and analyzed the literature from 2017 to 2023. The review aims to bridge the gaps by providing valuable insights into using LLMs in software engineering	✗	✗	✗
[24]	This review article primarily aims to understand the capabilities of ChatGPT in education and research, focusing on how ChatGPT can enhance teaching and learning and finding out the potential issues related to the association of ChatGPT in academics to enhance learning and teaching. The review is followed by PRISMA, focusing on 50 articles discussing the influence of ChatGPT in education.	✗	✗	✗
[25]	This paper summarizes present research on ChatGPT and its possible applications in several fields of NLP, including code generation, text generation, and many more. They thoroughly analyzed 194 publications on arXiv, including trend analysis, word cloud representation, and distribution analysis across diverse applications.	✓	✗	✗
[26]	This research paper offers a detailed overview of LLMs, covering their history, architecture, training techniques, applications, and problems. The review also highlights obstacles to using LLMs in real-world contexts, such as ethical considerations, model biases, interpretability, and computational resource requirements.	✓	✗	✗
[27]	The article reviews educational research on LLMs and highlights practical and ethical problems that must be addressed for LLM-based innovations to be effective. They conducted a comprehensive scoping analysis of 118 articles published since 2017 to identify current findings on employing LLMs to automate and support educational tasks.	✗	✗	✗
[3]	This article addresses the potential benefits and drawbacks of using LLM models in education from both student's and teachers' perspectives. Multiple opportunities related to the use of ChatGPT for elementary, middle, and high school students have been discussed in this article.	✗	✗	✗
[28]	This paper examines the application of LLMs, and the authors also discuss the bias, scalability, and accessibility of AI models such as ChatGPT. The article also briefly discusses how ChatGPT can help students in programming tasks such as writing new code, understanding existing code, and debugging the code in just one section. It also highlights its role in education as a writing assistant.	✗	✗	✗

using the RISMA approach. Section IV discusses the study's key findings and their answers to the designed Research Questions (RQs). Section V discusses future research direction, while Section VI presents key aspects and implications of the current survey study, and Section VII addresses the study's limitations. Finally, Section VIII concludes the current research.

II. PRIOR WORK

This section initially discusses the history of coding chatbots and further synthesizes existing reviews to identify the research gaps and emphasize the need for the current study.

A. HISTORY OF CODING CHATBOTS

The history of coding assistant software traces back to the early days of computing when programmers sought tools to streamline and enhance their coding processes. In the 1960s and 1970s, the emergence of integrated development environments (IDEs) marked a significant milestone in coding assistance. These environments provided programmers with features such as code editors, compilers, and debuggers. These have laid the foundation for modern coding assistance tools. As programming languages evolved and became more complex, so did the need for more sophisticated coding aids.

As shown in Figure 2, the late 20th and early 21st centuries witnessed rapid advancements in coding assistant software, with the introduction of tools like auto-complete and syntax highlighting. The rapid increase of open-source communities also contributed to developing collaborative coding platforms and version control systems, further empowering programmers with code sharing, collaboration, and project management tools. In addition, recent years have seen the integration of artificial intelligence and machine learning technologies into coding assistance software, enabling features like predictive coding, automated refactoring, and intelligent code suggestion. These features have marked the dawn of a new era in coding assistance.

The launch of the Transformer model in 2017 continued the evolution of LLMs. The Transformer could understand long-term language relationships and allowed for parallel training on several Graphical Processing Units (GPUs), allowing it to train significantly larger models [29], [30]. Since 2018, the progress of LLMs has been rapid and evolutionary, with significant advancements in research and various applications. OpenAI introduces the GPT-2 language model, which has 1.5 billion parameters. They first withheld the complete model, reporting different concerns about misuse [31]; they eventually made it available to the public. Later, in 2020, OpenAI published GPT-3 API, which allows users to access and integrate GPT-3 into their applications, giving a variety of the latest and valuable applications [32]. Several advances in natural language processing and deep learning have marked the evolution of LLMs. These models have found applications in different areas, including computing education and programming, and their progress continues to influence the future of LLMs in programming contexts. This includes exciting opportunities and challenges that must be considered in research.

B. EXISTING SURVEYS

Moreover, during our SLR preparation, we observed several surveys addressing the utilization of LLMs in computer science education, exploring their potential and challenges. These surveys, detailed in Table 2, primarily examine LLMs' roles in education across various applications in code generation, code explanation, and PEMs explanation. Notably, there is a lack of recent SLRs pondering on analyzing the behavior of different LLM tools for specific programming tasks, as focused in this review. Thus, our review uniquely concentrates on the current state-of-the-art analysis of LLMs' behavior as programming aids for these three distinct tasks, representing a significant contribution to computing education.

III. SYSTEMATIC LITERATURE REVIEW—MATERIAL AND METHODS

This section aims to provide a clear and transparent summary of this review's techniques and search approaches for discovering and evaluating relevant research. The goal was to conduct a detailed study of LLMs' activities as coding chatbots in computing education, analyze and critically evaluate the research findings, and reveal limitations and challenges in current understanding concerning the research topic.

A. METHODOLOGY

This systematic review utilized the Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA), recognized as a minimal set of evidence-based items for reporting systematic reviews and meta-analysis [33], [34], having multiple advantages such as helpful to examine the extensive database of research articles, clear to address RQs and compelling to decide inclusion and exclusion criteria for related work. As shown in Figure 3, the designed methodology has three stages: input, pre-processing, and output. The work presented by Ahmed et al. [35] inspired the methodological steps and diagram. Each stage is tightly linked to the next to ensure a thorough and open examination. The sub-stages inside each of these stages give a detailed clarification in Figure 3, which shows a three-stage quality process of PRISMA for SLR inspired by [36].

First, we designed our RQs and conducted a preliminary scoping literature review. Our research topic was chosen in response to the gaps in the literature and emerging research areas that need further exploration. The RQs were created to analyze the behavior of LLMs on programming prompts. To achieve these objectives, reliable sources were considered, including IEEE, ACM, ScienceDirect, Web of Science (WoS), and SPRINGER. The search scope was limited to academic publications released in the last few years, particularly from 2021 to 2024, because LLMs are a relatively new and rapidly evolving technology. Research in the area of LLMs in education has significantly advanced recently, with many groundbreaking studies. Following the initial collection of articles, we screened the titles and abstracts to exclude the studies that were not relevant. Then, we full-text screen the reports to identify those that satisfy our inclusion criteria defined in Table 6. The study documented and disclosed the overall count of articles that underwent the review process, from eligibility assessment to the final evaluation. To ensure the overall quality of collected data, we undertook a data extraction approach using a standardized procedure form that gathered essential details from each included article, such as types of tools, designed study, source of data, and valuable findings. Finally, the included studies were subject to SAR to determine each study's suitability level with targeted RQs.

B. RESEARCH QUESTIONS

This section introduced RQs designed to scrutinize a particular objective of our study within a broader context. Through developing these RQs, we aim to pioneer a fresh perspective of AI-powered coding chatbots in computing education. We aspire to comprehensively analyze cutting-edge developing concerns related to the behavior of LLM-based coding chatbot applications from 2021 to 2024. The designed set of RQs offers a thorough analysis of the topic of the review, which covers underlying assumptions and possible implications of findings. We aim to contribute to the existing body of knowledge and provide valuable insights to education stakeholders by thoroughly analyzing each research question.

- **RQ-1 LLMs as a Coding Assistant**: What are the current limitations and challenges LLMs face that might hinder their effectiveness as good coding assistants for computer science students?
- **RQ-2 LLM Prompt Generalization**: Do LLMs react similarly to code prompts as English prompts?
- **RQ-3 Students Learning**: In terms of output quality, comprehensibility, and correctness, how do LLMs affect computer science students' learning experiences and outcomes?
- **RQ-4 LLMs as a Debugging Tool**: Are LLMs eligible and capable enough to be used as a debugging tool in computing education?

The findings of the above RQs help readers understand different issues related to the behavior of LLMs on programming inputs. Further, by pointing out challenges and constraints, the article helps readers understand the latest problems regarding the above applications of LLMs and how they can be overcome. This work also highlights the performance of different LLMs related to programming tasks. This contributes to readers' understanding of performance and accuracy issues in the LLMs so that researchers can work on those to overcome them in the future. This information is valuable for researchers currently working on AI research in computing education.

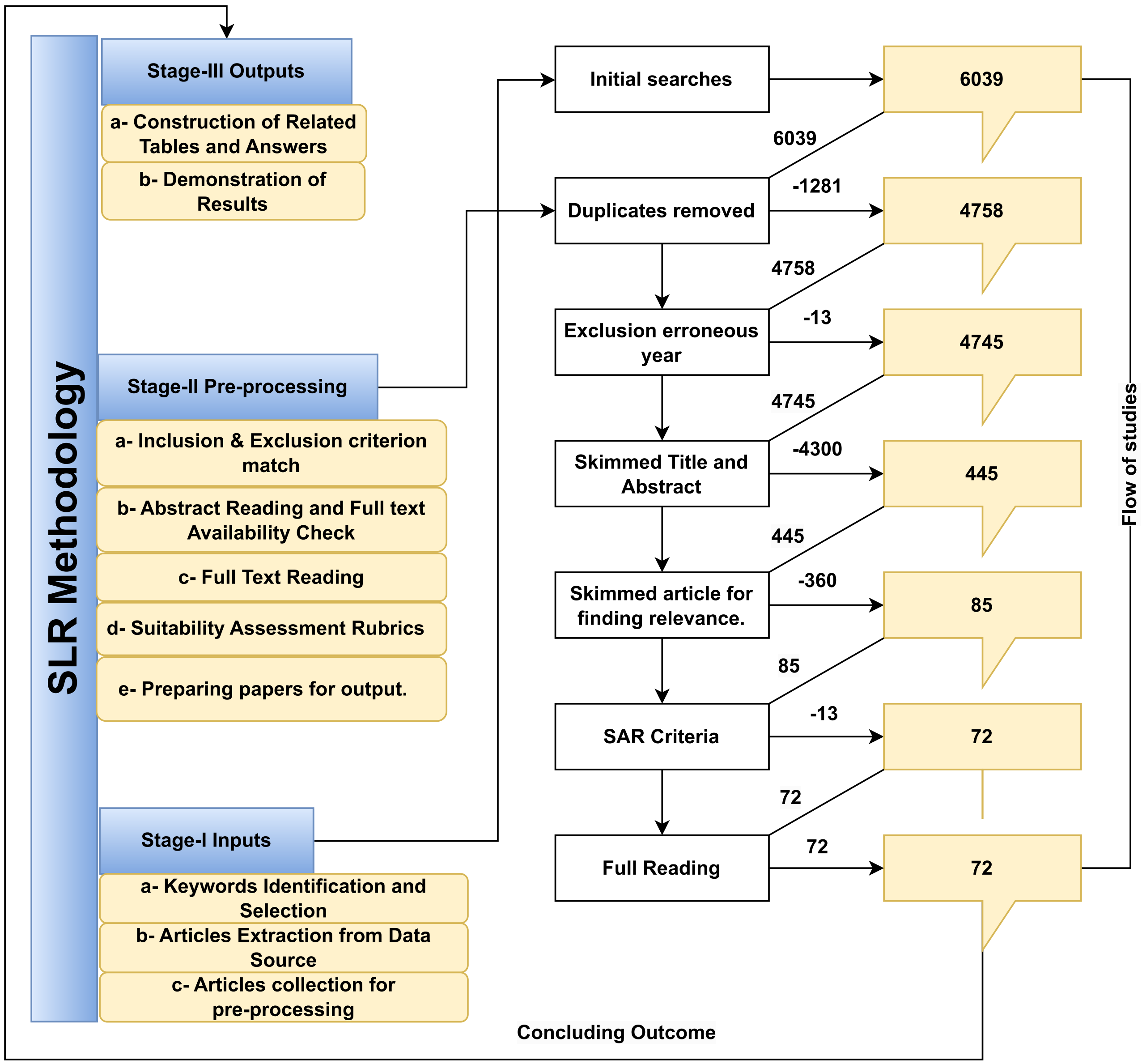

FIGURE 3. Methodology for articles selection.

