

Pedagogical Roles of Large Language Models in Computing Education: A Systematic Literature Review

Opetunde Ibitoye

ibitoyoo@mail.uc.edu

University of Cincinnati

Amir-reza Asadi

University of Cincinnati

Joel Appiah

University of Cincinnati

Taiwo Akinremi

University of Cincinnati

Popoola Saheed

University of Cincinnati

Systematic Review

Keywords: Large Language Models, Computing Education, Artificial Intelligence in Education, Personalized Learning, Cognitive Scaffolding, Human–AI Collaboration, AI Literacy

Posted Date: March 16th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-9107566/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Pedagogical Roles of Large Language Models in Computing Education: A Systematic Literature Review

OPETUNDE IBITOYE, AMIR REZA ASAD, JOEL APPIAH, TAIWO AKINREMI, and SAHEED POPOOLA, University of Cincinnati, USA

Large Language Models (LLMs) are rapidly transforming educational practices, offering new possibilities for personalized, interactive, and adaptive learning. However, their specific pedagogical roles and impact on student learning processes in computing education remain underexplored. This systematic literature review investigates how LLM-based tools support student learning, specifically highlighting pedagogical support mechanisms, student strategies, and factors shaping tool interaction. Through a rigorous analysis of 46 articles identified from the ACM, IEEE Xplore, and Scopus databases, this study employed a thematic analysis methodology to categorize the current evidence.

The results identified five primary support mechanisms: adaptive scaffolding, interactive pedagogical structures, formative feedback generation, personalization, and structured learning activities. These mechanisms facilitate a deeper conceptual understanding, learning transfer, and critical thinking. Conversely, the absence of these pedagogical supports can lead to negative outcomes, including skill atrophy, dependency, cognitive offloading, and reduced self-efficacy. Furthermore, the review identifies learner characteristics, AI literacy, tool design, and socio-institutional context as critical factors determining the quality of student-LLM interactions. The study concludes that when integrated with appropriate pedagogical frameworks, LLMs act as dynamic co-learners that scaffold learning, ultimately transforming computing education into a more adaptive and reflective ecosystem.

Additional Key Words and Phrases: Large Language Models, Computing Education, Artificial Intelligence in Education, Personalized Learning, Cognitive Scaffolding, Human-AI Collaboration, AI Literacy

ACM Reference Format:

Opetunde Ibitoye, Amir Reza Asad, Joel Appiah, Taiwo Akinremi, and Saheed Popoola. 2026. Pedagogical Roles of Large Language Models in Computing Education: A Systematic Literature Review. 1, 1 (March 2026), 34 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

1 Introduction

Large Language Models (LLMs) are advanced AI systems capable of processing and generating human-like text, code, and other artifacts [102]. LLM-based tools can revolutionize education and transform the way students learn, engage, and interact with educational content. Therefore, it is not surprising that LLMs systems are increasingly integrated into educational settings because they provide opportunities to improve teaching and learning [91]. In computing education, LLMs have a unique capability to support dynamic learning experiences, such as providing real-time feedback,

Authors' Contact Information: Opetunde Ibitoye, ibitoyoo@mail.uc.edu; Amir Reza Asad, asadiaa@mail.uc.edu; Joel Appiah, appiahjk@mail.uc.edu; Taiwo Akinremi, akinretp@mail.uc.edu; Saheed Popoola, popoolso@ucmail.uc.edu, University of Cincinnati, Cincinnati, Ohio, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM XXXX-XXXX/2026/3-ART

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

scaffolding programming tasks (i.e., structured guidance to build skills), and personalizing instruction to meet diverse student needs [8, 71, 89]. These capabilities align with the growing demand for innovative pedagogies that leverage emerging technologies to improve student processes.

Learning processes encompass cognitive (knowledge acquisition), metacognitive (learning self-regulation), and affective (motivation and engagement) dimensions [68]. Learning support across these dimensions is significant in computing education because complex problem-solving and skill development are central to computing pedagogy [80]. For example, LLMs can help structure curricula and provide adaptive support for programming assignments [24, 65]. However, challenges can limit the impact of LLMs on learning. Studies suggest that LLMs can encourage surface-level learning or pose risks to academic integrity if they are not implemented thoughtfully [38, 76]. Furthermore, the cognitive and metacognitive demands of validating AI-generated outputs require targeted support to ensure effective use [48].

These challenges create a need to understand the impact of LLMs on the learning processes in computing education. Existing systematic reviews have examined adoption patterns across levels and sub-disciplines, common models/languages, assessment roles, general benefits/challenges, instructional integration, and tool limitations as coding assistants [4, 30, 67, 72, 76, 78]. These studies offer strong overviews of the broad impact and use of LLMs in computing. However, to the best of our knowledge, no systematic literature review has specifically synthesized: (1) how LLM-based tools support learning; (2) the pedagogical strategies, scaffolding techniques, or built-in guardrails embedded within LLM-based tools to support meaningful learning processes in computing education; (3) the key challenges in leveraging these tools for deep learning tasks (beyond surface-level output generation); or (4) the multifaceted factors (e.g., tool affordances, learner characteristics, and contextual elements) that shape students' interactions and usage patterns with such tools.

This systematic literature review (SLR) addresses this gap following the guidelines of Kitchenham and Charters [42]. Synthesizes evidence to understand how LLMs support or undermine productive learning processes in computing education. In addition, the challenges and opportunities for using LLMs to support learning are discussed. This review is guided by the following research questions: The study also identifies challenges, opportunities, and effective LLM implementation strategies that enhance learning. Specifically, this review will be guided by the following research questions:

- RQ1 What LLM-based tools have been developed or employed to support learning in computing education?
- RQ2 How do LLM-based tools support learning processes in computing education?
- RQ3 What factors influence how students use LLM-based tools in computing education?
- RQ4 What are the challenges and limitations of integrating LLM-Based Tools to Support Learning in computing education?

This study aims to contribute to the theoretical and practical understanding of the role of LLMs in fostering meaningful learning. The findings of this study will guide educators and researchers in leveraging LLM tools responsibly and effectively.

2 Literature Review

LLMs have a transformative effect on the student learning process in computing education. This section reviews existing studies related to LLM in computing education. The review will be presented in narrative form to establish a foundation for understanding the current theoretical frameworks in this domain. We discuss key concepts within the human learning process and construct a lens to analyze the application of LLMs to human learning.

2.1 Human Learning Process

Learning is the process of acquiring new knowledge, skills, understanding, values, behaviors, or attitudes through experience, observation, or study [99]. Several researchers have conceptualized learning. Benjamin Bloom [68] defined learning as a set of cognitive activities that can be categorized into a hierarchy of increasing complexity. The hierarchical levels are knowledge recall, comprehension, knowledge use in new situations, analysis, synthesis, and evaluation. Another important concept is that learning does not occur suddenly, or in isolation. Rather, learning has a process termed the “learning process.” Philips et al. [70] stated that learning processes are all cognitive activities that students undertake to develop their understanding, including the way these activities are carried out. McCormick and Kiss [55] defined learning process as a set of cognitive activities that include how students engage with ideas, concepts and resources to develop knowledge through interaction. There are two types of learning: active and passive. Passive learning involves learning by reading, listening, or watching. Active learning describes an interactive process of learning through experimentation [17]. At its core, learning is an active process that involves encountering new information, paying attention to it, coordinating it with what is already known, storing it in memory, and applying it [95]. A typical example of learning can be observed when an individual seeks to understand a new concept or acquire a new skill. For instance, when a person encounters an unfamiliar scientific term while reading, they may consult a dictionary or a reliable source to determine its meaning. Through this process, individuals gain new information and strengthen their comprehension and vocabulary.

Laurillard [47] and Cate et al. [94] further described learning as an adaptive decision-making process shaped by cognitive, affective and metacognitive factors [47, 94]. Bloom’s taxonomy further reiterates these factors as dimensions in their learning taxonomy. According to Bloom’s taxonomy, the cognitive dimension of learning processes states that “learning occurs when the learner acquires knowledge of a topic or subject matter through processing information by reading, listening, thinking, memorizing facts, relating new facts to existing knowledge, analyzing problems, acquiring psychomotor skills, etc.” The affective component of learning deals with the learner’s motivation to start and persist in concentrated learning as well as the emotional relationship of the students with the learning materials and readiness to study. The metacognitive component involves the ability of the learner to independently plan study activities, monitor and evaluate progress, and address their own personal lack of knowledge. At its core, the learning process refers to the ways in which individuals acquire, process, and retain knowledge or skills through experience, instruction, and observation [92].

2.2 Student Learning Process in Higher Education Computing Education

Learning is an active, constructive, emotional, self-regulated, situated, and social process [96]. In the context of the computing education curriculum, learning is a multifaceted process that involves several forms of cognitive transformation. It is expected that in each phase, students develop computational thinking abilities and other higher-order thinking skills that move them from foundational surface-level (novice) to deep comprehension (expert). A practical illustration of this learning process can be observed in computing programs accredited by the Accreditation Board for Engineering and Technology (ABET), which emphasize a three-phase process: (1) foundational skill building, (2) applied learning through integration, and (3) advanced specialization. Students engage in a series of learning activities to develop their understanding, apply their knowledge, and specialize in a domain. This process often begins with foundational courses, such as Programming I, Programming II, and Data structures and algorithms. These courses embody a constructivist approach, in which knowledge is accumulated through active engagement and iterative practice.

At its core, this process requires students to construct mental models of computational concepts through hands-on activities. This process also enables students to transition from novice-level syntactic understanding to advanced algorithmic thinking and system design.

2.2.1 Foundational Skill Building. This is the initial phase in which students are introduced to programming. In this phase, students learn fundamental programming concepts through direct application. The learning processes or activities in this phase involve interacting with course materials, performing lab assessments, and receiving feedback [86]. A very important activity is having students attend structured laboratory sessions to reinforce active learning and enable learners to apply theoretical principles to practical exercises. This is also called learning through reflection and action. This stage is characterized by regular and supervised practice sessions, use of standard development tools, focus on problem-solving skill development within a controlled environment, and mastery of the syntax and semantics of the chosen programming language.

2.2.2 Applied Learning Through Integration. This is the second phase, in which learners advance to intermediate programming or programming II. The learning process has shifted toward project-based and collaborative learning to understand more advanced concepts, such as the object-oriented programming paradigm [86]. In this phase, learners move from isolated exercises to comprehensive scenarios that require collaborative problem-solving, integration of multiple concepts (e.g., object-oriented principles, data organization), application of theoretical knowledge to practical challenges, and development of architectural thinking and software design skills. This transition mirrors the development of higher-order cognitive skills highlighted in Bloom's taxonomy as learners move from comprehension and application to analysis and synthesis. In the later aspect of this phase, learners transition from applying technical concepts and showing technical competency to an efficient solution design. Here, students engage in learning activities that enable them to acquire skills such as the selection and implementation of appropriate algorithms and data structures, consideration of performance constraints (time and space complexity), advanced analytical and optimization skills, and the ability to evaluate trade-offs for multiple design decisions.

2.2.3 Advanced Application. After learners complete the foundational sequence, they achieve readiness for discipline-specific advanced coursework that enables them to pursue specialized areas within their chosen computing field. When students complete all of these phases, they develop computational thinking skills and technical proficiency. Students also acquire essential professional competencies such as time management, communication, and collaborative problem-solving skills.

2.2.4 The Novice to Expert Paradigm in Programming. Understanding the student learning process in computing requires a framework that accounts for profound cognitive changes that occur over time. The novice-expert paradigm drawn from cognitive science provides such a lens by comparing the knowledge levels and problem-solving strategies of beginners and professionals. A novice is defined as an individual who is new to a domain and possesses fragmented knowledge that is often context-specific. However, experts hold a highly organized and integrated body of knowledge that can be applied effortlessly. The fundamental difference between novices and experts is how knowledge is structured and utilized. Classic studies of chess masters revealed that experts perceive the board as a set of meaningful patterns and strategic configurations, whereas a novice perceives the board as a collection of individual pieces [27]. Similarly, novice programmers typically approach coding using a linear and localized cognitive framework. Their focus is predominantly on the surface level and syntactic features of the programming language, such as the rules for writing a for loop and the correct syntax for a function call. When faced with a problem, novices tend to employ a bottom-up strategy, where they attempt to write code line-by-line without a clear, overarching plan [81].

This cognitive state of novices explains many of the common struggles that beginners face. They require a great deal of time to solve problems and are prone to making numerous errors because each line of code presents a separate cognitive challenge. The cognitive architecture of expert programmers is fundamentally different. It is characterized by deep, interconnected knowledge organized into hierarchical structures [56]. Experts possess syntactic, semantic, and schematic knowledge that captures the what and why behind a code's function. Experts also have a vast understanding of programming plans, patterns, and idioms. This structured knowledge enables a top-down problem-solving approach, whereby an expert first seeks to understand the overall goal of the problem, and then devises a high-level plan. The high-level plan is then broken down into subproblems that map to the existing schema [2]. A computing curriculum is a guided process designed to facilitate this transition from a surface-level understanding to a deep and structured comprehension.

In summary, the student learning process in computing education is characterized by cognitive reorganization, which aims to restructure students' thinking. This process begins with instilling the fundamental principles of computational thinking in students and moving learners from a fragmented syntactic understanding of code to a deep schematic knowledge base. This process enables students to build, analyze, and manipulate complex systems with multiple levels of abstraction.

2.3 Enhancing Student Learning Process in Computing Education

There has been a long-standing effort to develop pedagogical interventions, methods, tools and systems that support and enhance learning. In this section, we describe existing techniques developed to enhance students' learning in computing education. These techniques include the use of notional machines, teaching assistants, gamification, knowledge repositories, question-and-answer websites, pair programming, and unplugged activities.

2.3.1 Notional Machines. A notional machine is a pedagogic device that assists in understanding some aspects of programs or programming. It is typically used to support the explanation of a programming construct or understand the semantics of a program [28]. For example, a variable can be represented as a box with a label, and an assignment operation copies or moves a value into the box [28]. The purpose of a notional machine is to support students' learning of computational concepts. A crucial aspect of a notional machine is that it should simplify an actual concept or skill as an aid to understanding [28].

2.3.2 Gamification. This involves the application of game design elements such as points, badges, leaderboards, and progress bars, to educational contexts [46]. In computing education, gamification is used to increase student motivation and participation in activities such as coding practice, forums contribution, and assignment completion. Well-designed gamified systems can provide immediate feedback, encourage persistence, and foster a sense of accomplishment, which is beneficial for subjects that many students find difficult [34].

2.3.3 Unplugged Activities. These are hands-on, screen-free activities that teach fundamental concepts of computer science [16]. For example, students might create "binary bracelets" with beads to understand data representations or follow a sequence of dance moves on index cards to learn about algorithms and debugging. Unplugged activities are exceptionally valuable for isolating abstract computational concepts from the high cognitive overhead associated with programming syntax. This allows students to grasp the core logic of a concept, such as conditional statements or loops, before they attempt to implement it in code.

2.3.4 Teaching Assistants. In current undergraduate educational settings, an instructor is usually aided by teaching assistants or graders to provide instruction, conduct labs, grade exams, and guide

students through a comprehensive learning experience. Although several activities for these roles can be automated (e.g., auto-graders, recorded lectures), teaching assistants play a critical role in the educational experience. Teaching assistants provide crucial support to students, continuously monitor their performance, and guide them through their learning journey [74]. For example, if students need help with a programming task, teaching assistants can provide an explanation [86]. However, doing this manually can become difficult based on the diverse needs of the student and the size of the class [86]. Therefore, providing a dedicated human TA to all students is extremely human-intensive and untenable in most settings [74]. An alternative is to use LLMs to provide a virtual TA that is always available to the students.

2.4 Large Language Models and Learning in Computing Education

Large Language Models (LLMs) are generative artificial intelligence technologies capable of generating objects, artifacts, and interactions with users in natural language. At its core, an LLM is a probability distribution over sequences of words learned using a deep neural network with a large number of parameters. These networks are trained on vast amounts of data (e.g., text, images, or code) with objectives such as predicting masked tokens or autoregressive modeling (predicting the next token given preceding tokens) [101].

The ability of LLMs to generate artifacts, such as text, code, images, and videos, based on natural language prompts has changed the trajectory of computing and transformed the way people interact with computers. Their generative capability is mainly attributed to their architecture, training data, and primary function. General-purpose LLMs are trained on vast and diverse textual datasets collected from the Internet, books, and articles. They can perform a wide variety of NLP tasks, such as answering questions, summarizing texts, having conversations, and translating languages. Popular examples of LLMs include ChatGPT, Gemini, and Claude. Another type of LLMs is domain-specific, which is specifically trained or fine-tuned on specialized datasets to perform a particular task or a range of tasks within a domain [101]. These types of LLMs can achieve higher accuracy within their field of application. Popular examples include BioGPT trained on biomedical data [31], PubMedGPT trained on medical research data [50], FinGPT trained on financial data [97], and LawGPT specifically trained on legal datasets [103].

The emergence of LLMs represents significant development in educational technology. Recently, researchers have explored the potential of LLMs to improve learning in computing education [53, 88]. Studies suggest that LLMs can support learning by helping students understand complex programming concepts, debug code, and generate explanations, examples, and feedback in natural language [38]. They also provide on-demand guidance to complement human instruction. However, scholars have raised concerns that reliance on AI-generated content may hinder genuine skill development and promote surface-level learning [18, 59, 76]. Overreliance on LLMs can undermine academic integrity, limit critical thinking, and reduce self-regulated problem-solving skills in students [38, 59].

Students' approaches to learning are usually shaped by their perceptions of the learning environment. Therefore, it is essential to understand how LLM agents influence these perceptions and behaviors of users. Properly designed LLM tools have the potential to align with student learning processes and promote deeper engagement. However, current research on this intersection remains fragmented. There is little synthesis of pedagogical roles, design strategies, or empirical evidence associated with the use of LLM tools in the student learning process. This gap calls for a systematic understanding of how LLM-based tools are conceptualized and implemented to support learning in computing education. A comprehensive review of the literature can provide insights into how such tools are designed to complement human teaching. This review can also help determine whether

LLM tools encourage the development of deeper learning approaches rather than reinforcing surface-level engagement in students.

Multiple theories of learning and cognitive processing have long informed educational practices. However, this literature review section establishes that the rapid expansion of LLM tools has introduced new complexities in the learning process. LLMs offer promising ways to address the traditional challenges in computing education, such as limited feedback, abstract content, and surface learning. The evidence for the potential of LLM tools in learning remains dispersed across individual studies, and there is no consolidated view of how these tools support the learning process. This systematic review of the literature responds to that need by synthesizing current research on LLM-based tools in computing education, focusing on how the tools are designed, what learning processes the tools engage in, and how the tools influence students' approaches to learning.

2.5 Related Work

Recent systematic literature reviews have explored the broad applications, benefits, challenges, and overall impact of large language models (LLMs) and generative AI (GenAI) tools in computing and education. For instance, Raihan et al. [76] conducted a comprehensive SLR analyzing 1,735 papers to examine the impact of LLMs in CS and computer engineering education. Their work mapped the distribution of research across educational levels (e.g., introductory programming), sub-disciplines of CS, predominant research methodologies, commonly used programming languages in LLM-related studies, and the specific LLMs (e.g., GPT-series, LLaMA) employed. The review highlights LLMs' potential of LLMs for personalized learning, code generation, debugging support, and curriculum enhancement, while also addressing challenges such as academic integrity and over-reliance.

Other reviews have targeted specific aspects of LLM integration. Pirzado et al. [72] systematically reviewed the behavior and limitations of LLMs as coding assistants for CS students. Their analysis addresses key questions, including current challenges hindering LLM effectiveness (e.g., inconsistencies in output), differences in model responses to code versus natural language prompts, the influence of output quality, comprehensibility, and correctness on students' learning experiences and outcomes, and the suitability of LLMs as debugging tools in computing education.

In the assessment domain, Pereira et al. [67] provided a systematic review of LLM-based contributions to evaluation processes in computer programming teaching. Their work investigated which LLM applications are used for assessing programming assignments, how these models deliver feedback to students, and the methodologies employed to evaluate the efficacy of such tools in improving learning outcomes, emphasizing the need for balanced human-AI oversight in grading.

Broader examinations of GenAI tools include Agbo et al. [4], who systematically reviewed the use of generative AI in computing education from K-12 to graduate levels. They synthesized the reported pros and cons of integration, including enhanced personalization and support for diverse tasks, alongside risks such as reduced critical thinking and integrity concerns. Gaitantzi et al. [30] focused on the role of artificial intelligence (including generative variants) in CS education through a systematic review. Their research questions cover methodologies for evaluating AI's impact of AI, integration into instructional design, roles and actions of GenAI in instruction, key benefits/challenges/limitations, transformations in AI-assisted programming approaches, and specific applications in areas such as database instruction.

Additionally, Reeping et al. [78] explored how students and faculty use LLMs (particularly ChatGPT) in engineering and computing education contexts for instruction and assessment, highlighting practical adoption patterns and implications for pedagogical practices. While these studies provide

valuable overviews of LLM and GenAI applications, benefits, challenges, assessment roles, instructional integration, and tool behaviors in CS/computing education, they primarily address broad impacts, research distributions, or specific use cases such as coding assistance and evaluation.

To the best of our knowledge, no systematic literature review has specifically synthesized (1) the pedagogical strategies, scaffolding techniques, or built-in guardrails embedded within LLM-based tools to support meaningful learning processes in computing education; (2) the targeted challenges associated with using these tools for deep learning tasks (beyond surface-level output generation); or (3) the multifaceted factors (e.g., tool affordances, learner characteristics, and contextual elements) that influence students' interactions and usage patterns with LLM-based tools for learning or academic tasks. This study addresses this gap by systematically reviewing these underexplored dimensions, focusing on how LLM tools can be intentionally designed and deployed to reinforce cognitive engagement and skill development.

3 Methodology

This study synthesizes the literature on LLM-based tools that support the learning process of students in computing education. This study adopts a Systematic Literature Review (SLR) approach guided by the framework proposed by Kitchenham and Charters [42]. The SLR method provides a rigorous and replicable process for identifying, evaluating, and synthesizing existing evidence to answer pre-defined research questions. The framework proposed by Kitchenham and Charters [42] is particularly suited to computing-related research because the process emphasizes transparency, methodological rigor, and the reduction of bias. This review proceeded through three main phases: planning, conducting, and reporting the results. Each phase is described in detail below.

3.1 Planning the Review

3.1.1 Review Objectives and Research Questions. There are similar literature reviews on LLM in computing education, such as that of Raihan et al. [76]. However, to our knowledge, no study has explored the use of LLM to enhance learning for students in computing education, the pedagogical strategies used, and the challenges of using these LLM-based tools. The purpose of this study was to systematically review how LLM-based tools have been used to support learning processes in computing education. The research was guided by the following questions.

RQ1 What LLM-based tools have been developed or employed to support learning in computing education?

RQ2 How do LLM-based tools support learning processes in computing education?

RQ3 What factors influence how students use LLM-based tools in computing education?

RQ4 What are the challenges and limitations of integrating LLM-Based Tools to Support Learning in computing education?

These research questions were designed to address the key gaps identified in the existing literature. Prior studies and reviews have examined how LLMs are used in education more broadly, but few have systematically explored their role in learning processes within computing education [5, 67, 76]. The research questions seek to capture the existing LLM tools (RQ1), the strategies embedded in these tools to influence the learning process (RQ2), the factors that affect how students learn with these tools (RQ3), and the challenges of integrating these tools into computing education (RQ4). This study highlights the need for an integrated pedagogical and technological understanding of LLM-supported learning environments.

3.1.2 Review Protocol. A detailed review protocol that followed the recommendations of Kitchenham and Charters [42] was developed to ensure consistency and replicability. The protocol defined the scope of the review, databases to be searched, formulation of search strings, inclusion and

exclusion criteria, and data extraction procedures. The protocol was established before we began the data collection process and served as a guiding document throughout the review process.

3.1.3 Source Selection. The authors initiated the process by snowballing and searching for previous systematic literature reviews on LLM in computing education to become familiar with the terminology and identify the relevant databases. After several rounds of iterations, the authors selected the ACM Digital Library, IEEE Xplore, and Scopus as the primary databases for this systematic inquiry to capture as many contributions as possible. ACM Digital and IEEE Xplore were selected because they cover the literature on computing education, engineering, and educational technology [1, 37]. Furthermore, ACM DL and IEEE Xplore host key conference proceedings on computing education, such as ACM SIGCSE, ACM SIGITE, and ICER proceedings. These conferences are highly relevant to the research questions posed in this study. Scopus was included for its multidisciplinary scope, which includes studies on educational technology and computing pedagogy in a wider range of journals [25].

3.1.4 Search Query. The search query was developed through an iterative process that involved collecting search queries from existing relevant studies. The systematic literature review by Raihan et al. [76] was used as a benchmark, and we deconstructed the research topic to find relevant keywords. We then performed pilot searches to refine the search query and optimize conceptual relevance. Alternative terms such as "learning," "education," and "teaching" were tested. The preliminary test showed that queries using the term "learning" retrieved over 6000 results in the ACM DL. However, a manual inspection of some sample articles showed that most of the articles were not related to human learning but were primarily related to machine learning, model learning and AI algorithms. In contrast, the term "learning process" yielded results that were more aligned with human learning and the research questions. Furthermore, the term "learning process" is directly aligned with the scope of this study and its research questions. The term is also aligned with process-oriented educational frameworks [43, 70].