C. SEARCHING KEYWORDS

To begin the investigation, we first look at the keywords associated with "Large Language Models," "Code Generation," "Computing Education," and "Copilot, Codex, ChatGPT." Furthermore, we extended our list of keywords. It includes the terms "code explanation," "programming errors explanation," and "Computer science students." These terms were developed with a view of the addressed topic and research objectives. Boolean search operators such as 'AND,' 'OR,' and 'NOT' were used to collect proper material; for reference, Table 4 lists searching keywords utilizing logic operators. The research question RQ-1 focuses on investigating the current limitations and challenges while dealing with programming queries received from computer science stakeholders. The search keyword contains words like "LLMs," "code generation prompts," "limitations," and "challenges." This search query helps identify articles that discuss different challenges and limitations LLMs face in the state of the art.

Additionally, the RQ-2 search query aims to gather information from many angles regarding prompts associated with programming activities to see if these systems perform well on prompts containing coding syntaxes, such as tasks involving code generation, code explanation, and PEMs explanations. Before moving to the last RQ, according to the primary focus of RQ-3 to find quality issues and explore students' learning experience, the data has been collected using the keywords "LLMs+code generation," "Quality," and "correctness." It has been used in search queries to find articles that have reported quality issues in the output of

LLMs corresponding to code generation, code explanation, and PEMs explanation tasks. In the end, RQ4 is about analyzing the effectiveness of these tools for debugging purposes when receiving PEM explanation prompts such as run time and compile time error explanations. This helps us understand whether these tools can be referred to as debugging tools for computer science students.

We used Publish Perish software to perform a search for data collection. This software allows us to retrieve and analyze data based on academic citation records. Table 3 presents a detailed overview of results obtained for keyword searching. The table records keyword-wise collective citations, year-wise citations, citations per author, and finally, the index of the articles. The table also highlights the emerging trend of the topic of interest, which necessitates the critical analysis for which this survey study is designed. Further, a word cloud of extracted keywords is prepared, as displayed in Figure 4, showing various key phrases from related publications as initially collected. Each word's size is based on its frequency and relevance in the corpus, visually depicting the dominating subjects and themes. This visualization shows the vast and diverse body of knowledge in the field of LLM education, illuminating the vital complex concepts being discussed in the reviewed literature. The key concepts and ideas in education can be better understood by analyzing the essential words displayed in the word cloud.

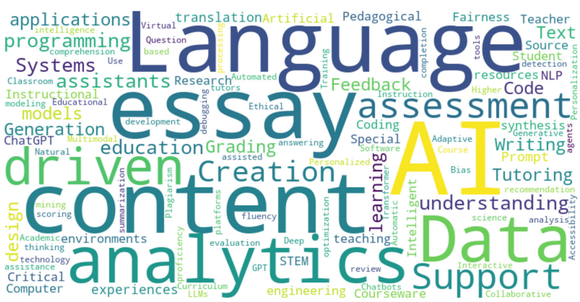

FIGURE 4. Exploring the landscape of educational insights: A word cloud of knowledge from topics discussed in the collected papers.

D. SNOWBALLING SEARCH

Our study used the snowballing search technique to find potentially relevant articles. This technique involves examining the reference lists or citations of the collected articles to increase the number of relevant articles. To improve snowballing, it is necessary to consider reference lists and citations and to analyze where similar works are referred to and cited systematically. There are two approaches to snowballing: backward and forward. Backward snowballing involves looking through references, while forward snowballing refers to exploring quotations. Before snowballing, a set of articles must be developed. After a quality assessment, this study's initial article list includes the remaining 65 papers. We followed forward and backward snowballing techniques, resulting in 303 and 400 articles, respectively. After removing duplicates, we were left with 150 articles only, which were considered for a detailed analysis to check the relevancy level. This way, we comprehensively analyzed these articles and selected an additional seven to include in our study.

E. INCLUSION AND EXCLUSION CRITERIA

When we first started our investigation, we used targeted keywords in a thorough search that produced hundreds of records. To guarantee the pertinence and quality of the research we examined, we executed a systematic review approach using PRISMA, shown in Figure 3. Therefore, we designed specific inclusion and exclusion criteria in Table 6, including selecting articles only written in English. They were published within a particular time and utterly accessible to the readers. The studies that did not pass the finalized inclusion criteria were removed from the further analysis. Following this process, we could conclude a specific subset of quality articles that fulfill the inclusion criteria and are entirely relevant to our research by using this rigorous SLR method.

F. DATA EXTRACTION AND ANALYSIS

A thorough assessment of chosen research articles, including in the bibliography, remains part of the current research. We followed a synthesis approach to extract and synthesize the related data using a methodological approach based on essential features presented in Table 7. These features include type of document, publishing source, research context, open database assessment, article ID, article title, author names, date of publishing, and count of citations. To efficiently organize and assess the research data, we collected it and synthesized it into an Excel sheet, which was later used in the analysis in Python.

In summarizing data, we conducted several analyses in the data synthesis section to gain insights about included studies, such as examining the distribution of publications per year, tallying the number of articles and citations per publisher, tracking counts over different publication years, and exploring additional relevant metrics. Our analysis examined the volume of articles and citations aggregated by publishers and years from 2021 to 2024, as shown in Figure 5 and Figure 6, respectively. This initial step provided a foundational understanding of publication trends over time. Subsequently, we delved into the distribution of publications among various publishers, along with aggregated citations, and included articles by year detailed in Table 5. This tabular analysis underscores the meticulous adherence of our research selection process to our methodology, ensuring comprehensive coverage across significant publishers. Furthermore, publisher-wise average citation count is presented individually in Figure 7. Additionally, publisher ranking and the relative contribution of included articles are shown in Figure 8 and Figure 9, respectively.In essence, this analysis aids in uncovering trends, patterns, and insights related to our research objectives.

TABLE 3. Publish perish search record at actual for the year range of 2021 to 2024.

Searching Keywords	Collective citations	Citation/year	Citation/paper	Citation/author	hIndex
Large Language Models for education	131217	32804.25	132.28	47766	175
Large Language Models as coding assistant	246665	61666.25	247.41	68269.20	224
Large Language Models for Computer Science Students	356316	89079	357.39	100157.21	283
Programming errors messages and large language models	45712	11428	228.56	11413.37	104
Programming error explanations	616270	154067.50	621.24	414947.44	273
Code generation and Large Language Models	104677	26169.25	523.39	22883.51	138
Code explanation and Large Language Models	144018	36004.50	720.09	33851.65	180
Copilot, Codex, ChatGPT, "computing education"	1067	533.50	6.39	64.6	21
Programming error messages explanation	24283	6070.75	121.42	11085.87	81

TABLE 4. Search queries for investigating Large Language Models in the context of code generation and computing education.

Query Category	Search Query
Large Language Models and Code Generation	“Large Language Models"AND “Code Generation"
	“LLMs"AND “Code Generation"AND “Computing Education"
	“Copilot"OR “Codex"OR “ChatGPT"AND “Code Generation"
Code Explanation	“Code Explanation"AND “Large Language Models"
	“Programming Errors Explanation"AND “LLMs"
	“Code Explanation"OR “Programming Errors Explanation"AND “Copilot"OR “Codex"OR “ChatGPT"
Computing Education	“Computing Education"AND “LLMs"
	“Computer Science Students"AND “Code Generation"
	“Computing Education"OR “Computer Science Students"AND “Large Language Models"
Challenges and Limitations	“LLMs"AND “Code Generation"AND “Limitations"
	“LLMs"AND “Code Generation"AND “Challenges"
	“Code Generation Prompts"AND “Limitations"OR “Challenges"
	“Programming Queries"AND “LLMs"AND “Challenges"
General RQs	“LLMs"AND “Programming Queries"AND “Computer Science Stakeholders"
	“LLMs"AND “Programming Queries"AND “Limitations"AND “Challenges"

TABLE 5. Count of included articles and citation sum by publisher and year.

Year	Publisher	Citation Sum	Article Count
2021	arXiv	595	3
2022	ACM	643	5
2022	arXiv	40	1
2023	ACM	1752	24
2023	Other	199	3
2023	Taylor & Francis	272	1
2023	arXiv	3800	23
2024	ACM	112	4
2024	arXiv	195	8

TABLE 6. Inclusion and exclusion criteria.

Criteria	Eligibility	Exclusion
Duplicate	Articles reports same content	Redundant Documents
Year	After 2020	Before 2020
Language	Published in English	Not in English
Type of document	Journal Articles, Reviews, and Conference Articles	Essay, book chapters, etc.

G. SUITABILITY ASSESSMENT RUBRICS

This section introduces four suitability assessment questions designed to evaluate the alignment of selected articles using SAR scores to decide for final inclusion.

TABLE 7. Key features used for data extraction.

Features	Description
ArticleID	Identification of article
Article Title	Title of the research article
Author	Author of the study
Date	Date published
Database	Source database
Publication source	What is the source of publication?
Topic focused	Is the topic related to the study?
Number of citations	Total count of citations of the study?

- (Q1) Is there explicit discussion within the article regarding LLMs and their involvement in computing education?
- (Q2) Does the article focus on LLMs coding chatbots?
- (Q3) Does the article outline potential hurdles associated with implementing LLMs like Codex, ChatGPT, Copilot, etc., in coding chat-bot applications within computing education?
- (Q4) Are the objectives and contributions in the articles valid and aligned with our goals?

The design methodology's interim step necessitates evaluating article quality based on the designed SAR, shown in Table 8 and Table 9. We calculated the score based on the following points:

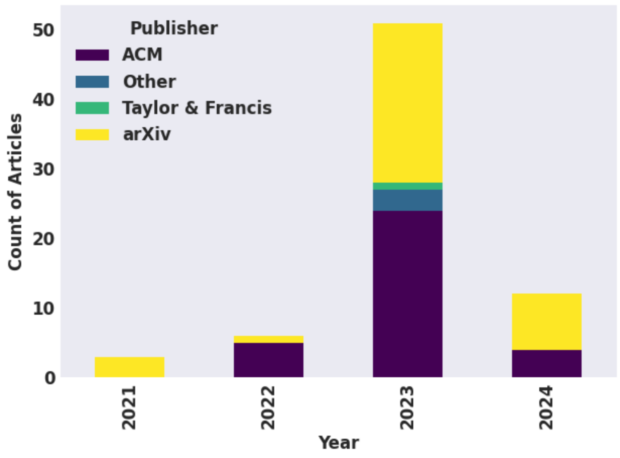

FIGURE 5. Count of articles by publisher and year.

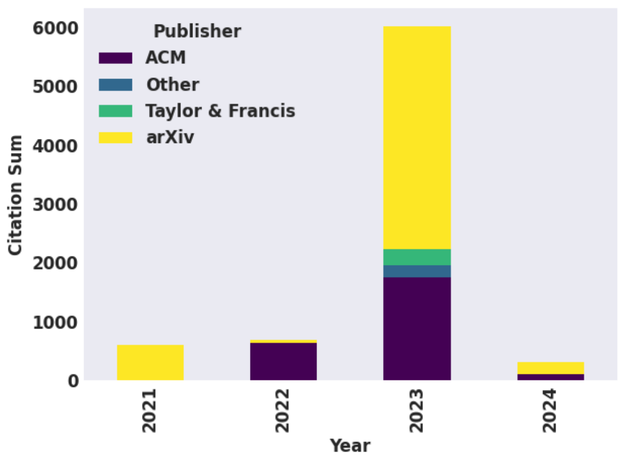

FIGURE 6. Citation sum by publisher and year.

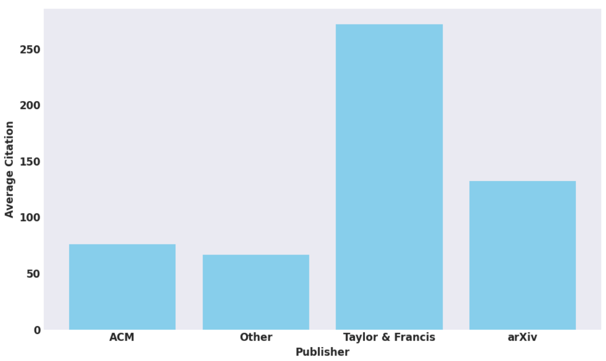

FIGURE 7. Publisher-wise average citation count.

- If the answer to a designed question is valid, Y is counted, and each Y is assigned a 2.5 score. Y maximum could be mapped to 10 points, as seen in the formula stated in Equation 1.
- If any selected article scored $>$ 5 to 10, we chose that article as the best fit, and the category was set to "High."
- To be classified as "Medium" level, the articles must have scored between 3 and 5, whereas "Low" articles must have a score of at least 2.5.

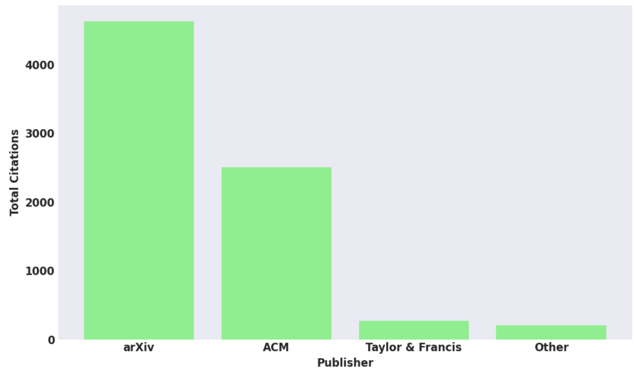

FIGURE 8. Publisher ranking by total citations.

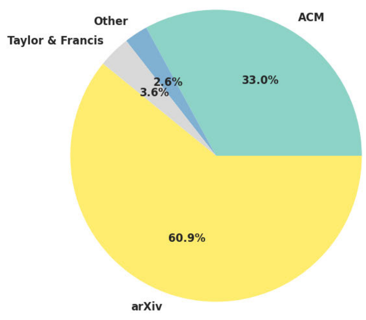

FIGURE 9. Relative contribution of publishers.

In this analysis, most of the articles fell in the "High" category because of the topics' recent advancement in the research. We processed our articles using the inclusion and exclusion criteria. Further details of the included articles are shown in the SAR Table 8 and Table 9. As per the PRISMA diagram 3, we initially input 85 articles to the SAR step. During this step, 13 articles were found irrelevant, with approximately zero scores. Furthermore, according to the designed suitability criteria of SAR, there were 72 in total: 11 medium, 4 low, and the remaining 57 fell into the high category. Such studies are the most relevant to answering our research question.

$$T\_Score\_Yes = \left(\sum_{i=1}^{4} Y_{Qi} \right) \times 2.5 \tag{1}$$

where Y_Qi is the possible sum of all "Yes", ranging from 1 to 4 counts depending on the individual article's SAR score.

$$TotalScore\_Other = 10 - \left(\sum_{i=1}^{4} N_{Qi} \right) \times 2.5 \tag{2}$$

where N_Qi represents the possible sum of "No" answers when recorded answers are a mix of "Yes" and "No," the SAR questions were a beneficial framework for examining the quality of articles. We processed studies from SAR before

TABLE 8. Score of suitability assessment rubrics (SAR)- [Total score (t. score) is equal to the addition of all Y, and all N used in 1 and 2 from calculation: High-Class ranges=6 to 10, Medium-Class=3 to 5 and Low-Class=2.5.] [Part 1 of 2].

References	Q1	Q2	Q3	Q4	T. Score	Category
[37]	Y	Y	Y	Y	10	High
[38]	Y	Y	Y	Y	10	High
[39]	Y	Y	Y	Y	10	High
[40]	Y	Y	Y	Y	10	High
[41]	Y	Y	N	N	5.0	Medium
[42]	Y	Y	N	Y	7.5	High
[43]	Y	Y	Y	Y	10	High
[44]	Y	Y	Y	Y	10	High
[45]	Y	Y	Y	Y	10	High
[46]	Y	Y	Y	Y	10	High
[28]	Y	Y	Y	Y	10	High
[47]	Y	N	N	N	2.5	Low
[48]	Y	Y	Y	Y	10	High
[49]	Y	N	N	Y	5.0	Medium
[50]	Y	Y	N	Y	7.5	High
[51]	Y	Y	Y	Y	10	High
[52]	Y	Y	Y	Y	10	High
[53]	Y	Y	Y	Y	10	High
[54]	Y	Y	Y	N	7.5	High
[11]	Y	Y	Y	Y	10	High
[55]	Y	Y	Y	N	7.5	High
[56]	N	Y	N	N	2.5	Low
[57]	Y	Y	Y	Y	10	High
[58]	Y	Y	Y	Y	10	High
[59]	N	Y	Y	Y	7.5	High
[60]	Y	Y	Y	Y	10	High
[61]	Y	Y	N	N	5.0	Medium
[62]	Y	Y	Y	Y	10	High
[63]	N	Y	Y	N	5.0	Medium
[64]	Y	Y	Y	Y	10	High
[27]	Y	N	N	N	2.5	Low
[65]	Y	Y	Y	Y	10	High
[66]	Y	Y	Y	Y	10	High
[67]	Y	N	N	N	2.5	Low
[68]	Y	N	N	Y	5.0	Medium
[69]	Y	Y	Y	Y	10	High
[70]	Y	Y	Y	Y	10	High
[71]	Y	Y	Y	Y	10	High
[72]	N	Y	Y	N	5.0	Medium
[73]	Y	Y	Y	Y	10	High
[74]	Y	Y	Y	Y	10	High
[75]	N	Y	Y	N	5.0	Medium
[76]	Y	Y	Y	Y	10	High
[77]	Y	Y	Y	Y	10	High
[78]	Y	Y	Y	Y	10	High
[26]	Y	Y	Y	N	7.5	High
[13]	Y	Y	Y	Y	10	High
[79]	Y	Y	Y	Y	10	High
[25]	Y	N	Y	N	5.0	Medium
[12]	Y	Y	Y	Y	10	High
[22]	Y	Y	Y	Y	10	High
[40]	Y	Y	Y	Y	10	High
[80]	N	Y	Y	N	5.0	Medium
[81]	Y	Y	Y	Y	10	High
[82]	N	Y	Y	Y	7.5	High
[10]	Y	Y	Y	Y	10	High
[83]	N	Y	Y	Y	7.5	High
[84]	Y	Y	Y	Y	10	High

going to the next step in methodology, complete reading, as indicated in Figure 3. Understanding that they are just one aspect of the SLR is essential. Additional factors that

TABLE 9. Score of suitability assessment rubrics (SAR)- [Total score (t. score) is equal to the addition of all Y, and all N used in 1 and 2 from calculation: High-Class ranges=6 to 10, Medium-Class=3 to 5 and Low-Class=2.5.] [Part 2 of 2].

References	Q1	Q2	Q3	Q4	T. Score	Category
[14]	N	Y	Y	Y	7.5	High
[85]	Y	Y	Y	Y	10	High
[86]	N	Y	Y	N	5.0	Medium
[87]	Y	Y	Y	Y	10	High
[88]	N	Y	Y	N	5.0	Medium
[89]	Y	Y	Y	Y	10	High
[90]	N	Y	Y	Y	7.5	High
[91]	Y	Y	Y	Y	10	High
[92]	N	Y	Y	Y	7.5	High
[93]	Y	Y	Y	Y	10	High
[83]	Y	Y	Y	Y	10	High
[87]	Y	Y	Y	Y	10	High
[91]	Y	Y	Y	Y	10	High
[94]	N	Y	Y	Y	7.5	High

can impact the quality of publications include, but are not restricted to, the study's design, the sample size, the statistical analysis employed, potential biases, and the overall relevance and contribution of the research to the field.

H. INCLUDED STUDY'S STATISTICAL ANALYSIS

The original inquiry yielded 6039 items, all obtained between 2021 and 2024. Based on the original output, 1281 articles were duplicated. Then, 13 articles were removed based on the exclusion of erroneous years. After scanning them by year and reviewing titles and abstracts, 4300 further articles were removed, decreasing the output to 445. After excluding 360 irrelevant articles, the study sample consisted of 85 articles. As stated in Figure 3, we ultimately passed 85 articles through SAR criteria and retrieved a final most relevant count of 72 papers in this study.

After answering individual RQs, we prepared a reference map table that succinctly helps users understand this survey's statistical analysis. Table 10 presents a reference map that reveals significant research activity and interest in exploring the role of LLMs as coding assistants for computer science students. Across four key RQs, 72 studies were identified, distributed predominantly from 2021 to 2024. The most studied research question, RQ-1, focusing on LLMs as coding assistants, garnered the highest number of studies, followed closely by RQ-4, examining LLMs as debugging tools. RQ-2 and RQ-3 investigate prompt generalization and students' learning experiences, respectively. These findings underscore the growing recognition of LLMs' potential in reshaping computer science education. They also highlight the need for further research to address the challenges and opportunities associated with their integration into learning environments.

IV. KEY FINDINGS

This section discusses the primary findings of this review. This study includes four RQs. The first focuses on using LLM Chatbots as coding helpers in computer education. The second tackles the issue of prompt generalization in LLMs. The third question focuses on determining the effect of using

TABLE 10. Reference map related to RQs.