At the end of this process, a comprehensive search query was developed that combined three key concept areas: LLM-related terms, computing education terms, and human learning process-related terms. For LLM-related terms, we included "large language model" or "LLM" to capture various references to large language models. The computing education component included terms such as "computer science," "information technology," "computing," and "software engineering." To identify research that focuses on the human learning process in LLM, we used the term "learning process." All search components were connected to Boolean operators ("OR" and "AND"). The final search string was as follows:

("large language model" OR "LLM") AND ("computing education" OR "computer science" OR "information technology" OR "computing" OR "software engineering" OR "computer engineering") AND ("learning process")

3.1.5 Inclusion and Exclusion Criteria. For our research, we limited the time frame of the results to publications published between 2022 and 2025. This period defined the time LLMs became popular up to the time when this research was conducted. The inclusion and exclusion criteria presented in Table 1 were refined through several iterations. A retest was conducted after one week to assess the consistency of these criteria. The criteria were finalized after the retest.

Table 1. Inclusion and Exclusion Criteria

Include	Exclude
Peer-reviewed articles	Posters, magazines, newsletters or reports
Studies published between 2020 to 2025	Studies not focused on computing education
Investigates the use of LLM agents to facilitate learning in computing education	Papers not focused on using LLM agents to facilitate learning process
Written in English	Duplicate studies
Open access articles	Studies where full text is not available

3.2 Conducting the Review

The search was conducted on June 7, 2025. For our search, we limited the time frame of the results to publications published between 2022 and 2025 because this was the period of widespread adoption of large language models. We excluded posters, magazines, and reports, as stated in Table 1 of the inclusion and exclusion criteria. The search yielded 701 results, as shown in Table 2.

Table 2. Search results in each database

Database	Query	Result
ACM Digital Library	("large language model" OR "LLM") AND ("computing education" OR "computer science" OR "information technology" OR "computing" OR "software engineering" OR "computer engineering") AND ("learning process")	625
IEEE Xplore	("large language model" OR "LLM") AND ("computing education" OR "computer science" OR "information technology" OR "computing" OR "software engineering" OR "computer engineering") AND ("learning process")	55
Scopus	("large language model" OR "LLM") AND ("computing education" OR "computer science" OR "information technology" OR "computing" OR "software engineering" OR "computer engineering") AND ("learning process")	22
Total		701

3.2.1 *Screening Process.* The article selection process followed a two-stage screening. After removing 17 duplicates from the initial retrieval, 695 unique records remained in the review. In phase 1, the title, abstract, and keywords of all records were independently assessed against the predefined inclusion and exclusion criteria. This phase resulted in a set of records to 50 articles. In Phase 2, the same inclusion/exclusion criteria were applied to the remaining 50 articles. This phase reduced the number of articles to 50.

3.3 Quality Assessment

The quality of the 46 articles was assessed. The quality assessment criterion is the question "to what degree does the article focus on student learning with LLM in computing education?" The quality assessment process showed that 87.5% of the final 46 articles were mostly focused on student learning with LLM, while the remaining 12.5% were partially focused on student learning with LLM. The 46 articles were included as primary studies. Figure 1 presents a flowchart of the identification, screening, and selection processes.

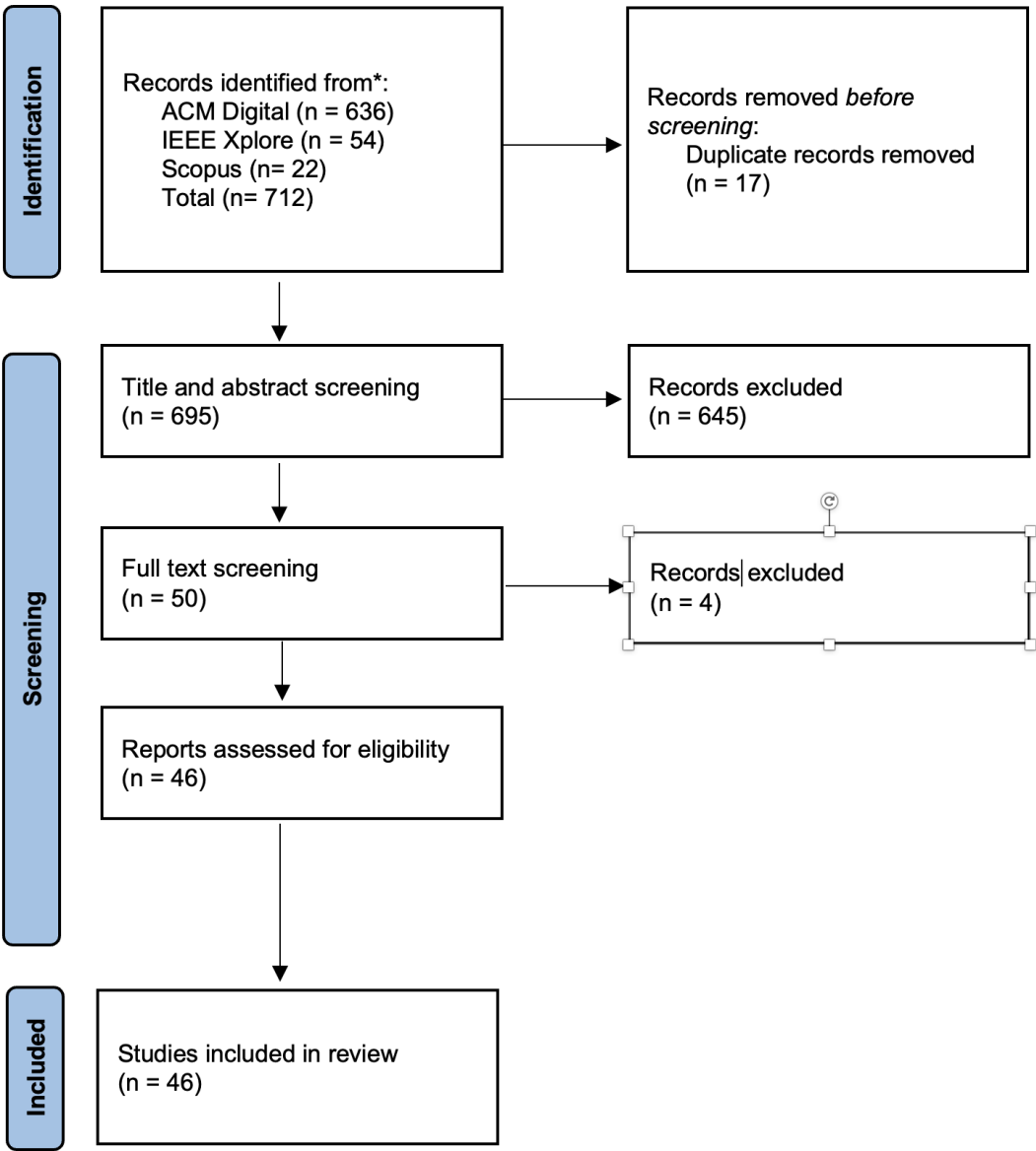


Fig. 1. Overview of processes involved in articles selection

3.4 Data Analysis

A reflexive thematic analysis was performed following the six-phase framework outlined by Braun and Clarke [12]. The phases included data familiarization, initial coding, theme searching, theme reviewing, theme defining and naming, and report production.

Phase 1: Familiarization. The first four authors were divided into two groups of two authors each. During this phase, each author read all 46 primary studies multiple times to gain a comprehensive understanding of their content. The authors recorded the reflexive notes, impressions, and preliminary analytical ideas. This reflexive engagement marked the beginning of the thematic

analysis. Relevant qualitative information from each study was extracted using a structured data form developed in Microsoft Excel software. The form captured bibliographic details (author, year, venue), study context (educational level, computing domain), system type and purpose, pedagogical strategies, student interactions, and reported challenges/outcomes.

Phase 2: Initial Coding. The initial codes were inductively generated to capture meaningful features of how LLM-based systems were applied, implemented, and evaluated in computing education. These codes were documented in shared analytic memos and Excel sheets to facilitate reflexive discussions. The authors collaboratively interpreted, compared, and refined their codes through dialogue, treating subjectivity as an analytic resource rather than a bias. Throughout the analysis, the team engaged in reflexive dialogue to acknowledge how their disciplinary backgrounds and assumptions shaped the interpretation.

Phase 3: Searching for Themes. The authors collaboratively reviewed the final codes. The codes were organized into conceptually related groups, from which subthemes were developed.

Phases 4 and 5: Reviewing and Naming Themes. The sub-themes were grouped into broader themes. Each broader theme was defined and named to capture its central meaning in relation to the research question. The themes were reviewed and refined to ensure internal coherence and interpretative clarity.

Phase 6: Generating of the report The final phase involved the construction of a thematic narrative that synthesized the findings into a conceptual framework that will be discussed in Section 5.4. This narrative connects the identified themes to the learning outcomes reported in the literature and highlights how challenges, such as dependency risks and ethical concerns, moderate the effectiveness of LLM-based systems in computing education. To enhance analytic transparency and trustworthiness, the authors maintained reflexive memos and audit trails documenting decisions, assumptions, and interpretive shifts throughout the analysis.

3.5 Overview of Included Studies

A total of 46 studies met the inclusion criteria and were included in the final synthesis of the results. The list of all primary studies used in this study is presented in Table 3 in Appendix A. These studies will be published between 2023 and 2025. Most studies published in conference proceedings and journals have focused on human-computer interaction, computing education, and educational technology. Some of the venues include the CHI Conference on Human Factors in Computing Systems, Innovation and Technology in Computer Science Education, International Conference on Intelligent User Interfaces, In Computing Education Practice, International Conference on e-Society, e-Learning, and e-Technologies (ICSLT 2023), IEEE Frontiers in Education, and Education Technology and Computers.

The distribution of papers across different course categories is shown in Figure 2. The dataset is dominated by programming. 32 of the 46 articles (69.6%) explored LLM-based tools to support student learning processes in programming contexts. The remaining papers are distributed across five major categories: Software Engineering (4 papers or 8.7%), Database Systems (4 papers), visualization (1 paper or 2.2%), and robotics (1 paper). Three articles (6.5%) were classified as general because they represented studies in which computing students' learning processes were examined without reference to a particular course context. A word cloud of the keywords identified in the primary studies was also created. Figure 3 presents a word cloud of the keywords from the primary studies.

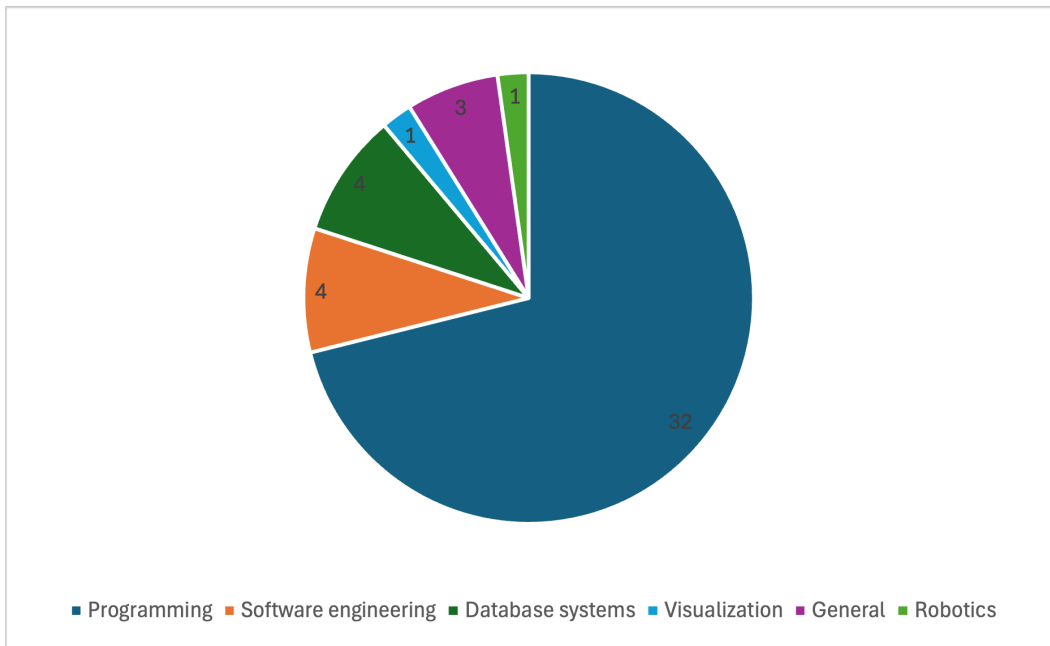


Fig. 2. Distribution of Papers by Computing Courses

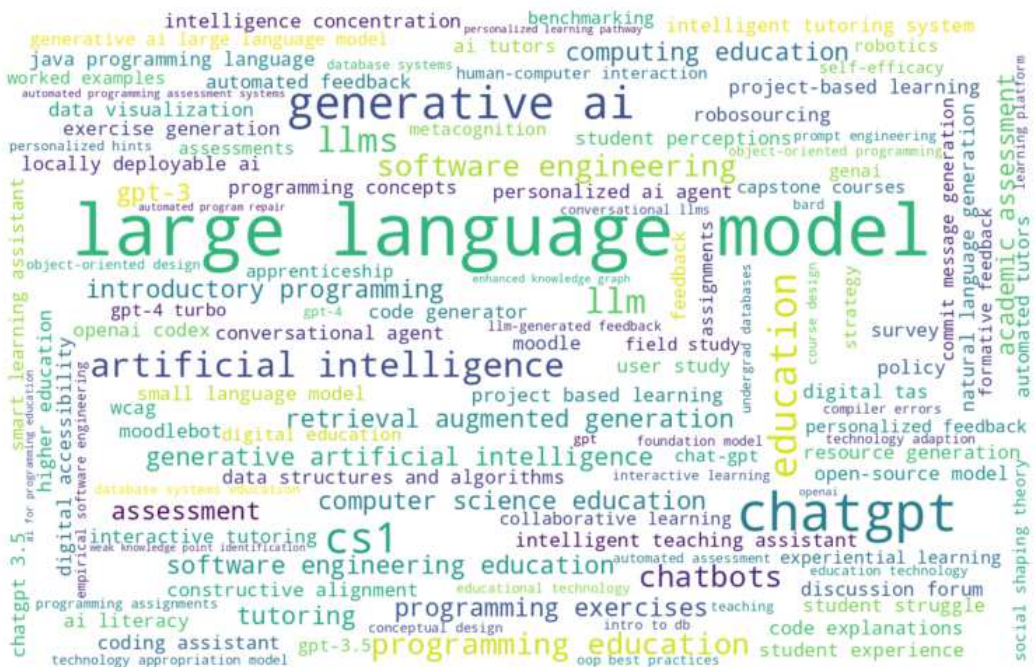


Fig. 3. Keyword Visualization Cloud

4 Results

This section presents the findings derived from the analysis of 46 primary research studies. The findings are grouped based on the research questions introduced in Section 1.