Research Questions	Referenced Articles
RQ1	[1], [14], [22], [25], [27], [28], [39], [40], [44], [45], [47]–[49], [51]–[53], [56], [57], [60], [61], [63], [66], [68], [70], [71], [73], [75], [76], [78], [90]
RQ2	[6], [26], [38], [46], [58], [59], [62], [64], [81], [83], [87], [88], [95]
RQ3	[1], [3], [14], [22], [37], [40], [43], [44], [48], [50]–[53], [66], [67], [72], [78], [90]
RQ4	[13], [41], [42], [55], [69], [74], [76], [79], [82]–[86], [89], [91]–[94], [96]

LLMs on students' learning. In the final RQ, we look at the role of LLMs as debugging tools in computer education.

A. LLMs AS A CODING ASSISTANT - ANSWER TO RQ-1

The findings of this review related to RQ-1 suggest that it is too early to decide on LLMs as assistants to help in coding tasks in computer science education because LLMs can not be fully considered to identify all the issues available in a code. Concerning computing education stakeholders [1], [14], [22], [25], [27], [28], [39], [40], [44], [45], [47], [48], [49], [51], [52], [53], [56], [57], [60], [61], [63], [66], [68], [70], [71], [73], [75], [76], [78], [90]. Firstly, the ChatGPT's ability to perform debugging tasks and rectify errors is currently limited, particularly regarding PEM's explanation. Therefore, it necessitates ongoing research and refinement to enhance its efficiency and power for coding tasks [82]. It is the same with other coding chatbots on programming queries, such as Codex, which provides code explanations that cover the majority (90%) of the code; however, it still exhibits inaccuracies, with 67.2% of the explanation lines being correct. Though these errors in the output of LLMs on programming inputs are frequently minor, straightforward, and easily fixable by instructors or teaching assistants, they raise a concern for the reliability of these tools for understanding code and integration with traditional education systems [10].

Furthermore, research comparing the quality of code generated by Copilot with human-written code indicates that while Copilot increases productivity in terms of the number of lines of code, the quality of its output is generally inferior, with higher failure rates on medium and complex tasks [14], [75]. While the latest GPT models can produce high-quality output on code generation and explanation inputs, they still struggle with input containing large and complex coding snippets. Also, these models face issues with generalization, often underperforming on new and unseen problems related to programming [81], [96]. Similarly, research on Copilot's performance on a public dataset of 166 programming activities reported a 47.6% success rate on the first attempt, improving to 60% with natural language changes to the problem description [76]. Nonetheless, ChatGPT showed a 71.81% overall success rate on a Leetcode dataset, successfully solving 84 out of 128 problems on the first attempt but failing to generate correct answers for 36 problems even with feedback somehow [82]. Codex is reported as good at performing moderately complex code reading and writing tasks. Still, it struggles with more complex requirements in the programming queries, likely due to the absence of such scenarios in its training data [78]. These findings underscore the current limitations and challenges LLMs face, highlighting areas for improvement to make them more effective coding assistants for computer science students. Furthermore, Table 11 presents critical findings related to RQ-1, highlighting supporting statements.

In conclusion, despite LLMs' promising capabilities as coding assistants, several constraints and obstacles were recorded in this review that may restrict their effectiveness as full coding assistants for computer science students. The challenges addressed in this review are crucial for maximizing the utility of LLMs in supporting students' learning and development in computer science education. At the moment, LLMs are capable of helping with code generation, explanation, and debugging, but they have various limitations when integrating into the traditional education system. For example, LLMs can sometimes generate code that looks correct but may contain bugs or logical errors and may struggle to understand complex programming queries with large code snippets as input.

Apart from that, LLMs rely on the clarity of user input; ambiguous or poorly phrased programming questions can lead students to prevalent or incorrect solutions. Students must learn how to prompt with LLMs while effectively asking programming questions. LLMs' inability to perform well on unseen queries, including coding snippets, highlights the necessity for constant improvement and evolution. However, there is still room for improvement in the accuracy and quality of the code generated by LLMs, particularly when compared to human-generated code. Because of the complexity of PEM explanation and debugging tasks, it is considered more challenging to incorporate these LLMs into computing education. Considering these obstacles, LLMs could improve programming learning with further development in a positive manner. To overcome these challenges and utilize LLMs in programming tasks in computing education, researchers, educators, and other stakeholders must work together to address these difficulties and use LLMs to strengthen computer science education. It is possible to integrate these tools into computing education by overcoming the reported weaknesses of LLMs to create more effective and efficient coding assistants that can improve programming education and practice in the future.

B. PROMPT GENERALIZATION- ANSWER TO RQ-2

The review's findings, which align with RQ-2, show that LLMs can typically handle both code and English prompts well but somehow weaken queries contained in non-English data, especially coding snippets. Also, their performance varies depending on the complexity and specificity of the task. For English prompts, LLMs excel at producing cohesive and contextually relevant responses. They may hold

TABLE 11. Key findings related to RQ-1.

Reference	Supporting Statement
[96]	ChatGPT suffers from generalization problems because it can not perform well on new and unseen issues or new problems.
[82]	Concerning accuracy, ChatGPT faces challenges in generating correct answers. Roughly 30% of them produced inaccurate solutions that remained unsolved even with feedback provided by leetcode.
[82]	ChatGPTs capability to debug a given code and learn from errors presented in PEMs prompts appears to be weaker. There seems to be room for improvement. Research needs continuous improvement and evolution to enhance LLMs' coding chatbot abilities to have an effective and powerful tool for programming tasks in computing education.
[10]	Another study reported that the explanations generated by Codex cover a significant portion (90%) of the code despite some inaccuracies (67.2% of explanation lines were correct). They also observed that in most cases, the erroneous lines contained only minor mistakes that an instructor or teaching assistant could quickly fix.
[14], [75]	On the contrary, some studies explore the quality of the code generated by Copilot and compare the programming code written by humans to see the differences between both. The team infers that Copilot's code quality is weaker than human-generated code.
[81]	ChatGPT requires more efficiency in utilizing information to generate accurate outputs.
[48]	Although ChatGPT appears to be excellent in various tasks, it is not adept at providing textbook solutions. Around half of the correct solutions generated by ChatGPT contain complex and useless information.
[28]	ChatGPT can occasionally report incorrect codes as correct ones. Furthermore, it may not anticipate edge cases that could cause the code's functionality to fail in some circumstances, and it may not always generate the best coding technique.
[57]	Codex was worse on Parsons's problems; in around 50% of cases, Codex could not resolve the issues correctly.
[57], [64]	Codex was worse on Parsons's problems; in around 50% of cases, Codex could not resolve the issues correctly. However, GPT-4 seems good at solving Persona problems, completing 96.7% of the tasks.
[50], [59], [83], [88]	MCQ-type questions containing coding snippets make the prompt more challenging for GPT models to understand. GPT-3 only correctly solved 199 problems out of 530 questions, and GPT-3.5 was observed to be more successful since this model solved 341 questions correctly. However, GPT-4 remained most capable in solving MCQ queries by solving 446 questions correctly, which reflects that GPT models improved noticeably over generations and GPT-4 is good at grading feedback and task synthesis.
[38]	Despite being discussed as more successful than GPT 3.5, GPT-4 still struggles with fine-grained formatting requirements for both the output and code.
[74]	The previous versions of LLMs like GPT-3 should not be directly handed over to students because code explanations of fewer quality outputs in the correctness of code descriptions.

conversations, answer inquiries, explain things, and even create original material on various topics and genres [6], [26], [38], [46], [58], [59], [62], [64], [83], [87], [88], [95]. They may, however, struggle with highly specialized or domain-specific tasks because their training data does not thoroughly cover every programming language.

A recent study examined the ability of LLMs, particularly OpenAI Codex and GPT-3.5, to identify and respond to students' help requests in programming queries. Data collection has been performed via an online programming course at Aalto University, Finland, and included help requests and code samples. The study results show that LLMs struggle with queries containing data apart from English letters, such as input with code snippets. This highlights the need for a customized version of LLMs explicitly designed for computer science students' coding queries to use in computing education [58]. Further examination of different openAI GPT models revealed that, although these models can generate code from natural language descriptions and provide line-by-line explanations of code execution, they are more effective with queries in everyday language than those involving code snippets. This was particularly evident in a study using MCQ assessment exercises from three Python courses, which showed that LLMs perform better on English language inputs than on code or symbol-containing prompts [95].

While LLMs are exceptionally good at handling English input, another similar finding is that they struggle with coding prompts and special symbols, which leads to inaccurate outputs. In particular, GPT models underperformed on questions that contain code snippets and MCQs compared to purely natural language questions [83]. In addition, the research highlighted similar limitations in GPT-3 models when receiving coding snippets or any other non-English input, underscoring LLMs' language dependency [6]. Because most programming queries contain non-English data as input to LLMs, these chatbots often produce lines with minor errors that require later correction by a teacher or teaching assistant. Regarding insights related to RQ-2, these findings demonstrate that while LLMs can process both code and English prompts, their responses significantly differ. Unlike English, code's fundamental structure and nature affect how LLMs perceive and generate results. Understanding these differences is crucial for effectively deploying LLMs in various applications, from programming assistance to natural language processing tasks. Furthermore, Table 12 summarizes key findings related to RQ-2 from several studies.

In conclusion, RQ-2 shows how well LLms can handle code and English questions while dealing with programming queries. LLMS can come up with answers to different kinds of questions, even ones that include code snippets. However, how well they perform on receiving coding queries depends on how well the students write the input. Even though LLMs can make outputs that make sense and are essential to the situation, they might have trouble with very specific or domain-specific tasks, especially those that require code inputs. These problems make it clear that more study and development are needed to make LLMs better at knowing and answering code-related questions with non-English data in the input. This also reports areas that can be improved, especially when making it easier for LLMs to understand and respond to code inputs correctly by training them on non-English datasets or datasets, including code snippets.

By fixing problems like wrong code reading and the limited ability to work with non-English data, LLMs can become more valuable and dependable tools for many tasks, such as helping students with complex programming tasks in computing education. Overall, some problems need to be fixed, but looking into what LLMs can do opens up new and better ways to teach programming in computing education in the future.

C. STUDENTS LEARNING- ANSWER TO RQ-3

The RQ-3 of this review focuses on the impact of low-quality outputs generated by LLM coding chatbots on the student's learning experience. It explores how such outputs' inaccuracies, confusion, frustration, and misconception can affect students' understanding of programming concepts and impede their learning progress. In response to RQ-3, the literature highlights promising opportunities and challenges that we will report in this section. A definite trend demonstrates that students frequently struggle to manually perform programming assignments, resulting in errors, wasted effort, and frustration. [1], [3], [14], [22], [37], [40], [43], [44], [48], [50], [51], [52], [53], [66], [67], [72], [78], [90]. To establish a good learning environment for students and teachers, there is an increasing demand for computer science education technologies that may address these difficulties [77], [97]. As adopting these intelligent code generators becomes increasingly prevalent in educational systems, especially for novices, many educational opportunities and challenges are being raised in the literature. It becomes imperative to gain a deep understanding of such tools. Many studies focused on usability and security aspects of the code generated by Copilot [98]. Throughout this review, numerous articles have highlighted concerns regarding the readiness of LLMs for use in introductory programming classes.

A recent study found that without guidance from an instructor, students can misinterpret potentially incorrect outputs LLMs, which may adversely affect their learning output [13]. Despite novice's positive feedback about incorporating such technologies into future classrooms, there is a chance that novices struggle with tools like Copilot [40]. Additionally, it has been revealed that GPT-3 has been deemed inadequate for providing feedback on programming tasks. Hence, it is suggested that the latest models, like GPT-4, should focus on improving these capabilities [74]. A related study shows GPT-3.5 can detect specific problems when receiving PEM explanation queries. Still, it can also report 'imaginary' non-issues, potentially misleading students and negatively impacting their learning outcomes [58]. Students and instructors generally find these tools easy to use and helpful for programming problems in computing education. Still, different challenges, such as incorrect or misleading output, can negatively impact students learning [79]. However, students and instructors believed that LLM-generated code explanations can be supplementary materials for students to study and understand code [11] in programming courses. Codex, for instance, is considered a valuable tool for creating programming exercises to enhance learning skills. However, research reports it sometimes produces minor mistakes that require instructor intervention, which need to be considered and followed while integrating these tools into computing education [10].

In the case of accuracy, length, and understandability of content generated by LLMs vs student-generated content, the study found that while the explanations were similar in size, LLM-generated content differed in accuracy and understandability as compared to student-generated content, which can affect the student's learning in different ways [11]. Another similar study compared LLM-generated solutions for introductory programming tasks with student-generated resources. The results indicated that the quality of AI-generated content was equivalent to student-generated content, suggesting that AI tools can serve as viable alternatives to traditional learning resources, potentially reducing the burden on students by providing straightforward explanations and examples [44]. In some studies, the students reported that LLM-generated code explanations can help them understand complex code, positively impacting their learning outcomes [12]. However, other articles note that the complete reliance of students on these systems for programming tasks can make students lazy and anxious, especially when the systems fail to provide correct answers [85], or they aren't able to analyze the output of these systems critically.

TABLE 12. Key findings related to RQ-2.

Reference	Supporting Statement
[58]	The LLM's responses to non-English inputs, e.g., containing coding snippets, are only marginally less accurate than its responses to questions well written in English.
[62]	It has been observed that clear and quality prompting is essential for generating accurate code with ChatGPT. Students often find it challenging to use GPT effectively to produce correct code on programming queries containing non-English content.
[83]	It is often seen that MCQs with code snippets are less successfully answered than those only written in English.
[58], [95]	Because coding inputs differ from English language inputs, LLMs can encounter difficulties when students ask to debug a given code.
[6]	When asking LLMs to explain code in which prompts contain non-English terms, the output reported minor mistakes because of less understanding of the system's capability in receiving queries containing mixed values of code synapses with English.
[55]	Students seem to want the ability to ask clarifying questions regarding errors in their code, have a conversation about their mistakes with AI coding chatbots, or have the chatbot use a chat thread to provide more insightful and contextualized responses.
[13]	The findings indicate that LLMs may be a valuable tool for improving PEMs. However, further work is required to develop methods for making high-quality adjustments and using LLMs to improve the error messages.
[82]	ChatGPT's present debugging capabilities appear to have a flaw that prevents it from learning from mistakes and fixing them based on user feedback.
[84]	LLM-generated explanations more effectively produce compile-time error explanations than run-time ones. These explanations, including mistake descriptions and analysis appropriate for a computer-educated beginner, can help resolve code-related problems.

Overall, there are still possibilities that these systems may only sometimes give correct answers for different reasons, including proper prompting, etc. [85]. Further, findings related to RQ-3 are emphasized in Table 13.

In Summary, the findings of RQ-3 can be focused since the impact of LLMs on students' learning experience and outcomes has provided valuable insights into three crucial aspects: output quality, comprehensibility, and accuracy. The findings indicate that LLM-generated outputs contain high-quality ideas, clear explanations, and correct answers, but issues such as inconsistency, ambiguity, and errors remain. Understanding these distinctions is critical for improving the incorporation of LLMs into computer science education, enabling richer learning experiences and improved student results; nonetheless, LLM-generated content can serve as supplementary material for students. Continued research and development efforts are required to address these limitations and realize LLMs' full potential as revolutionary educational instruments.

When incorporating LLMS into computing education for programming tasks, some suggestions, such as evaluating ethical values and upholding academic integrity, must be carefully considered. It is crucial to highlight the productive and accountable use of these systems based on the ethical and moral principles advocated by educational institutions. The primary concern should not be whether a student used LLMs but how they used it. Just as instructors help with homework, the aim is to ensure that kids use LLMs as a tool for learning programming rather than as a way to take shortcuts in completing their tasks. Submitting exact solutions produced by LLMs in answer to a specific request might raise concerns regarding academic dishonesty and the absence of originality in the work in computing education, which ultimately negatively affects their learning outcomes. Hence, educators and institutions must advocate for rules and practices that foster the utilization of ChatGPT to enhance students' comprehension, analytical reasoning, and innovative thinking while upholding academic integrity. This strategy promotes the proper utilization of AI technology while maintaining the integrity and excellence of education.

D. LLMS AS A DEBUGGING TOOL- ANSWER TO RQ-4

Regarding RQ-4, which questions whether computer science students may use LLMs as a debugging tool, the findings show both challenges and opportunities for using these tools in debugging tasks in computing education. These tools offer a promising way to help and guide students in debugging their code by providing context-sensitive suggestions, identifying potential errors, and explaining detected issues, which can ultimately benefit students in their programming subjects [13], [41], [42], [55], [69], [74], [76], [79], [82], [83], [84], [85], [86], [89], [91], [92], [93], [94], [96]. According to [80], students can use LLMs to get help understanding complex code and the root causes of errors and gain a detailed understanding of advanced programming topics [80]. Furthermore, LLMs can automate repetitious debugging activities, ultimately saving students time and allowing them to focus on more complex programming

TABLE 13. key findings related to RQ-3.

Reference	Supporting Statement
[39]	Most students hold a significantly positive attitude towards using ChatGPT as a tool for programming-related tasks in the course. However, they believe there are some concerns regarding various issues about usability, dependability, learning, and some ethical issues that must be focused on when using these tools in computing education.
[40]	It has also been observed that novice students find it challenging to use Copilot, express concerns about its implications, and remain optimistic about incorporating these systems into computing education in the future.
[22], [44], [60], [90]	Contrary to some beliefs, various insights indicate that AI-generated content can be useful as additional material for students in computing education. Which claims that LLMs coding chatbots can potentially reduce the workload of students and teachers dealing with complex programming tasks.
[66]	AI coding chatbots have empowered novices to write code more quickly and effectively because new students have a higher chance of generating program errors. This benefits novices, who can use this system to solve their course assignments. This can increase their engagement and interest in coding, leading to positive learning outcomes.
[67]	Students reported quick iterations and imaginative brainstorming as essential motivations for coding chatbots to solve programming tasks.
[73]	Coding chatbots can improve students' engagement and motivation to learn to program. Hence, students can demand that these tools be introduced and integrated into their curriculum in the future.
[13]	LLMs are not currently applicable in introductory programming classes because students might misinterpret potentially flawed outputs as authoritative and make harmful changes to the tasks, dramatically impacting their learning.
[79]	CodeHelp is welcomed by students, who notably value its availability and help with error solving. It is considered easy to deploy and complement rather than replace the support instructors provide students when solving programming tasks.
[81]	AI coding solutions, such as ChatGPT, have tremendous potential to provide customized instructions to enhance the learning process as an intelligent tutoring system (ITS) for students.
[85]	The use of coding chatbots in computing education to teach code writing and explain provides overall benefits that can increase students' learning process, including self-confidence, motivation, code writing skills, and academic success. In conclusion, introducing generative AI technologies into programming education might be valuable for improving programming courses' learning outcomes and efficacy.
[87]	Because the output qualities vary significantly and often contain false information, it is debatable whether this is appropriate for novice programmers. They'd need the ability to create relevant cues, and new programmers may not be able to examine input critically.
[28]	Despite these problems, it should be claimed that students can be taught to use ChatGPT constructively following educational institutions' ethics or honor codes. Ultimately, the issue should not be 'whether' the student used ChatGPT but 'how,' similar to how parents might assist their children in completing homework.
[91]	AI-generated code is on its approach to becoming an integral part of the programming education landscape. Still, we don't know how to adapt our techniques to handle the problems and maximize the benefits. It appears that programming in the future will involve a growing amount of auto-generated code, and the usage of such tools by students preparing for programming assignments and other tasks seems prominent.

tasks. Given these, employing LLMs as a debugging tool in computing education can offer several significant obstacles that need to be addressed before implementing these systems into the education system. Several concerns are considered risks, such as LLMs providing inaccurate or misleading suggestions on PEM queries, leading to further confusion or introducing new errors in students' code [99]. This type of output generated by LLMs can negatively affect students' debugging skills overall.

A recent study released in 2023, [13] claims that LLMs could be used as debugging tools to generate explanations for PEMs and suggest effective fixes for buggy code, which can be very helpful for new students to understand debugging tasks. The team collected Python error messages in this research and created code examples to generate these PEMs. They employed prompt engineering using Codex to discover prompts that could explain PEMs and propose solutions. The results showed that most of Codex's explanations

TABLE 14. Key findings related to RQ-4.

Reference	Supporting Statement
[89]	Copilot still needs improvement in repairing buggy programs because almost 95% of the buggy solutions generated by Copilot were fixed after the repair process.
[14]	Copilot frequently provides a helpful starting point that saves time when searching online. However, users often struggle to understand, change, and bug the coding outputs that Copilot generated, ultimately affecting its problem-solving efficacy.
[84]	Considering debugging as a challenging task, LLMs generated PEMs explanations help remove compile-time and run-time errors in a given code.
[82]	ChatGPT cannot develop accurate solutions for debugging code when receiving PEM prompts, which demonstrates its limits in inefficiently incorporating debugging information from Leetcode.
[80]	Codex successfully fixed far more Python bugs than JAVA (up to 23 Python bugs versus 14 Java bugs), highlighting that it can handle Python bugs far better than Java.
[13]	LLM-generated explanations and suggestions related to bug fixation are not yet ready for production use in introductory programming classes because there is a high risk that students will misunderstand the inaccurate outputs.
[69]	Recently, in 2024, a critical difference between GPT-3 and GPT-4 was that GPT-4 emphasized these minor issues more than GPT-3. GPT-3 achieved a bug detection rate of 79.4%, indicating that the code was bug-free. GPT-4, on the other hand, performs far worse, with 42.2% of bug-free code correctly.
[58]	GPt-3.5 is mainly promoted as better than Codex at finding bugs in students' code, but students should not depend on these coding chatbots to uncover all bugs and errors in a piece of code. In that case, these chatbots can be a reason for misleading students in computing education's debugging activities.
[28]	A general warning: ChatGPT may accidentally display code with some bugs as bug-free. Furthermore, it may be unaware of or unable to anticipate edge situations that could cause the code's functionality to fail under certain conditions, and it may not automatically display the best or most efficient coding technique.

for error messages were understandable, with successful explanations ranging from 67% to 100%. However, some of the explanations also contained unnecessary content like repeated sentences and extra question marks, indicating that PEM explanations and recommended fixes generated by LLMs are not yet ready for production use in introductory programming courses. There is room for improvement in using LLMs for PEM explanations across different programming languages to prepare these systems for integration with the traditional education system. An important consideration is that over-reliance on LLMs for debugging may cause students to develop critical debugging skills and problem-solving strategies, as they might become overly dependent on automated tools instead of learning to troubleshoot independently, which ultimately affects their learning skills [7]. Different steps have been taken in the research to analyze these tools, such as in [84], researchers introduced a method to help novice students understand compiler errors using LLM-generated explanations. The team developed a tool integrated with LLM APIs to explain mistakes in C programming. Their results suggested that LLM-generated explanations were more effective for compile-time errors than run-time errors. This indicates that these systems can help students understand a variety of errors and generate ideas to solve those errors but at a supplementary level.