4.1 LLM-Based Tools for Learning in Computing Education

Of the 46 final studies, 24 used existing commercial, general-purpose LLM-based systems, and the remaining 22 developed custom LLM-based tools to support learning in computing education. The commercial tools and the associated studies include ChatGPT [6, 7, 9, 14, 19, 26, 32, 35, 41, 52, 57, 64, 69, 75, 77, 85], GitHub Copilot [14, 35, 85], Bard, Gemini and others [35, 63, 82]. The 22 custom made LLM-based learning tools include Iris [10], AI Defenders [58], VizGroup [93], CodeTutor [53], RAGMan [54], ContextTutor [60], RAG-AI [36], CodeHelp [21], Programmer's Assistant [84], MoodleBot [61], Progmate [49], AI-Tutor [29], PyDex [101], ClueBot [73], LetsCommit [62], DB Tutor [74], WorkedGen [39], Feedback Generator [9], Feedback System [79], Exercise Generator [86], CodeLLM [40], GPT Hint System [66], and Adaptive Prompt System [3].

The 22 custom tools were analyzed to extract design patterns that have implications for how LLMs support learning in computing education. These patterns are discussed in the following sections.

Temporal concentration in active problem-solving. The overwhelming majority of tools (86% or 19 out of 22) focused on the active learning phase, such as when students were engaged in problem-solving. Only three tools (14%) provided pre-instruction support, and six tools (27%) offered post-submission feedback or metacognitive support. This active phase concentration suggests that developers primarily envision LLMs as 'just in time' support mechanisms rather than comprehensive learning companions that can be useful throughout the full instructional cycle.

Learner initiated interaction as default. Nearly three-quarters of the tools (73% or 16/22) required learners to initiate requests for help, and only six tools (27%) proactively offered system-initiated support. Although interactions initiated by learners help to preserve student agency, research on help-seeking behavior suggests that struggling students, especially those from underrepresented groups or with lower self-efficacy, may be less likely to request assistance when needed [100]. This design pattern may unintentionally create equity issues in accessing support.

Prevalence of direct instruction over guided discovery. Direct instruction emerged as the most common pedagogical approach (59% or 13/22), and the tools primarily explained concepts or provided answers to student queries. In contrast, only eight tools (36%) explicitly incorporated solution-withholding mechanisms to maintain productive struggle [98] and only one tool (5%) employed Socratic questioning techniques [15, 23]. This pattern reveals a conflict between the convenience of direct answers and learning science principles that emphasize the importance of maintaining cognitive engagement and productive struggle [45].

Solution-withholding as minority practice. Despite substantial evidence from learning sciences on the importance of maintaining productive struggle and avoiding premature solution-giving [98], only 36% of the tools explicitly designed mechanisms to withhold complete solutions. Tools such as AITutor [29], Iris [10], CodeHelp [21], and RAGMAN [54] include explicit prompting strategies (e.g., "Do not provide code") or verification layers to prevent the disclosure of the solution. However, most tools prioritized immediate comprehensive answers over scaffolded guidance. However, this design choice may undermine the development of independent problem-solving skills.

Multifunctionality Thirteen tools [3, 10, 22, 29, 36, 39, 54, 58, 60, 66, 73, 74, 84] performed multiple functions simultaneously. For example, Iris [10] provides hints, answers questions, and evaluates students' work. ContextTutor [60] evaluates the work, provides feedback, and tracks

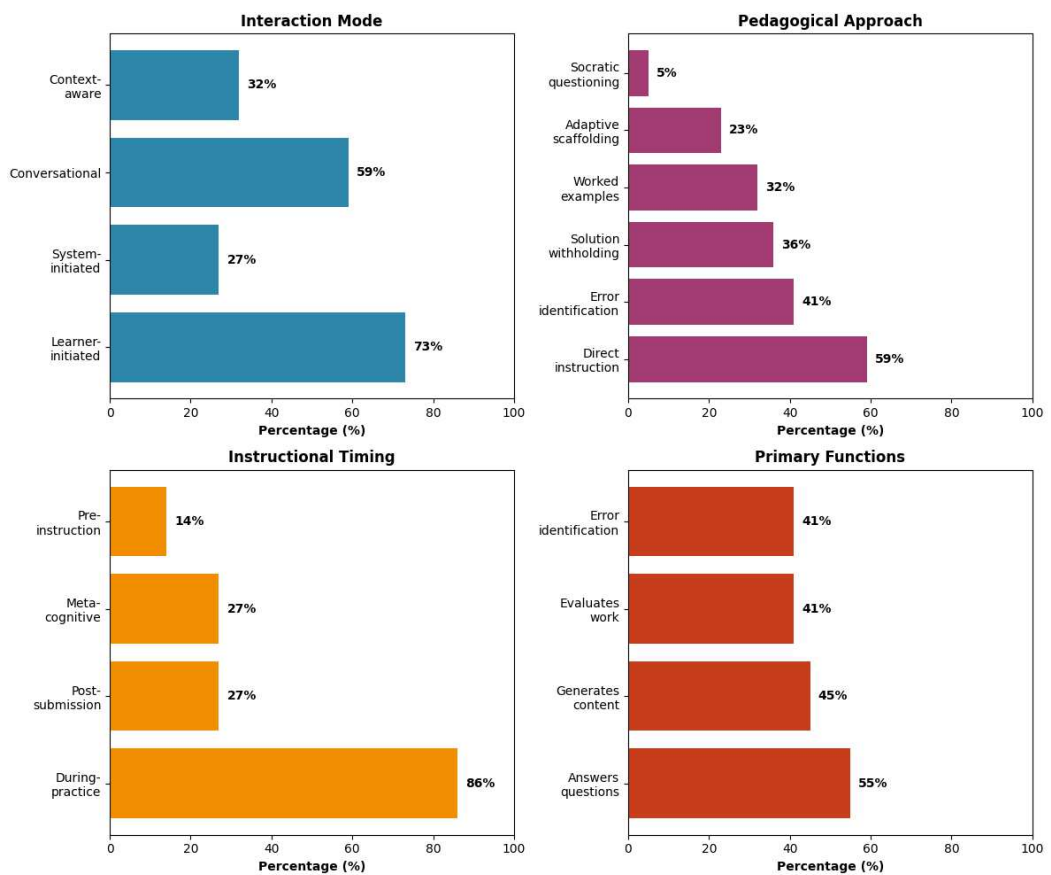


Fig. 4. Design Patterns Across 22 Custom LLM Tools

progress over time. This multi-functionality demonstrates the complexity of LLM-based educational tools and explains why simple functional taxonomies may be inadequate for classifying these tools.

Figure 4 summarizes the design patterns of the custom LLM tools. These patterns reveal both opportunities and challenges in the design of current LLM-based tools for computing education. These tools effectively provide accessible on-demand support during active problem-solving activities. Unfortunately, the tools’ predominant reliance on direct instruction and learner-initiated interaction may limit their effectiveness in promoting deep learning and equitable access to support.

4.2 How LLM-Based Tools Support Learning Processes in Computing Education

The analysis of the 46 studies revealed five themes as the primary mechanisms through which LLM-based tools support learning. These themes include *adaptive scaffolding*, *interactive pedagogical structures*, *formative feedback generation*, *personalization*, and *structured learning activities*.

4.2.1 *Adaptive Scaffolding*. Adaptive scaffolding has emerged as the most common pedagogical support mechanism and encompasses four main components: *calibrated solution withholding*, *guardrailed solution refusal*, *process-oriented guidance*, and *explanation adaptation*. These strategies

are designed to support immediate task completion, long-term skill development, and enhance student engagement with challenging content. 77% of the tools provide at least one form of adaptive scaffolding.

Calibrated solution withholding focuses on providing minimal help to learners so that they can construct the solution on their own. These tools do not provide complete solutions to tasks. Instead, the tools deliberately offered graduated levels of support, such as minimal hints or counter questions to learners. These supports are designed to maintain productive struggle. Guardrailed solution refusal prevents direct solutions even when requested by the user. This approach differs from calibrated withholding because the refusal is automatic and absolute rather than contextually calibrated. Process-oriented guidance emphasizes the provision of support during task execution to guide learners through execution steps without providing the full solution. This approach guide students through problem-solving processes by providing structural support that guided students through problem-solving processes. Examples of such support include breaking tasks into smaller steps with explanations. Explanation adaptation prioritizes the adaptive delivery of responses in the form of explanations that match the learner's needs. It uses different mechanisms, such as *conceptual simplification* to summarize concepts into simple points, *concept decomposition* to break down unfamiliar concepts into understandable parts, and *contrastive explanation* to explain why an incorrect approach fails or why the correct one works. Six tools [10, 21, 29, 54, 60, 66] provided support for calibrated solution withholding, and three tools supported guardrailed solutions. Three tools also provided process-oriented guidance, while eight tools adopted explanation adaptation mechanisms.

Adaptive scaffolding provides measurable cognitive benefits that extend beyond immediate task completion. Pankiewicz et al. [66] noted that students who had previously received calibrated support demonstrated superior performance (when scaffolding was removed) to those who had not. Furthermore, adaptive scaffolding has been shown to support the iterative refinement of student thinking [77, 79] and produce significant improvements in students' emotional states during learning [39, 66]. These findings have important implications for equity in computing education, as frustration and confusion are known predictors of attrition, particularly among students from underrepresented groups. Adaptive scaffolding supports engagement during challenging moments, which can help retain students who might want to disengage from computing.

4.2.2 Formative Feedback Generation for Conceptual Understanding Development. Six studies discussed formative feedback through three primary mechanisms: *informative feedback delivery* [29, 52, 79], *iterative error detection* [9, 13, 79], and *alternative solution suggestions* [75]. These feedback mechanisms supported the development of a deeper conceptual understanding.

Informative feedback delivery provides immediate corrective feedback during active problem-solving. *Iterative error detection* mirrors expert tutoring to progressively identify errors instead of overwhelming students with all errors at once [9, 13, 79]. Foundational errors are revealed first, and further errors are revealed as students address initial errors and refine their work. Students in programming classes valued iterative error detection for debugging purposes [13]. *Alternate solution suggestions* provides suggestions on other solutions that may be explored to address a problem. These feedback mechanisms produce learning outcomes that transcend immediate error correction [79]. Students described how LLM-based feedback "serves as a valuable learning tool that helps students understand complex concepts, generate ideas, and receive feedback" [91].

4.2.3 Personalized Content for Individualized and Accessible Learning Support. Six studies [10, 13, 41, 52, 61, 69, 75, 77] highlighted content generation mechanisms that provide students with on-demand access to personalized learning materials, practice problems, and explanatory resources. These content generation mechanisms can address the perennial challenge of resource scarcity in

large-scale computing education settings. Personalization mechanisms produce more effective and efficient learning experiences than one-size-fits-all instructional approaches because they tailor LLM content or responses to individual students' needs, knowledge levels, and learning contexts. The primary mechanisms explored in the six studies are *context-aware tailoring* [9, 10, 52, 75, 79, 91], *quick information access* [13, 41, 61, 69, 77], *interest-based customization* [26], and *personalized content generation*.

Context-aware tailoring matches response complexity to student readiness, by providing "clear, precise answers that are tailored to individual student queries and challenges, similar to a tutor who understands precisely where a student is struggling" [52]. *Quick information access* provides rapid access to starting points and this was particularly valuable for students that struggle to begin complex tasks. *Interest-based customization* integrates student personal interests and hobbies into learning materials. For example, tools might generate analogies that are relevant to students' interests. Research has shown that interest-tailored analogies are both more diverse and more memorable than generic examples, and personal relevance enhances both engagement and retention [26]. *Personalized support* involves tailoring pedagogical support such as content difficulty and explanatory approach, to each student's unique conceptual gaps and readiness. Students develop a deeper understanding when the tools meet them at their current knowledge level. Personalization mechanisms may be particularly valuable for heterogeneous classrooms where students start class with varying levels of prior knowledge.

4.2.4 Interactive Pedagogical Structures for Critical Thinking and Reasoning Development. Studies have integrated multiple interactive features to transform LLM tools from static answer providers into dynamic learning partners that support reasoning development. Integrating interactive features include *pedagogically structured interactions* [39, 54, 60], *dialogic clarification support* [39, 61], and *interactive reasoning support* [39, 41, 77, 91].

Pedagogically structured interactions such as "theory-based feedback structure", "explain-then-exemplify sequencing", and "worked example" follows established pedagogical principles about how learners construct understanding [54, 60]. The pedagogical principle in the theory-based feedback structure uses Hattie and Timperley's three levels of feedback [60]. Explain-then-exemplify sequencing involves the tool first explaining the concept and then providing relevant examples [54]. Jury et al. [39] integrated the "worked example" pedagogical principle into LLM tools to support learners' understanding. Learners also get to view concepts before putting them into practice.

Dialogic clarification support explores the conversational nature of LLMs to enable follow-up questioning that helped students to iteratively refine their understanding [39, 61]. *Interactive reasoning support* uses well designed structures to help learners generate and explore ideas through AI as a thought partner. Some tools also provide checkpoints for students to verify their reasoning [39], while others help learners generate and explore new ideas and perspectives [41, 77, 91]. These tools can serve as thought partners that challenge and extend student thinking.

4.2.5 Structured Learning Activities. Three studies designed complete learning activities that positioned LLM interactions within broader pedagogical frameworks. These activities include practice resource generation [39], prompt problems, and alternative accessibility pathways [26].

Practice resource generation automates the creation of diverse and high-quality practice exercises [39, 79]. *Prompt problems* explicitly teaches prompt engineering as a learning objective because efficient use of LLM requires specific skills such as writing effective prompts. This approach prioritizes computational thinking and critical evaluation of AI-generated outputs rather than direct learning. The focus is on how to articulate problems in natural language to bridge the gap between problem solving and programming. For example, learners may be asked to explain the high-level purpose of a provided code fragment by writing a prompt that will generate code



Fig. 5. Challenges in LLM-Based Learning Support for Computing Education

equivalent to the original. This requirement forces students to understand the code deeply before attempting to describe it. Automated feedback on their natural language descriptions also provided immediate guidance on whether their understanding was accurate.

Alternative accessibility pathways focus on equity by providing alternative complementary skills to learners such as the use of natural language to explain programming concepts for students who struggled with syntax. This can potentially expand access for students who possess a strong conceptual understanding but struggle with syntactic precision. The complementary skill development approach recognizes that programming expertise encompasses multiple dimensions, and LLM tools could support the development of some dimensions (problem specification, algorithmic thinking, code comprehension) even for students who face challenges in others (syntax mastery, memorization of language features).

4.3 Challenges and Limitations of Integrating LLM-Based Tools to Support Learning

Our analysis revealed eight themes that characterize the challenges of integrating LLM tools to support the learning process. These challenges include *system quality and reliability issues*, *lack of personalization*, *dependency and decline of self-efficacy*, *impact on learning and skill development*, *critical thinking and metacognitive deficits*, *interaction difficulties*, *academic integrity and learning bypass*, *reduced engagement with learning resources*, and *equity and access disparities*. Figure 5 presents the themes and subthemes of the challenges found in the literature. These challenges operate across technical, pedagogical, and behavioral dimensions to reveal important tensions in LLM integration in computing education.

4.3.1 *System Quality and Reliability Issues*. The quality and reliability of LLM outputs are among the most frequently reported challenges. Six studies reported these challenges [9, 14, 29, 32, 39, 52–54, 77]. The challenges related to system issues are discussed below.

- **Declining trust and preference over time.** Longitudinal studies have revealed shifts in student attitudes toward LLM-based tools after extended use due to reduced comprehension benefits, declining tool accuracy, and unmet expectations [53, 77].
- **Incorrect or misleading output.** Multiple studies have documented instances where LLM tools generated incorrect solutions or explanations, creating potential obstacles to learning

rather than support [39]. Furthermore, some error explanations could be misleading, even when the corrections themselves are accurate, and students may also receive contradictory responses [9, 52, 54].

- **Poor feedback quality.** LLM-generated feedback often exhibits characteristics that impede learning, even if the feedback is technically correct [9]. The feedback may be verbose, redundant, inconsistent, or generic [29, 32].
- **Complexity mismatch.** LLM responses can introduce concepts beyond students' current knowledge. This scenario often increases confusion instead of providing clarity [14].

These challenges revealed a fundamental challenge where those who need scaffolded support the most are least able to obtain it, such as students who lack foundational knowledge but cannot constrain LLM outputs to match their expertise level.

4.3.2 Lack of Personalization. Personalization is one of the major support mechanisms reported in custom-developed tools. However, studies have revealed persistent challenges in adapting to individual learner needs, particularly regarding matching the learner's expertise level and domain specialization. A major factor is expert-level mismatch, where the tools did not consider the expertise level or assumed an incorrect expertise level of the learners. This mismatch manifested both as overly complex responses for beginners and insufficiently sophisticated guidance for advanced learners to follow.

4.3.3 Dependency and Self-Efficacy Decline. Arguably, the most concerning challenges are centered on psychological effects, such as decreased self-efficacy and dependency on the tool. These challenges can limit the independent learning capabilities of students. Five studies [10, 29, 64, 73, 77] reported diminished self-efficacy and lack of self-confidence when learners attempted to solve problems without the tools. Padiyath et al. [64] reported statements from learners such as "I would say that if you give me like a coding assessment right now without access to ChatGPT, I would freak out". This statement captures both the trigger of dependency (copy-paste behavior) and its affective consequences (anxiety about unaided performance).

4.3.4 Impact on Learning and Skill Development. A particularly concerning cluster of challenges involved documented impacts on actual skill development and learning outcomes was identified. These impacts include basic skill atrophy, acquisition of poor coding practices, and the inability to synthesize knowledge across contexts. Basic skill atrophy refers to the failure to develop basic competencies, such as students who successfully completed an introductory programming class but could not independently write a print statement [26]. Poor coding practices occur when learners internalize problematic practices from LLM-generated code [69]. LLM tools also struggle to help students connect concepts across exercises, even though this is a critical skill for developing an integrated understanding. These challenges show that LLMs are best used for learning if the learners possess initial prior knowledge that can be connected to. Unfortunately, this prerequisite creates a problematic dynamic in which students who most need support (i.e., those lacking foundational knowledge) are least able to benefit from LLM assistance, whereas those with strong foundations can use LLMs productively to extend their knowledge.

4.3.5 Critical Thinking and Metacognitive Laziness. The studies documented challenges in students' capacity to evaluate LLM outputs and engage metacognitively with AI-assisted learning. This challenge often manifests when learners struggle to assess the quality and appropriateness of LLM responses or find it burdensome to verify LLM outputs.

4.3.6 Interaction Difficulties. There are technical and practical challenges in interacting effectively with LLM tools. These interaction challenges have emerged as substantial barriers to productive use,

including difficulties in formulating necessary prompts, iterative refinement burdens, and ineffective help-seeking patterns. Many learners find it difficult to create effective prompts. Furthermore, well-crafted prompts often require multiple rounds of interaction to obtain satisfactory outputs. Hence, learners may struggle to interact effectively with LLMs to receive meaningful feedback.

4.3.7 Academic Integrity and Learning Bypass. A cluster of challenges centered on students' use of LLM tools to bypass learning processes, violate academic integrity policies, and submit work without genuine engagement. These challenges were explicitly linked to the motivational factors. Three studies [26, 44, 64] reported evidence of academic integrity violations, task-completion orientations that prioritized grades over learning, and confusion about the appropriate use of LLM.

4.3.8 Reduced Engagement with Learning Resources. Four Studies [26, 29, 41, 63] documented shifts in student help-seeking behaviors as easy access to LLMs was associated with decreased engagement with traditional learning resources, including instructors and peers. The displacement of human interaction is particularly concerning, given that instructor and peer interactions do not address technical questions only. Instructors and peer interactions also provide affective support, community building, and the development of professional communication skills dimensions that LLM interactions cannot replicate.

4.3.9 Equity and Access Disparities. The final challenge extracted from the studies shows that LLM tools might worsen educational inequities through economic disparity and differential capacity for effective use. For example, paid versions of LLM tools might have more advanced features that are not accessible to learners who cannot afford the paid version. This creates the potential for economic advantage to translate into a learning advantage. Furthermore, differences in LLM usage skills creates another dimension of inequity. LLM tools may benefit students who already possess strong metacognitive skills and domain knowledge, while providing limited value to those who need support.

4.4 Factors Influencing How Students Use LLM-Based Tools

Our analysis of 46 studies identified factors that shaped how students engaged with LLM-based educational support tools within the context of computing education. Seven primary themes were identified in the study. The themes are: *learner characteristics*, *AI literacy and interaction skills*, *tool design and technical characteristics*, *pedagogical integration and structure*, *assessment design and academic policies*, *social and institutional context*, and *temporal and experience factors*. Figure 6 provides an overview of the themes and subthemes.

4.4.1 Learner Characteristics. Students' personal characteristics have emerged as a foundational influence on their engagement with LLM-based tools. These personal characteristics include prior knowledge, self-efficacy, and motivation. LLM tools may function more effectively as support for extending existing knowledge rather than as primary instruction for novices that lacks foundational understanding [14]. Students' beliefs about their own capabilities can influence both the way they use LLM and the consequences of that use [64]. Intrinsic learning motivation may promote LLM engagement that is more pedagogically productive [64].

4.4.2 AI Literacy and Interaction Skills. The effectiveness of LLM-based learning support was mediated by students' capabilities to craft appropriate prompts and understand the characteristics of the AI system. Learners must be proficient in providing precise prompts to LLM tools to generate useful content. Iterative refinement of the initial prompt with follow-up questions may also be required when the initial responses are inadequate. This iterative approach requires metacognitive awareness of the inadequacy of the initial response and the strategic capacity to reformulate

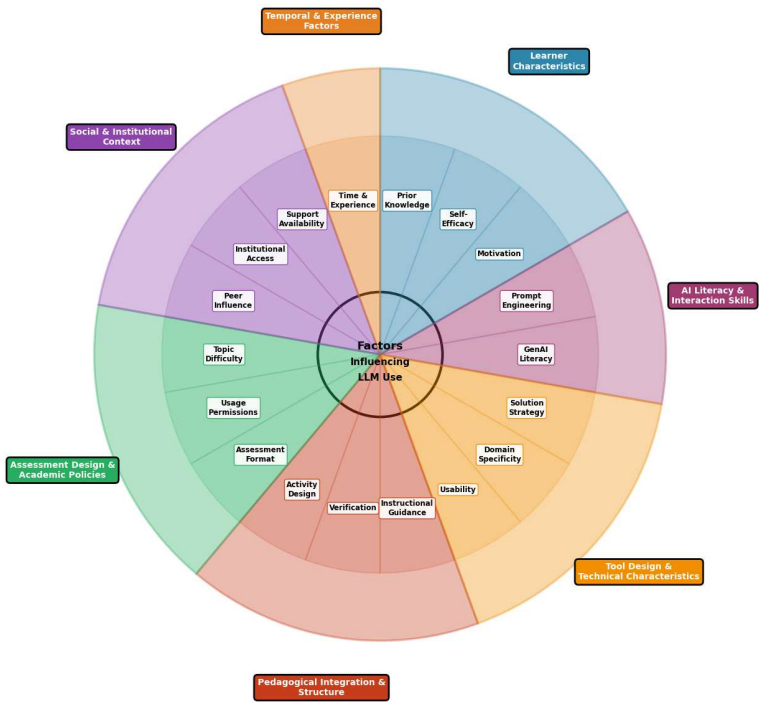


Fig. 6. Factors Influencing Student Use of LLM-Based Educational Tools

queries. Furthermore, a broader understanding of AI’s capabilities and limitations can influence its productive use. AI literacy encompasses awareness of when AI support is appropriate, recognition of output limitations, and ethical considerations regarding academic integrity and professional development.

4.4.3 Tool Design and Technical Characteristics. The design decisions embedded in LLM-based tools can substantially influence the learning interactions. The tool’s strategy for providing solutions, the domain coverage of the training data, and the usability of the interface all influence learning interactions. For example, tools that provide complete solutions will maximize immediate task success but potentially minimize the learning struggle necessary for skill development.