In summary, although utilizing LLMs as debugging tools offers significant opportunities for computer science students, several obstacles necessitate resolution. LLMs can streamline troubleshooting processes, deliver contextually relevant recommendations, and enhance understanding of code. They can potentially improve students' comprehension of programming concepts. However, concerns continue regarding the precision and dependability of LLM-generated recommendations, as they may occasionally offer misleading or erroneous data and vary the response precision on input prompts. Additionally, an excessive dependence on LLMs to rectify problems may impede students' growth in critical problem-solving abilities. Irrespective of these challenges, continuous research and advancements in the capabilities of LLM present the potential for future debugging tools that are more efficient. LLMs have the potential to enhance user comprehension and accuracy in their programming endeavors, thereby positively contributing to the field of computer science education. Further, this study presents a Table14 that summarizes key findings related to RQ-4 from various studies. These findings record supporting statements of the research question.

V. FUTURE RESEARCH DIRECTIONS

This section outlines all the gaps found during this review. These gaps provide a comprehensive understanding of how various factors influence the effectiveness of LLMs-generated content of programming prompts. These findings also offer insights into usability issues and challenges computing education stakeholders face when using coding chatbots.

- **Performance Evaluation of Different GPT Models:** Investigating the performance of various GPT Models to discern which is most suitable for code explanation tasks is one of the most promising future gaps. Doing comparative analysis in this direction can shed light on the strengths and weaknesses of different versions of GPT in facilitating adequate code comprehension and learning outcomes [12], [86]. By focusing on this, researchers can provide positive insights into which GPT model produces the most accurate results and helps ask programming queries.
- **Novices Students Interaction:** In the future, exploring various ways of interaction of novices with LLMs coding chatbots and understanding how novices engage with LLMs can inform the creation of personalized learning experiences and support mechanisms to boost individual programming skills would be interesting [11], [76]. This offers promising findings into usability issues and learning challenges faced by novice students when using different coding chatbots
- **Reliability and Suitability of Feedback for PEMs tasks:** Another interesting future work found is to delve deeper into the reliability and suitability of feedback provided by LLMs for PEMs explanation tasks is an interesting future concern [87]. It can offer the potential for automating and scaling feedback provision in programming education. Apart from that, focusing on this research direction in the future can lead to the development of robust evaluation frameworks for assessing the reliability and effectiveness of LLM-generated content.
- **Generalization Issues in LLMs:** Focusing on generalization issues with LLMs to assess their performance across diverse input would be interesting in the future. It is suggested that providing multiple unseen inputs to LLMs to explore other issues related to their ability to generalize learning and reasoning capability is one of the promising future directions [96]. Also, future studies should look into LLMs' applicability and effectiveness in contexts of non-English computing education prompts. Investigate the ability of LLMs, such as ChatGPT, to generate content and provide assistance in languages other than English, catering to varied linguistic backgrounds and educational environments [95]. Working on generalization issues in LLMs in the future can help identify potential biases when receiving non-English queries and unseen problems. This can also expand the accessibility and inclusivity of LLM-based educational platforms to the non-English speaking population.
- **Performance of LLMs on Programming MCQs:** A promising future that has been found during this review is addressing the limitations and gaps in assessing LLMs coding chatbots performance on programming languages MCQs containing coding snippets is much-needed action in the future [83].
- **Complexity of Exercises and Project Specifications:** Exploring the potential of LLMs as coding assistants for handling programming exercises and more complex projects is a challenging future gap [10]. By covering this gap in the future, we can avail ourselves of different opportunities to automate the generation of computing education material related to programming subjects and others.

By focusing on these research directions, scholars can enhance our comprehension of how LLMs function as coding assistants in computing education. This advancement can lead to more efficient, inclusive learning outcomes. The identified future gaps shed light on possible hurdles, chances of improvement, and opportunities for collaboration in computing education for interested readers.

VI. KEY ASPECTS AND IMPLICATIONS

The current study systematically reviewed 72 recent studies on the adaptability of LLMs' different coding chatbots as programming assistants in computing education. While the majority of the reviewed studies in the literature focused on examining the role of LLMs, primarily targeting only ChatGPT and its role in different applications at the education level [24], [81]. The findings of this SLR provide several beneficial insights into using LLMs as a coding assistant in computing education, which can guide instructors and students in using these tools effectively. Regarding RQ-1, we identified several limitations and challenges that could impact the effectiveness of coding chatbots as programming assistants for students in computing education. The findings suggest that different chatbots have diverse strengths and weaknesses. For example, according to the significant findings of this review, ChatGPT can be effective for general code generation and explanation but requires careful verification for occasional inaccuracies in the output. These findings align with some of the studies included in this review [28], [62].

Similar findings related to ChatGPT reported in a recent SLR focusing on the specific role of ChatGPT in education highlighted that using ChatGPT in education presents many challenges concerning its accuracy and reliability in generated content [24]. Similarly, the Codex is helpful for code generation but has limitations in debugging capabilities and solving complex programming problems. However, Copilot is considered a good starting point for programming tasks, but it often generates code that requires significant modifications and understanding. For RQ-2, which is related to prompt generalization, it has been found that most of all coding chatbots depend upon the quality of input for generating accurate output, especially when it comes to coding inputs; the behaviors of these chatbots are challenging because of containing non-English content as compared to queries with only English. These findings resonate with multiple studies reviewed during this SLR, such as [6], [58], and [95]. Concerning RQ-3, we identified several concerns related to students' learning outcomes that must be considered before using these systems in the traditional education system. Findings show that novice students find these chatbots very challenging to use in regular academic activities, with some benefits of these tools for assisting them in solving their programming assignments and questions at the beginning of their journey. Using these coding chatbots in computing education to teach code writing and explain provides benefits that can increase students' learning process, including self-confidence, motivation, code writing skills, and academic success. This is also related to some of the findings reported in studies included in this SLR [40], [66].

Discussing RQ-4 for debugging tasks, debugging is a complex task that often involves understanding the interplay between various components, libraries, and systems. LLMs might struggle with issues spanning different parts of a system or involving complex dependencies. Therefore, the findings suggest that in the case of debugging, the LLMs should be used by more experienced students who can refine and debug the outputs. Understanding the suggestions received by these systems on debugging queries is challenging, which may be difficult for novices. In the case of students, an instructor should be responsible for critically analyzing output generated by LLMs for debugging and providing instructions to the students accordingly. These findings directly align with [13], [28], and [89], which are included in this study.

In conclusion, it is beneficial to incorporate these tools into computing education for programming tasks. Still, instructors should consider LLMs like ChatGPT, Codex, and Copilot as supplementary resources rather than replacements for traditional teaching methods. If students are monitored and guided properly, these tools can enhance the learning experience by providing additional explanations, debugging support, and code-generation examples. However, they should not be relied upon exclusively due to their limitations in accuracy and completeness. Students also need critical evaluation skills to analyze the output generated by LLMs in response to programming queries to avoid the incorrect solutions generated by these systems. This can be facilitated by informing instructors to design assignments that require students to verify and explain the outputs provided by LLMs. Apart from that, there is a demand for training and workshops for students and instructors to train them in practical, prompt engineering. Because clear, specific, and well-structured prompts are essential for obtaining accurate and helpful responses from LLMs. Policymakers must also address the ethical implications of using LLMs as programming assistants, including plagiarism issues and students' over-reliance on automated tools in computing education. Establishing guidelines and policies for using LLMs for programming assistance can help maintain academic integrity.

VII. LIMITATIONS

There are some limitations to this systematic literature review. It only covers the general role of different LLM coding chatbots for code generation, code explanation, and PEMs in computer science education. Still, a separate focus in each task can become in the future, for example, just focusing on code generation or explanation focusing in detail. Apart from that, we purposefully covered only literature on coding tasks in computer science education. However, the study does not cover other aspects of analyzing the capabilities of LLMs coding chatbots for different fields, such as software engineering. Second, we only cover the literature published between 2021 and 2024; future research should be addressed to see the progress for a new study orientation in this area.

VIII. CONCLUSION

In conclusion, this systematic review investigated the behavior of different LLM chatbots as a programming assistant in performing activities, including code explanations, PEM explanations, and other information for computer science education learners. 72 articles were finalized, following a rigorous PRISMA methodology, and most were published between 2021 and 2024. This research demonstrates that LLMs can be optimistically used as programming assistants for numerous tasks. These systems can help students understand the code and enhance their coding and debugging skills in the future. The findings provide the current role of LLMs as programming assistants in computer science education. Likewise, the different performances and issues regarding LLMs' behavior on programming queries submitted by subjected students are also discussed. After highlighting the problems and opportunities of utilizing LLMs for programming tasks in computer science, this survey has contributed a lot to narrowing the research gap. It strongly suggests new avenues of research in the area of study.

Ideally, the LLMs appear to be more effective chatbots for learners and teachers to comprehend and debug the code. Thus, this study suggests a potential for improvement in using LLMs for various tasks in computer science teaching; the possible areas for improvement have been presented in section V. Being the latest systematic review on the role of LLMs as programming assistants in computer science education, this study advises that it is too early to entirely rely on these systems for programming-related tasks because literature has drastically reported diverse issues regarding the correctness and reliability of the above-cited systems' behaviors on programming queries, so it is highly recommended that human control should be integrated when dealing with such systems in full swing.

DATA AVAILABILITY STATEMENT

The authors declare that no external data sources were collected. However, data related to systematic literature review may be available upon reasonable request.

DECLARATION OF INTEREST

None.

REFERENCES

[1] S. MacNeil, A. Tran, D. Mogil, S. Bernstein, E. Ross, and Z. Huang, "Generating diverse code explanations using the GPT-3 large language model," in *Proc. ACM Conf. Int. Comput. Educ. Res.*, vol. 2, Aug. 2022, pp. 37–39.

[2] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," 2017, *arXiv:1707.05589*.

[3] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, and E. Hüllermeier, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learn. Individual Differences*, vol. 103, Apr. 2023, Art. no. 102274.