4.4.4 Pedagogical Integration and Structure. One of the most consequential factor that influence productive use of LLM tools is the way instructors integrate the tools into learning activities. Tools with structured pedagogical frameworks usually have more productive use than those that allows unsupervised access. Four studies [19, 26, 32, 57] reported at least one form of pedagogical structure. Many tools also embed verification mechanisms that required students to validate LLM outputs rather than accept them uncritically. The design of learning activities will influence whether the use of LLM supports or undermines learning. Learning activities that require integration of multiple prior exercises or synthesis across various contexts are more resistant to LLM offloading.

4.4.5 Assessment Design and Academic Policies. Institutional policies around the use of LLMs and the formats of assessments can substantially shape how student engage with LLM tools. The availability of LLM support during assessments influenced both usage patterns and performance

distributions. Explicit policies around when and how LLMs could be used also influenced student behaviors. The difficulty level of assessment tasks also influence when students sought the support of LLMs because students often turned to AI when they encounter a challenging material. This pattern indicates that difficulty may be a trigger for help-seeking. However, it is not clear whether this represents a productive struggle supplemented by support or avoidance of the necessary challenge for learning [26, 40, 63].

4.4.6 Social and Institutional. Social dynamics and institutional resources shaped the context within which students made decisions about LLM use. For example, peer behavior can influence students' decisions to adopt these tools, even if they initially have concerns about the tool's negative impacts on their learning. The availability and accessibility of traditional support resources influenced whether students turned to LLM tools. Finally, institutional policies on LLM access created differential opportunities for students [26, 41, 64].

4.4.7 Temporal and Experience Factors. Students' experiences with LLM tools evolved over time and extended use often reveals limitations not apparent in initial interactions. Initial enthusiasm can turn into skepticism as students encounter limitations, inconsistencies, or instances where LLM support proves inadequate. The factors identified in this analysis are inter related and did not operate in isolation. Students with strong prior knowledge, high self efficacy and learning-oriented motivation appeared to be better positioned to use LLM productively. This advantage is amplified when the LLM tools employed sophisticated pedagogic designs, and instructors provided structured guidance with verification requirements. In contrast, students with limited foundations, low confidence, and task-completion orientations were more vulnerable to problematic dependency patterns. This vulnerability is more apparent with general-purpose LLMs that do not have pedagogical constraints or when there is limited instructor guidance.

The results reported in this section suggests that effective integration of LLM in computing education requires concurrent attention to multiple viewpoints such as selecting or designing tools with appropriate pedagogical attributes, structuring learning activities to promote productive use, implementing assessment practices that verify independent competence, and potentially differentiating support based on student characteristics. The substantial influence of pedagogical integration factors are more susceptible to instructor control than student characteristics or tool capabilities. This suggests that thoughtful instructional design may be the most powerful lever to optimize LLM-based learning support.

5 Discussion

The synthesis of the findings of the four research questions reveals that LLM-based tools are able support learning processes across the cognitive, metacognitive, and affective dimensions.

5.1 LLMs as Adaptive Scaffolds of Cognitive and Metacognitive Processes

LLMs can consistently provide personalized and context-sensitive guidance that mirrors human tutoring processes. These systems deliver stepwise explanations, iterative feedback, and conceptual clarification that help learners navigate their zone of proximal development (ZPD) [87]. This adaptive scaffolding operates through fine-grained feedback loops, where learners externalize their reasoning via prompts and receive tailored elaborations that reinforce conceptual understanding and procedural fluency.

Importantly, this process extends beyond cognitive support to metacognitive engagement. Students can be prompted to reflect on their own problem solving strategies, assess the reliability of AI outputs, and calibrate trust processes that align with models of self-regulated learning (SRL) [104]. In this sense, LLMs can serve as metacognitive mirrors that help students monitor, plan, and

evaluate their learning processes. The theoretical implication is that LLM integration transforms feedback from a unidirectional event into a dialogic, iterative process of co-regulated cognition where agency is dynamically distributed between learner and system.

5.2 LLMs as Co-Regulators of Learning Agency and Motivation

The findings of research questions 2, 3, and 4 reveal that LLMs exert a strong motivational and behavioral influence on the way students approach learning tasks. Student engagement is shaped by their perceived autonomy, career orientation, and the social normalization of LLM use. The identified motivational dynamics align closely with self-determination theory and this suggests that LLM tools can enhance or undermine intrinsic motivation depending on how they balance support with autonomy [20].

LLMs tends to foster competence and engagement when learners view LLMs as a collaborative partner that is responsive, transparent, and aligned with their goals. In contrast, when LLMs are perceived as a shortcut or opaque authority, they diminish self-efficacy and independent thinking. These findings extend socio-cognitive models of learning and introduces the notion of AI-mediated co-regulation: a process in which motivational and cognitive control are distributed between human and machine. This theoretical contribution re-frames AI in education as a dynamic participant to regulate effort, confidence, and curiosity.

5.3 LLMs as Socio-Technical Mediators of Pedagogical Practice

Pedagogically, the integration of LLMs introduces new configurations of human-AI collaboration that reflect constructivist and socio-cultural principles. As observed in RQ2 and RQ4, educators are embedding LLMs into instructional designs that emphasize scaffolding, reflection, collaboration, and prompt literacy. These strategies transform the classroom into a hybrid learning space or a socio-technical environment where students negotiate meaning with both human and non-human agents.

This shift can be theoretically situated within activity theory and distributed cognition frameworks [33] that conceptualize learning as emergent from interactions between tools, individuals, and communities. LLMs mediate this system by shaping how tasks are represented, how feedback is circulated, and how learners conceptualize their role in problem solving. The introduction of LLMs thus reconfigures the traditional boundaries of pedagogy: teachers move from direct instruction toward orchestration and oversight of AI-supported inquiry, while students navigate new forms of agency and responsibility in co-constructing knowledge.

Evidence also revealed that students with strong foundations, high metacognitive skills, and learning orientations benefit the most from LLM support, while struggling students who need the most help benefit the least or experience harm. This pattern is in contrast to how effective human tutoring operates because human tutors often adapt to meet students where they are. This pattern emerges because LLMs require students to possess sophisticated skills for productive engagement. LLM interactions requires prior knowledge to evaluate output quality, metacognitive awareness to recognize appropriate help-seeking moments, and domain expertise to formulate productive prompts. Böttcher and Schlierkamp [14] observed that experienced students benefits while inexperienced students were "over-strained" when solutions didn't fit the expected contexts. This challenges democratization narratives and suggests that the current trajectory toward unrestricted access may widen rather than narrow achievement gaps.

5.4 Practical Implications and Design Principles

This paper presents an evidence base to support multiple design principles for pedagogically sound LLM integration. Solution withholding emerged as a foundational principle that was validated

across multiple implementations and has been shown to be effective in learning when hints rather than solutions were provided [10, 26, 66]. This principle can be implemented through programmatic guardrails, calibrated hint systems, or providing partial solutions. Metacognitive scaffolding that requires evaluation of LLM outputs have also proved essential to develop critical thinking. This scaffolding can be implemented through a two-stage workflow that also include activities to mandate the critique of LLM outputs. Another principle is integrate a verification process where LLM is used with independent validation to ensure that students developed assessment capabilities rather than vulnerable dependence. It is very important that these principles must be technically enforced in the system rather than relying on student to implement them.

The development of an efficient LLM-based tool to support learning in computing education requires collaborative efforts from the tool designer, class instructor, and the institution. Effective design must engineer each mechanism toward the positive alignment pathway. The tool designer should implement the following guidelines: 1) Withhold complete solutions; use hints, partial scaffolds, and progressive fading of support, 2) Deploy LLMs as interactive reasoning partners rather than answer repositories. Prioritize interactions where the tool poses clarifying questions, prompts the student to explain thinking, and challenges assumptions to activate metacognitive processes, 3) Calibrate feedback specificity to learning goal, and embed verification mechanisms for responses 4) Context should be tailored towards student's demonstrated knowledge and metacognitive state, 5) Build practice into the interaction, 6) Account for the seven moderating context factors, 7) provide instructors with usage analytics.

Instructors should: 1) structure activities to require cognitive participation, 2) Align LLM usage permissions with learning objectives 3) redesign assessment to verify independent competency, and, 4) address student motivations that drive usage patterns. Finally, we provide the following recommendations for institutions: 1)set clear policies that distinguishes acceptable use of LLMs from problematic use, 2)make these policies available and accessible to all students, 3)provide instructor support structures for LLM-integrated pedagogy and support professional development of AI literacy and course redesign efforts for faculty, 4) invest in custom-made LLM-based tools instead of general-purpose LLMs.

We propose a theoretical framework of LLM-supported learning based on the insights generated from this study. The framework has three core mechanisms that are interrelated: 1) the implementation of support mechanism such as adaptive scaffolding through calibrated solution withholding, 2) learner's factors such personal characteristics and AI literacy skills, and 3) the pedagogical context.

Figure 7 presents an overview of the proposed theoretical framework for LLM-supported learning in computing education. The framework shows how three context factors (tool design, student agency, pedagogical context) moderate LLM support mechanisms and exhibit dual-edge characteristics that can produce opposite outcomes. Positive alignment of factors leads to skill development and learning transfer, while misalignment creates conditions for the performance-learning paradox. The equity dimension illustrates the differential impact across student populations.

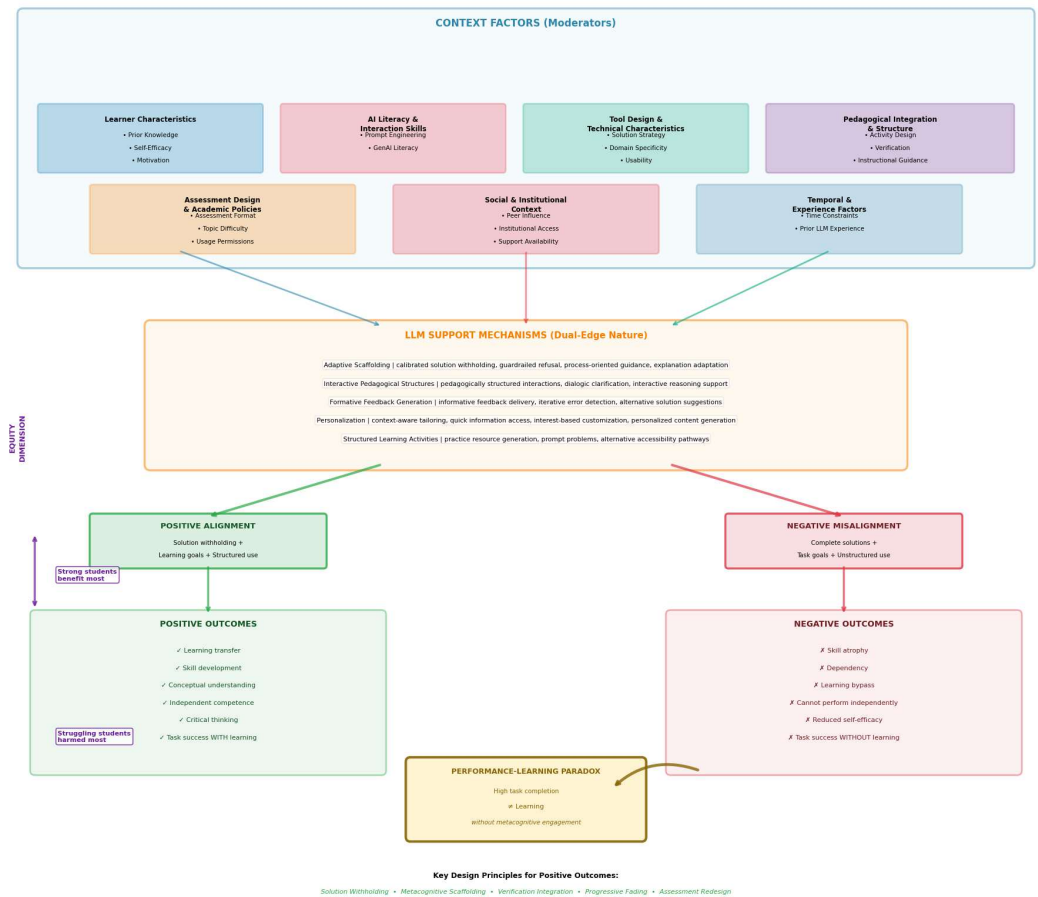


Fig. 7. Framework for LLM-Supported Learning in Computing Education

6 Threats to Validity

This study followed a rigorous and transparent systematic literature review process guided by Kitchenham and Charters [42]. However, we acknowledge several limitations. First, the review was restricted to studies published between 2022 and 2025, and limited to three databases (ACM Digital Library, IEEE Xplore, and Scopus). Although, these sources capture a significant proportion of research in computing education, we may have missed relevant studies in other databases or gray literature.

Second, the inclusion criteria required articles to explicitly discuss the use of LLM-based tools to support student learning in computing education. Consequently, some studies that indirectly address related pedagogical or cognitive processes may have been excluded. Additionally, the review was limited to peer-reviewed publications in English language and this may introduce language or publication bias.

Third, the synthesis relied on the interpretations of the authors during the coding and thematic analysis stages. Although the process included iterative cross-checking to enhance reliability, thematic interpretation inherently involves subjectivity. Finally, given the rapid evolution of LLM

technology and its applications in education, some findings may become outdated as more sophisticated models and pedagogical frameworks emerge. Future reviews should continuously update the evidence base generated in this study to reflect these developments.

7 Future Research Directions

This systematic review synthesizes evidence across 46 studies and revealed critical gaps that requires sustained research attention across technical, pedagogical, and equity dimensions. We identified promising research gaps that include design of tools that inherently do not provide complete solutions, class activities that require students to critically evaluate AI output, and personalized support based on individual skill levels. However, most evidence comes from observational studies rather than controlled experiments. Rigorous trials are needed to compare different design approaches to establish what actually causes better learning outcomes. For example, an open research question is whether requiring students to document what they tried before receiving help genuinely improve learning, or does it simply create a compliance burden without educational benefit. Another open research question is if metacognitive nudges or prompts actually foster deeper thinking. Example of such nudges are reflection questions that ask students to predict what code will do or explain why an approach might work.

We also found substantial variation in how students engage with LLM tools. The variation ranges from iterative questioning for understanding to seeking direct solution for completing tasks. However, the evidence is derived primarily from small-scale qualitative studies and single tool usage logs. Hence, the prevalence, distribution, and predictors of different learning strategies across diverse student populations remains unknown. Future work could find out what specific strategies do students employ when using LLMs for learning, identify predictors of productive engagement, and examine the relationships between learning strategies and learning outcomes.

Evidence from this study also suggests that current approaches to LLM integration may widen achievement gaps where stronger students benefit but struggling students experience harm. Research must move beyond documenting these disparities to testing interventions that address them. Some sample research questions include: 1)What teaching approaches help students from under-represented backgrounds develop the skills needed to use LLMs productively? 2)Can personalized systems that provide more support to struggling students close gaps rather than widen them? 3)Do alternative learning pathways that emphasize natural language over syntax provide genuine equity or simply shift barriers?. Studies must explicitly examine effects across different student populations and test whether proposed solutions actually improve outcomes for those most at risk.

Limited research have examined how educators integrate LLM tools into teaching practice or what support do educators need to do so effectively. Studies should investigate how instructor roles evolve when AI provides feedback, what pedagogical approaches successfully guide students toward productive use, and what institutional support enables thoughtful implementation. This includes understanding barriers educators face in redesigning courses and assessments, what professional development proves helpful, and how policies can enable rather than simply restrict AI use. Research that engages educators as partners to develop integration approaches would ensure findings address real implementation challenges.

Traditional assessments that measures assignment completion and code correctness will fail when students have access to LLM tools during learning. This review revealed a crisis of assessment where students achieved high grades but lack basic skills. Future research must develop and validate new assessment approaches to measure what students actually understand regardless of AI assistance. This includes techniques such as whether having students explain code verbally, debug problems in real time, or document their thinking processes, can reveal the understanding that are missed by written exams. Studies should examine whether these alternative assessments are practical to

implement at scale and fair to diverse learners. Educators will not be able to distinguish genuine understanding from sophisticated copying, without valid ways to measure learning.

Finally, current evidence comes primarily from single semester studies and this leaves critical questions unanswered. For example, do students who rely heavily on LLMs during introductory courses develop independent skills later, or do early dependencies persist into advanced coursework and professional practice?. How do usage patterns established in first-year programming shape approaches to learning throughout the degree program?. Longitudinal research that follow students across multiple years and into their careers would reveal whether observed challenges represent temporary adaptation or persistent competence gaps. This work is essential to understand the true educational cost or benefit of extensive LLM use during foundational learning.

8 Conclusion

This systematic review synthesized evidence from 46 studies that examined LLM-based tools in computing education and revealed a technology with profound but conditional educational impact. The central finding is clear: LLMs can either support or undermine learning depending on how they are designed, integrated, and used.

LLM tools demonstrate genuine educational value when they are thoughtfully designed and pedagogically structured. They provide ‘round the clock’ accessible support, reduce social barriers to help seeking, and enable personalized learning experiences. Students can receive immediate feedback, explore concepts through iterative questioning, and access explanations that is matched to their current understanding. LLMs can provide alternative pathways to programming competence for learners who struggle with syntax but possess strong conceptual thinking. These benefits are real and documented across multiple contexts.

Contrastingly, the same tools that enable the benefits mentioned above also create serious risks. This review identified a troubling performance-learning paradox where students achieve high task completion rates but simultaneously experience fundamental skill atrophy. The studies documented students who submit sophisticated code but cannot write basic statements independently. They earn strong grades on assignments completed with the assistance of LLMs but fail when required to demonstrate competence without LLM support. Traditional measures of learning such as homework completion, assignment grades, and code functionality have become unreliable indicators of actual understanding when students have unrestricted LLM access.

This study also highlighted three critical factors to determine if LLMs help or harm learning. First, tool design matters as the study showed that systems that withhold complete solutions while providing calibrated hints produce better learning outcomes than those that offer unrestricted assistance. Second, student motivation shapes usage because learners who pursue understanding use the tools productively, while those focused on task completion exploit them to bypass learning. Third, pedagogical context determines impact as the study showed that structured integration with appropriate assessment preserves learning processes, while unrestricted access without verification enables sophisticated learning bypass.

The equity implications from this study require urgent attention. Unrestricted LLM access does not democratize learning as often claimed. Instead, students with strong foundations, metacognitive skills, and learning orientations extract substantial value from LLMs, while struggling students who most need support experience compounded difficulties. The struggling students cannot evaluate output quality, recognize when assistance becomes harmful, or develop the independent competence necessary for long term success. Current integration trajectories risk widening rather than narrowing achievement gaps.

The evidence base for addressing these challenges now exists. Effective approaches include personalizing support to individual struggle thresholds, requiring metacognitive engagement with tool

outputs, integrating verification mechanisms that ensure understanding, and redesigning assessment to measure independent competence. Purpose-built educational tools that implements these principles show promise, while unrestricted access to general-purpose LLMs without pedagogical structure consistently produces problematic outcomes.

Students already have pervasive access to LLM tools. The question facing computing education is not whether these technologies will be used but how their use should be structured and assessed. Should we respond with intentional pedagogical design grounded in evidence, or allow unrestricted adoption to continue? The choice determines whether LLMs become tools that enhance learning for all students or a technology that benefits the already advantaged while systematically undermining skill development for those who need support most.

This review provides the theoretical framework and empirical foundation needed for thoughtful integration. The theoretical model developed in this study specifies when, why, and for whom LLMs support learning. This will enable educators and institutions to make evidence based decisions about implementation. The challenge now is how to translate this knowledge into practice to design better tools, restructure pedagogy, reimagine assessment, and ensure equitable access to productive learning experiences. Meeting this challenge will require sustained collaboration among computing education researchers, tool developers, instructors, and institutions that are committed to leverage AI capabilities and preserve the human learning processes. The preservation of human learning processes is essential to develop independent and capable professionals.

References

- [1] 2025. About ACM Digital Library. <https://dl.acm.org/about>. Accessed: 2025-10-10.
- [2] Agnar Aamodt. 1991. *A knowledge-intensive, integrated approach to problem-solving and sustained learning*. Ph.D. Dissertation. Universitetet i Trondheim (Norway).
- [3] Mohammad Abolnejadian, Sharareh Alipour, and Kamyar Taeb. 2024. Leveraging ChatGPT for Adaptive Learning through Personalized Prompt-based Instruction: A CS1 Education Case Study. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery. doi:10.1145/3613905.3637148
- [4] Friday Joseph Agbo, Chris Olivia, Godsalvation Oguibe, Ismaila Temitayo Sanusi, and Godwin Sani. 2025. Computing education using generative artificial intelligence tools: A systematic literature review. *Computers and Education Open* 9 (2025), 100266.
- [5] Yazid Albadarin, Mohammed Saqr, Nicolas Pope, and Markku Tukiainen. 2024. A systematic literature review of empirical research on ChatGPT in education. *Discover Education* 3, 1 (2024), 60.
- [6] Wajdi Aljedaani, Marcelo Medeiros Eler, and P. D. Parthasarathy. 2025. Enhancing Accessibility in Software Engineering Projects with Large Language Models (LLMs). In *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 25–31. doi:10.1145/3641554.3701841
- [7] Matin Amoozadeh, Daye Nam, Daniel Prol, Ali Alfageeh, James Prather, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2024. Student-AI Interaction: A Case Study of CS1 students. (6 2024). doi:10.1145/3699538.3699567
- [8] A Attard and A Dingli. 2024. Empowering educators: Leveraging large language models to streamline content creation in education. In *ICERI2024 Proceedings*. IATED, 1312–1321.
- [9] Imen Azaiz, Natalie Kiesler, and Sven Strickroth. 2024. Feedback-Generation for Programming Exercises With GPT-4. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 31–37. doi:10.1145/3649217.3653594
- [10] Patrick Bassner, Eduard Frankford, and Stephan Krusche. 2024. Iris: An AI-Driven Virtual Tutor for Computer Science Education. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 394–400. doi:10.1145/3649217.3653543
- [11] Rodrigo Borela, Zhixian Liding, and Melinda McDaniel. 2025. Enhancing CS1 Education through Experiential Learning with Robotics Projects. In *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 144–150. doi:10.1145/3641554.3701810
- [12] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
- [13] Ritvik Budhiraja, Ishika Joshi, Harshal Akolekar, Jagat Sesh Challa, and Dhruv Kumar. 2024. "It's not like Jarvis, but it's pretty close!" - Examining ChatGPT's Usage among Undergraduate Students in Computer Science. In *ACM*

- International Conference Proceeding Series*. Association for Computing Machinery, 124–133. doi:10.1145/3636243.3636257
- [14] Axel Böttcher, Veronika Thurner, and Benedikt Zönnchen. 2025. Concepts for Teaching Software Development in the Age of AI-Tools. In *2025 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1–10. doi:10.1109/EDUCON62633.2025.11016505
 - [15] Timothy A Carey and Richard J Mullan. 2004. What is Socratic questioning? *Psychotherapy: theory, research, practice, training* 41, 3 (2004), 217.
 - [16] Peng Chen, Dong Yang, Ahmed Hosny Saleh Metwally, Jari Lavonen, and Xiao Wang. 2023. Fostering computational thinking through unplugged activities: A systematic literature review and meta-analysis. *International Journal of STEM Education* 10, 1 (2023), 47.
 - [17] Kendra Cherry. 2025. *What Is Learning?* <https://www.verywellmind.com/what-is-learning-2795332> Updated January 08, 2025; reviewed by Amy Morin, LCSW.
 - [18] Zhendong Chu, Shen Wang, Jian Xie, Tinghui Zhu, Yibo Yan, Jinheng Ye, Aoxiao Zhong, Xuming Hu, Jing Liang, Philip S Yu, et al. 2025. Llm agents for education: Advances and applications. *arXiv preprint arXiv:2503.11733* (2025).
 - [19] Bruno Pereira Cipriano and Pedro Alves. 2024. LLMs Still Can't Avoid Instanceof: An Investigation Into GPT-3.5, GPT-4 and Bard's Capacity to Handle Object-Oriented Programming Assignments. In *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, 162–169. doi:10.1145/3639474.3640052
 - [20] Edward L Deci and Richard M Ryan. 2012. Self-determination theory. *Handbook of theories of social psychology* 1, 20 (2012), 416–436.
 - [21] Paul Denny, Stephen Macneil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable Characteristics for AI Teaching Assistants in Programming Education. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 408–414. doi:10.1145/3649217.3653574
 - [22] Paul Denny, Stephen MacNeil, Jaromir Savelka, Leo Porter, and Andrew Luxton-Reilly. 2024. Desirable characteristics for ai teaching assistants in programming education. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 408–414.
 - [23] Linda Elder and Richard Paul. 1998. The role of Socratic questioning in thinking, teaching, and learning. *The Clearing House* 71, 5 (1998), 297–301.
 - [24] Said Elnaffar, Farzad Rashidi, and Abedallah Zaid Abualkishik. 2025. Teaching with AI: A Systematic Review of Chatbots, Generative Tools, and Tutoring Systems in Programming Education. *arXiv preprint arXiv:2510.03884* (2025).
 - [25] Elsevier. 2025. Scopus Product Information. <https://www.elsevier.com/products/scopus>. Accessed via university proxy on 2025-10-10.
 - [26] Tony Haoran Feng, Andrew Luxton-Reilly, Burkhard C. Wünsche, and Paul Denny. 2024. From Automation to Cognition: Redefining the Roles of Educators and Generative AI in Computing Education. (12 2024). doi:10.1145/3716640.3716658
 - [27] Vincent Ferrari, Andre Didierjean, and Evelyne Marmeche. 2008. Effect of expertise acquisition on strategic perception: The example of chess. *Quarterly Journal of Experimental Psychology* 61, 8 (2008), 1265–1280.
 - [28] Sally Fincher, Johan Jeuring, Craig S Miller, Peter Donaldson, Benedict Du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, et al. 2020. Capturing and characterising notional machines. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 502–503.
 - [29] Eduard Frankford, Clemens Sauerwein, Patrick Bassner, Stephan Krusche, and Ruth Breu. 2024. AI-Tutoring in Software Engineering Education Experiences with Large Language Models in Programming Assessments. In *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, 309–319. doi:10.1145/3639474.3640061
 - [30] Alkmini Gaitantzi and Ioannis Kazanidis. 2025. The role of artificial intelligence in computer science education: A systematic review with a focus on database instruction. *Applied Sciences* 15, 7 (2025), 3960.
 - [31] Sofia Sánchez González. 2023. What is BioGPT, Microsoft's language model for Life Sciences? <https://www.narrativa.com/what-is-biogpt-microsofts-language-model-for-life-sciences/>. Accessed: 2025-10-28.
 - [32] Virginia Grande, Natalie Kiesler, and Maria Andreina Francisco R. 2024. Student Perspectives on Using a Large Language Model (LLM) for an Assignment on Professional Ethics. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 478–484. doi:10.1145/3649217.3653624
 - [33] Christine A Halverson. 2002. Activity theory and distributed cognition: Or what does CSCW need to DO with theories? *Computer Supported Cooperative Work (CSCW)* 11, 1 (2002), 243–267.
 - [34] Michael Hitchens and Rowan Tulloch. 2018. A gamification design for the classroom. *Interactive Technology and Smart Education* 15, 1 (2018), 28–45.
 - [35] Irene Hou, Sophia Mettill, Zhuo Li, Owen Man, Cynthia Zastudil, and Stephen MacNeil. 2024. The Effects of Generative AI on Computing Students' Help-Seeking Preferences. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 39–48. doi:10.1145/3636243.3636248