[4] L. Gonçales, K. Farias, B. da Silva, and J. Fessler, "Measuring the cognitive load of software developers: A systematic mapping study," in *Proc. IEEE/ACM 27th Int. Conf. Program Comprehension (ICPC)*, May 2019, pp. 42–52.

[5] D. Helgesson, E. Engström, P. Runeson, and E. Bjarnason, "Cognitive load drivers in large scale software development," in *Proc. IEEE/ACM 12th Int. Workshop Cooperat. Hum. Aspects Softw. Eng. (CHASE)*, May 2019, pp. 91–94.

[6] M. Chen, "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

[7] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.

[8] A. Chowdhery, "PaLM: Scaling language modeling with pathways," 2022, *arXiv:2204.02311*.

[9] Y. Li, "Competition-level code generation with AlphaCode," *Science*, vol. 378, no. 6624, pp. 1092–1097, Dec. 2022.

[10] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proc. ACM Conf. Int. Comput. Educ. Res.*, vol. 1, 2022, pp. 27–43.

[11] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, and A. Hellas, "Comparing code explanations created by students and large language models," 2023, *arXiv:2304.03938*.

[12] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, and J. Leinonen, "Experiences from using code explanations generated by large language models in a web software development e-book," in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, 2023, pp. 931–937.

[13] J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B. A. Becker, "Using large language models to enhance programming error messages," in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, 2023, pp. 563–569.

[14] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Proc. CHI Conf. Human Factors Comput. Syst. Extended Abstr.*, Apr. 2022, pp. 1–7.

[15] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," in *Proc. ACM Program. Lang.*, vol. 7, Apr. 2023, pp. 85–111.

[16] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," 2019, *arXiv:1904.09751*.

[17] D. Ippolito, R. Kriz, M. Kustikova, J. Sedoc, and C. Callison-Burch, "Comparison of diverse decoding methods from conditional language models," 2019, *arXiv:1906.06362*.

[18] N. N. M. Kasim and F. Khalid, "Choosing the right learning management system (LMS) for the higher education institution context: A systematic review," *Int. J. Emerg. Technol. Learn. (iJET)*, vol. 11, no. 6, p. 55, Jun. 2016.

[19] M. Sallam, "ChatGPT utility in healthcare education, research, and practice: Systematic review on the promising perspectives and valid concerns," *Healthcare*, vol. 11, no. 6, p. 887, 2023. [Online]. Available: https://www.mdpi.com/2227-9032/11/6/887

[20] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, "BloombergGPT: A large language model for finance," 2023, *arXiv:2303.17564*.

[21] M. M. Rahman and Y. Watanobe, "ChatGPT for education and research: Opportunities, threats, and strategies," *Appl. Sci.*, vol. 13, no. 9, p. 5783, May 2023.

[22] S. MacNeil, A. Tran, J. Leinonen, P. Denny, J. Kim, A. Hellas, S. Bernstein, and S. Sarsa, "Automatically generating CS learning materials with large language models," 2022, *arXiv:2212.05113*.

[23] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," 2023, *arXiv:2308.10620*.

[24] C. K. Lo, "What is the impact of ChatGPT on education? A rapid review of the literature," *Educ. Sci.*, vol. 13, no. 4, p. 410, Apr. 2023.

[25] Y. Liu, T. Han, S. Ma, J. Zhang, Y. Yang, J. Tian, H. He, A. Li, M. He, Z. Liu, Z. Wu, L. Zhao, D. Zhu, X. Li, N. Qiang, D. Shen, T. Liu, and B. Ge, "Summary of ChatGPT-related research and perspective towards the future of large language models," *Meta-Radiol.*, vol. 1, no. 2, Sep. 2023, Art. no. 100017.

[26] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, and S. Mirjalili, "A survey on large language models: Applications, challenges, limitations, and practical usage," *TechRxiv*, vol. 1, Jul. 2023.

[27] L. Yan, L. Sha, L. Zhao, Y. Li, R. Martinez-Maldonado, G. Chen, X. Li, Y. Jin, and D. Gašević, "Practical and ethical challenges of large language models in education: A systematic scoping review," *Brit. J. Educ. Technol.*, vol. 55, no. 1, pp. 90–112, Jan. 2024.

[28] J. G. Meyer, R. J. Urbanowicz, P. C. N. Martin, K. O'Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez, and J. H. Moore, "ChatGPT and large language models in academia: Opportunities and challenges," *BioData Mining*, vol. 16, no. 1, p. 20, Jul. 2023.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, and M. Funtowicz, "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empirical Methods Natural Lang. Process., Syst. Demonstrations*, 2020, pp. 38–45.

[31] V. Cohen and A. Gokaslan, "OpenGPT-2: Open language models and implications of generated text," *XRDS: Crossroads, ACM Mag. Students*, vol. 27, no. 1, pp. 26–30, Sep. 2020.

[32] E. Sezgin, J. Sirrianni, and S. L. Linwood, "Operationalizing and implementing pretrained, large artificial intelligence linguistic models in the U.S. health care system: Outlook of generative pretrained transformer 3 (GPT-3) as a service model," *JMIR Med. Informat.*, vol. 10, no. 2, Feb. 2022, Art. no. e32875.

[33] R. A. Frank, P. M. Bossuyt, and M. D. F. McInnes, "Systematic reviews and meta-analyses of diagnostic test accuracy: The PRISMA-DTA statement," *Radiology*, vol. 289, no. 2, pp. 313–314, Nov. 2018.

[34] R. E. O'Dea, M. Lagisz, M. D. Jennions, J. Koricheva, D. W. Noble, T. H. Parker, J. Gurevitch, M. J. Page, G. Stewart, and D. Moher, "Preferred reporting items for systematic reviews and meta-analyses in ecology and evolutionary biology: A Prisma extension," *Biol. Rev.*, vol. 96, no. 5, pp. 1695–1722, 2021.

[35] A. Ahmed, R. Xi, M. Hou, S. A. Shah, and S. Hameed, "Harnessing big data analytics for healthcare: A comprehensive review of frameworks, implications, applications, and impacts," *IEEE Access*, vol. 11, pp. 112891–112928, 2023.

[36] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering—A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009.

[37] D. H. Smith IV and C. Zilles, "Code generation based grading: Evaluating an auto-grading mechanism for 'explain-in-plain-english' questions," 2023, *arXiv:2311.14903*.

[38] J. Savelka, A. Agarwal, M. An, C. Bogart, and M. Sakr, "Thrilled by your progress! Large language models (GPT-4) no longer struggle to pass assessments in higher education programming courses," 2023, *arXiv:2306.10073*.

[39] I. Joshi, R. Budhiraja, H. D Akolekar, J. S. Challa, and D. Kumar, "'It's not like Jarvis, but it's pretty close!'—Examining ChatGPT's usage among undergraduate students in computer science," 2023, *arXiv:2311.09651*.

[40] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos, "'It's weird that it knows what I want': Usability and interactions with copilot for novice programmers," 2023, *arXiv:2304.02491*.

[41] G.-J. Hwang and C.-Y. Chang, "A review of opportunities and challenges of chatbots in education," *Interact. Learn. Environments*, vol. 31, no. 7, pp. 4099–4112, Oct. 2023.

[42] J. D. Zamfirescu-Pereira, L. Qi, B. Hartmann, J. DeNero, and N. Norouzi, "Conversational programming with LLM-powered interactive support in an introductory computer science course," in *Proc. NeurIPS Workshop Generative AI Educ. (GAIED)*, 2023, pp. 1–10.
[43] A. Mehta, N. Gupta, A. Balachandran, D. Kumar, and P. Jalote, "Can ChatGPT play the role of a teaching assistant in an introductory programming course?" 2023, *arXiv:2312.07343*.
[44] P. Denny, H. Khosravi, A. Hellas, J. Leinonen, and S. Sarsa, "Can we trust AI-generated educational content? Comparative analysis of human and AI-generated learning resources," 2023, *arXiv:2306.10509*.
[45] P. Denny, B. A. Becker, J. Leinonen, and J. Prather, "Chat overflow: Artificially intelligent models for computing education-renaissance or apocAIypse?" in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, vol. 1, 2023, pp. 3–4.
[46] C. W. Okonkwo and A. Ade-Ibijola, "Chatbots applications in education: A systematic review," *Comput. Educ., Artif. Intell.*, vol. 2, Feb. 2021, Art. no. 100033.
[47] Z. H. Ipek, A. Gözüm, S. Papadakis, and M. Kallogiannakis, "Educational applications of the ChatGPT AI system: A systematic review research," *Educ. Process Int. J.*, vol. 12, no. 3, pp. 26–55, 2023.
[48] M.-D. Popovici, "ChatGPT in the classroom. Exploring its potential and limitations in a functional programming course," *Int. J. Hum. Comput. Interact.*, pp. 1–12, 2023, doi: 10.1080/10447318.2023.2269006.
[49] I. Joshi, R. Budhiraja, H. Dev, J. Kadia, M. O. Ataullah, S. Mitra, D. Kumar, and H. D. Akolekar, "ChatGPT in the classroom: An analysis of its strengths and weaknesses for solving undergraduate computer science questions," 2023, *arXiv:2304.14993*.
[50] A. Tran, K. Angelikas, E. Rama, C. Okechukwu, D. H. Smith, and S. MacNeil, "Generating multiple choice questions for computing courses using large language models," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2023, pp. 1–8.
[51] M. Kazemitabaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman, "CodeAid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs," 2024, *arXiv:2401.11314*.
[52] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, and B. Akram, "Detecting ChatGPT-generated code in a CS1 course," in *Proc. Workshop Empowering Educ. LLMS-Next-Gen Interface Content Gener.*, 2023, pp. 53–63.
[53] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price, and B. Akram, "Detecting ChatGPT-generated code submissions in a CS1 course using machine learning models," in *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, Mar. 2024, pp. 526–532.
[54] R. Wu and Z. Yu, "Do AI chatbots improve students learning outcomes? Evidence from a meta-analysis," *Brit. J. Educ. Technol.*, vol. 55, no. 1, pp. 10–33, Jan. 2024.
[55] B. Kimmel, A. Geisert, L. Yaro, B. Gipson, T. Hotchkiss, S. Osae-Asante, H. Vaught, G. Wininger, and C. Yamaguchi, "Enhancing programming error messages in real time with generative AI," 2024, *arXiv:2402.08072*.
[56] B. Jury, A. Lorusso, J. Leinonen, P. Denny, and A. Luxton-Reilly, "Evaluating LLM-generated worked examples in an introductory programming course," in *Proc. 26th Australas. Comput. Educ. Conf.*, 2024, pp. 77–86.
[57] B. Reeves, S. Sarsa, J. Prather, P. Denny, B. A. Becker, A. Hellas, B. Kimmel, G. Powell, and J. Leinonen, "Evaluating the performance of code generation models for solving parsons problems with small prompt variations," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, vol. 1, 2023, pp. 299–305.
[58] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva, "Exploring the responses of large language models to beginner Programmers' help requests," 2023, *arXiv:2306.05715*.
[59] J. Savelka, A. Agarwal, C. Bogart, and M. Sakr, "From GPT-3 to GPT-4: On the evolving efficacy of LLMs to answer multiple-choice questions for programming classes in higher education," in *Proc. Int. Conf. Comput. Supported Educ.* Cham, Switzerland: Springer, 2023, pp. 160–182.
[60] B. A. Becker, M. Craig, P. Denny, H. Keuning, N. Kiesler, J. Leinonen, A. Luxton-Reilly, J. Prather, and K. Quille, "Generative AI in introductory programming," in *Computer Science Curricula 2023*. New York, NY, USA: ACM, 2024.
[61] J. Sheard, P. Denny, A. Hellas, J. Leinonen, L. Malmi, and Simon, "Instructor perceptions of AI code generation tools—A multi-institutional interview study," in *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, Mar. 2024, pp. 1223–1229.
[62] J. Prather, P. Denny, J. Leinonen, D. H. Smith IV, B. N. Reeves, S. MacNeil, B. A. Becker, A. Luxton-Reilly, T. Amarouche, and B. Kimmel, "Interactions with prompt problems: A new way to teach programming with large language models," 2024, *arXiv:2401.10759*.
[63] Q. Ma, T. Wu, and K. Koedinger, "Is AI the better programming partner? Human-human pair programming vs. human-AI pAIr programming," 2023, *arXiv:2306.05153*.
[64] I. Hou, O. Man, S. Mettille, S. Gutierrez, K. Angelikas, and S. MacNeil, "More robots are coming: Large multimodal models (ChatGPT) can solve visually diverse images of parsons problems," in *Proc. 26th Australas. Comput. Educ. Conf.*, Jan. 2024, pp. 29–38.
[65] P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, and B. N. Reeves, "Prompt problems: A new programming exercise for the generative AI era," 2023, *arXiv:2311.05943*.
[66] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI code generators on supporting novice learners in introductory programming," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2023, pp. 1–23.
[67] I. Hou, S. Mettille, O. Man, Z. Li, C. Zastudil, and S. MacNeil, "The effects of generative AI on computing students' help-seeking preferences," in *Proc. 26th Australas. Comput. Educ. Conf.*, Jan. 2024, pp. 39–48.
[68] M. Amoozadeh, D. Daniels, D. Nam, A. Kumar, S. Chen, M. Hilton, S. S. Ragavan, and M. A. Alipour, "Trust in generative AI among students: An exploratory study," 2023, *arXiv:2310.04631*.
[69] S. MacNeil, P. Denny, A. Tran, J. Leinonen, S. Bernstein, A. Hellas, S. Sarsa, and J. Kim, "Decoding logic errors: A comparative study on bug detection by students and large language models," in *Proc. 26th Australas. Comput. Educ. Conf.*, 2024, pp. 11–18.
[70] C. A. Philbin, "Exploring the potential of artificial intelligence program generators in computer programming education for students," *ACM Inroads*, vol. 14, no. 3, pp. 30–38, Sep. 2023.
[71] V. Agarwal, M. K. Garg, S. Dharmavaram, and D. Kumar, "'Which LLM should I use?': Evaluating LLMs for tasks performed by undergraduate computer science students," 2024, *arXiv:2402.01687*.
[72] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models. (2022)," 2022, *arXiv:2206.15000*.
[73] A. Shynkar, "Investigating the influence of code generating technologies on learning process of novice programmers in higher education computer science course," B.S. thesis, Univ. Twente, 2023. [Online]. Available: https://essay.utwente.nl/96067/
[74] R. Balse, B. Valaboju, S. Singhal, J. M. Warriem, and P. Prasad, "Investigating the potential of GPT-3 in providing feedback for programming assessments," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, 2023, pp. 292–298.
[75] S. Imai, "Is GitHub copilot a substitute for human pair-programming? An empirical study," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2022, pp. 319–321.
[76] P. Denny, V. Kumar, and N. Giacaman, "Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language," in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ.*, Mar. 2023, pp. 1136–1142.
[77] P. Denny, J. Prather, B. A. Becker, C. Mooney, J. Homer, Z. C. Albrecht, and G. B. Powell, "On designing programming error messages for novices: Readability and its constituent factors," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2021, pp. 1–15.
[78] J. Finnie-Ansley, P. Denny, A. Luxton-Reilly, E. A. Santos, J. Prather, and B. A. Becker, "My AI wants to know if this will be on the exam: Testing OpenAI's codex on CS_2 programming exercises," in *Proc. 25th Australas. Comput. Educ. Conf.*, 2023, pp. 97–104.
[79] M. Liffiton, B. Sheese, J. Savelka, and P. Denny, "CodeHelp: Using large language models with guardrails for scalable support in programming classes," 2023, *arXiv:2308.06921*.
[80] J. A. Prenner and R. Robbes, "Automatic program repair with OpenAI's codex: Evaluating QuixBugs," 2021, *arXiv:2111.03922*.
[81] B. Qureshi, "Exploring the use of ChatGPT as a tool for learning and assessment in undergraduate computer science curriculum: Opportunities and challenges," 2023, *arXiv:2304.11214*.
[82] F. A. Sakib, S. H. Khan, and A. H. M. R. Karim, "Extending the frontier of ChatGPT: Code generation and debugging," 2023, *arXiv:2307.08260*.
[83] J. Savelka, A. Agarwal, C. Bogart, and M. Sakr, "Large language models (GPT) struggle to answer multiple-choice questions about code," 2023, *arXiv:2303.08033*.

[84] A. Taylor, A. Vassar, J. Renzella, and H. Pearce, "Dcc-help: Generating context-aware compiler error explanations with large language models," 2023, *arXiv:2308.11873*.
[85] R. Yilmaz and F. G. Karaoglan Yilmaz, "Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning," *Comput. Hum. Behav., Artif. Humans*, vol. 1, no. 2, Aug. 2023, Art. no. 100005.
[86] C. Fang, N. Miao, S. Srivastav, J. Liu, R. Zhang, R. Fang, Asmita, R. Tsang, N. Nazari, H. Wang, and H. Homayoun, "Large language models for code analysis: Do LLMs really do their job?" 2023, *arXiv:2310.12357*.
[87] N. Kiesler, D. Lohr, and H. Keuning, "Exploring the potential of large language models to generate formative programming feedback," 2023, *arXiv:2309.00029*.
[88] T. Phung, V.-A. Pădurean, J. Cambronero, S. Gulwani, T. Kohn, R. Majumdar, A. Singla, and G. Soares, "Generative AI for programming education: Benchmarking ChatGPT, GPT-4, and human tutors," *Int. J. Manage.*, vol. 21, no. 2, 2023, Art. no. 100790.
[89] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. Jiang, "GitHub copilot AI pair programmer: Asset or liability?" *J. Syst. Softw.*, vol. 203, Sep. 2023, Art. no. 111734.
[90] P. Denny, H. Khosravi, A. Hellas, J. Leinonen, and S. Sarsa, "Human vs machine: Comparison of student-generated and AI-generated educational content," 2023, *arXiv:2306.10509*.
[91] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard-or at least it used to be: Educational opportunities and challenges of AI code generation," in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, 2023, pp. 500–506.
[92] J. Savelka, P. Denny, M. Liffiton, and B. Sheese, "Efficient classification of student help requests in programming courses using large language models," 2023, *arXiv:2310.20105*.
[93] M. Kazemitabaar, X. Hou, A. Henley, B. J. Ericson, D. Weintrop, and T. Grossman, "How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment," 2023, *arXiv:2309.14049*.
[94] N. Al Madi, "How readable is model-generated code? Examining readability and visual inspection of GitHub copilot," in *Proc. 37th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Oct. 2022, pp. 1–5.
[95] M. S. Orenstrakh, O. Karnalim, C. A. Suarez, and M. Liut, "Detecting LLM-generated text in computing education: A comparative study for ChatGPT cases," 2023, *arXiv:2307.07411*.
[96] H. Tian, W. Lu, T. On Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé, "Is ChatGPT the ultimate programming assistant—How far is it?" 2023, *arXiv:2304.11938*.
[97] A. Hellas, J. Leinonen, and P. Ihantola, "Plagiarism in take-home exams: Help-seeking, collaboration, and systematic cheating," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, 2017, pp. 238–243.
[98] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 754–768.
[99] Y. Zhang, S. Sun, M. Galley, Y.-C. Chen, C. Brockett, X. Gao, J. Gao, J. Liu, and B. Dolan, "DialoGPT: Large-scale generative pre-training for conversational response generation," 2019, *arXiv:1911.00536*.

FARMAN ALI PIRZADO (Member, IEEE) received the bachelor's and master's degrees from the University of Sindh, in 2012 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Tecnológico de Monterrey, Mexico. He is on study leave from his position as a Lecturer with the Department of Computer Science, Muhammad Ali Jinnah University, Karachi. Previously, he was with the IBA Community College, Computer Science Department, as a Lecturer. His research interests include generative AI, AI in computing education, and computational thinking.

AWAIS AHMED (Member, IEEE) received the B.S. and M.S. degrees in computer science from the National University of Computer and Emerging Sciences (NUCES), in 2016 and 2019, respectively, and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China (UESTC). With six and half years of experience in research and university-level teaching, he was a Lecturer with Muhammad Ali Jinnah University, Karachi. Previously, he was a Research Associate and an Instructor with NUCES-FAST. His research interests include big data healthcare, multimodal data, data science, large language models, and natural language processing.

ROMÁN ALEJANDRO MENDOZA-URDIALES (Member, IEEE) is currently a dedicated Post-doctoral Researcher with the Computer Science Department, Tecnológico de Monterrey. With expertise in advanced artificial intelligence, he contributes valuable insights to the Research Group in Advanced Artificial Intelligence.

HUGO TERASHIMA-MARIN (Senior Member, IEEE) received the B.Sc. degree in computational systems from the Tecnológico de Monterrey, Campus Monterrey, in 1982, the M.Sc. degree in computer science from the University of Oklahoma, in 1987, the M.Sc. degree in information technology and knowledge-based systems from the University of Edinburgh, in 1994, and the Ph.D. degree in informatics from the Tecnológico de Monterrey, Campus Monterrey, in 1998. He is currently a Full Professor with the School of Engineering and Sciences Monterrey and a member of the Research Group in Advanced Artificial Intelligence. He is also a member of the National System of Researchers (Rank II), Mexican Academy of Sciences, and Mexican Academy of Computing. His research interests include computational intelligence, heuristics, metaheuristics and hyper-heuristics for combinatorial optimization, automated generation of algorithms and applications of artificial intelligence, and machine learning. He has been a principal investigator of various projects for industry and CONACyT. He has current collaboration with research groups with the University of Nottingham, University of Stirling, University of Edinburgh-Napier, University of Texas at San Antonio, the University of Houston, the University Andrés Bello in Santiago de Chile, and Chinese Academy of Sciences. He has published more than 125 research papers in international journals and conferences. He has supervised five Ph.D. dissertations and 38 master's thesis. In the past, he has been the Director of the Graduate Program in Computer Science, the M.Sc. in Intelligent Systems, the Ph.D. in Artificial Intelligence, the Ph.D. in Information Technology and Communications, the Ph.D. Programs, and Graduate Programs with the Tecnológico de Monterrey, Campus Monterrey; and the Leader of the Research Group in Intelligent Systems.