- [36] Guiu Puigcercos i Vilar, Parvez Rashid, and Navid Hashemi Tonekaboni. 2025. Accessible and Reliable AI Coding Tutors: Augmenting Large Language Models with Retrieval-Augmented Generation for Java Programming. In *2025 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1–10. doi:10.1109/EDUCON62633.2025.11016497
- [37] Institute of Electrical and Electronics Engineers. 2025. About IEEE Xplore Digital Library. <https://ieeexplore.ieee.org/Xplorehelp/overview-of-ieee-xplore/about-ieee-xplore>. Accessed: 2025-10-10.
- [38] Ishika Joshi, Ritvik Budhiraja, Harshal Dev, Jahnvi Kadia, Mohammad Osama Ataullah, Sayan Mitra, Harshal D Akolekar, and Dhruv Kumar. 2024. ChatGPT in the classroom: An analysis of its strengths and weaknesses for solving undergraduate computer science questions. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 625–631.
- [39] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating LLM-generated Worked Examples in an Introductory Programming Course. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 77–86. doi:10.1145/3636243.3636252
- [40] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Conference on Human Factors in Computing Systems - Proceedings*. Association for Computing Machinery. doi:10.1145/3544548.3580919
- [41] Nam Wook Kim, Hyung-Kwon Ko, Grace Myers, and Benjamin Bach. 2024. ChatGPT in Data Visualization Education: A Student Perspective. (4 2024). doi:10.1109/VL/HCC60511.2024.00022
- [42] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering—a systematic literature review. *Information and software technology* 51, 1 (2009), 7–15.
- [43] David A Kolb. 2014. *Experiential learning: Experience as the source of learning and development*. FT press.
- [44] Kai Korpimies, Antti Laaksonen, and Matti Luukkainen. 2024. Unrestricted use of LLMs in a software project course: Student perceptions on learning and impact on course performance. In *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*. 1–7.
- [45] Amy Chen Kulesa, Marisa Mission, Michelle Croft, and Mary K Wells. [n. d.]. Productive Struggle. ([n. d.]).
- [46] Dimitra Lamprinou and Fotini Paraskeva. 2015. Gamification design framework based on SDT for student motivation. In *2015 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL)*. IEEE, 406–410.
- [47] Diana Laurillard. 1979. The processes of student learning. *Higher education* 8, 4 (1979), 395–409.
- [48] Warren Li. 2024. *Understanding Learners and the Interplay Between Metacognitive Judgements of Learning and AI-Generated Explanations*. Ph. D. Dissertation.
- [49] Ying Li, Runze Yang, Xiantao Zhang, Peng Shi, Da Yang, and Xuefei Huang. 2024. ProgMate: An Intelligent Programming Assistant Based on LLM. In *Proceedings - Frontiers in Education Conference, FIE*. Institute of Electrical and Electronics Engineers Inc. doi:10.1109/FIE61694.2024.10893312
- [50] Hao Liu. 2023. PubMed GPT: A Domain-Specific Large Language Model for Biomedical Text. <https://medium.com/@haoliu/pubmed-gpt-a-domain-specific-large-language-model-for-biomedical-text-b3bbd6fc32cb>. Accessed: 2025-10-28.
- [51] Rongxin Liu, Julianna Zhao, Benjamin Xu, Christopher Perez, Yuliya Zhukovets, and David J Malan. 2025. Improving AI in CS50. (2025).
- [52] Suqing Liu, Zezhu Yu, Feiran Huang, Yousef Bulbulia, Andreas Bergen, and Michael Liut. 2024. Can Small Language Models With Retrieval-Augmented Generation Replace Large Language Models When Learning Computer Science?. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 388–393. doi:10.1145/3649217.3653554
- [53] Wenhao Lyu, Yimeng Wang, Tingting Rachel Chung, Yifan Sun, and Yixuan Zhang. 2024. Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study. In *L@S 2024 - Proceedings of the 11th ACM Conference on Learning @ Scale*. Association for Computing Machinery, Inc, 63–74. doi:10.1145/3657604.3662036
- [54] Iris Ma, Alberto Krone-Martins, and Cristina Videira Lopes. 2024. Integrating AI Tutors in a Programming Course. In *SIGCSE Virtual 2024 - Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1*. Association for Computing Machinery, Inc, 130–136. doi:10.1145/3649165.3690094
- [55] Kes McCormick and Bernadett Kiss. 2015. Learning through renovations for urban sustainability: The case of the Malmö Innovation Platform. *Current Opinion in Environmental Sustainability* 16 (2015), 44–50.
- [56] Katherine B McKeithen, Judith S Reitman, Henry H Rueter, and Stephen C Hirtle. 1981. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology* 13, 3 (1981), 307–325.
- [57] André Menolli, Bruno Strik, and Luiz Rodrigues. 2024. Teaching Refactoring to Improve Code Quality with ChatGPT: An Experience Report in Undergraduate Lessons. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 563–574. doi:10.1145/3701625.3701681

- [58] Simone Mezzaro, Alessio Gambi, and Gordon Fraser. 2024. An Empirical Study on How Large Language Models Impact Software Testing Learning. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 555–564. doi:10.1145/3661167.3661273
- [59] Rosario Michel-Villarreal, Eliseo Vilalta-Perdomo, David Ernesto Salinas-Navarro, Ricardo Thierry-Aguilera, and Flor Silvestre Gerardou. 2023. Challenges and opportunities of generative AI for higher education as explained by ChatGPT. *Education Sciences* 13, 9 (2023), 856.
- [60] Moritz Mueller, Corinna List, and Michael Kipp. 2025. The Power of Context: An LLM-based Programming Tutor with Focused and Proactive Feedback. In *Proceedings of the 6th European Conference on Software Engineering Education* (New York, NY, USA). ACM, 1–10. doi:10.1145/3723010.3723034
- [61] Alexander Tobias Neumann, Yue Yin, Sulayman Sowe, Stefan Decker, and Matthias Jarke. 2024. An LLM-Driven Chatbot in Higher Education for Databases and Information Systems. *IEEE Transactions on Education* (2024). doi:10.1109/TE.2024.3467912
- [62] Andrés Neyem, Agustín Rios-Letelier, Kevin Céspedes-Arancibia, Juan Pablo Sandoval Alcocer, and Marcelo Mendoza. 2024. Enhancing commit message quality in software capstone projects with generative AI. *SoftwareX* 28 (12 2024). doi:10.1016/j.softx.2024.101947
- [63] Valeria Ramirez Osorio, Angela Zavaleta Bernuy, Bogdan Simion, and Michael Liut. 2025. Understanding the Impact of Using Generative AI Tools in a Database Course. In *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education*, Vol. 1. Association for Computing Machinery, Inc, 959–965. doi:10.1145/3641554.3701785
- [64] Aadarsh Padiyath, Xinying Hou, Amy Pang, Diego Viramontes Vargas, Xingjian Gu, Tamara Nelson-Fromm, Zihan Wu, Mark Guzdial, and Barbara Ericson. 2024. Insights from Social Shaping Theory: The Appropriation of Large Language Models in an Undergraduate Programming Course. In *ICER 2024 - ACM Conference on International Computing Education Research*, Vol. 1. Association for Computing Machinery, Inc, 114–130. doi:10.1145/3632620.3671098
- [65] Wuming Pan, Ying Yang, and H Yin. 2024. Integrating LLMs and Software-Defined Resources for Enhanced Demonstrative Cloud Computing Education in University Curricula. *Journal of Infrastructure, Policy and Development* 8, 12 (2024), 8751.
- [66] Maciej Pankiewicz and Ryan S. Baker. 2024. Navigating Compiler Errors with AI Assistance - A Study of GPT Hints in an Introductory Programming Course. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 94–100. doi:10.1145/3649217.3653608
- [67] Andre Fabiano Pereira and Rafael Ferreira Mello. 2025. A Systematic Literature Review on Large Language Models Applications in Computer Programming Teaching Evaluation Process. *IEEE Access* (2025).
- [68] Kim Petram. 2010. Bloom’s Taxonomy: Levels of Understanding. <https://www.psia-nw.org/newsletter-articles/blooms-taxonomy-levels-of-understanding/>. Accessed: 2025-10-28.
- [69] Olga Petrovska, Lee Clift, Faron Moller, and Rebecca Pearsall. 2024. Incorporating Generative AI into Software Development Education. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 37–40. doi:10.1145/3633053.3633057
- [70] Rob Phillips, Carmel McNaught, and Gregor Kennedy. 2010. Towards a generalised conceptual framework for learning: the Learning Environment, Learning Processes and Learning Outcomes (LEPO) framework. In *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education (AACE), 2495–2504.
- [71] Alexandru Pirjan and Dana-Mihaela PETROȘANU. 2024. EXPLORING LARGE LANGUAGE MODELS IN THE EDUCATION PROCESS WITH A VIEW TOWARDS TRANSFORMING PERSONALIZED LEARNING. *Journal of Information Systems & Operations Management* 18, 2 (2024).
- [72] Farman Ali Pirzado, Awais Ahmed, Román Alejandro Mendoza-Urdiales, and Hugo Terashima-Marin. 2024. Navigating the pitfalls: analyzing the behavior of LLMs as a coding assistant for computer science students—a systematic review of the literature. *IEEE access* 12 (2024), 112605–112625.
- [73] Alfonso Piscitelli, Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. 2025. Influence of Large Language Models on Programming Assignments - A user study. In *Proceedings of the 2024 the 16th International Conference on Education Technology and Computers, ICETC 2024*. Association for Computing Machinery, Inc, 33–38. doi:10.1145/3702163.3702168
- [74] Kishore Prakash, Shashwat Rao, Rayan Hamza, Jack Lukich, Vatsal Chaudhari, and Arnab Nandi. 2024. Integrating LLMs into Database Systems Education. (2024). doi:10.1145/3663649
- [75] Basit Qureshi. 2023. ChatGPT in Computer Science Curriculum Assessment: An analysis of Its Successes and Shortcomings. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 7–13. doi:10.1145/3613944.3613946
- [76] Nishat Raihan, Mohammed Latif Siddiq, Joanna CS Santos, and Marcos Zampieri. 2025. Large language models in computer science education: A systematic literature review. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*. 938–944.

- [77] Jaakko Rajala, Jenni Hukkanen, Maria Hartikainen, and Pia Niemelä. 2023. "Call me Kiran" ChatGPT as a Tutoring Chatbot in a Computer Science Course. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 83–94. doi:10.1145/3616961.3616974
- [78] David Reeping and Aarohe Shah. 2024. Board 50: Work in progress: A systematic review of embedding large language models in engineering and computing education. In *2024 ASEE Annual Conference & Exposition*.
- [79] Sara Riazzi and Pedram Rooshenas. 2024. LLM-Driven Feedback for Enhancing Conceptual Design Learning in Database Systems Courses. (12 2024). doi:10.1145/3641554.3701940
- [80] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172.
- [81] Anthony V Robins. 2019. 12–Novice programmers and introductory programming. *The Cambridge handbook of computing education research* (2019), 327–376.
- [82] André Rocha, Lino Sousa, Mário Alves, and Armando Sousa. 2024. The underlying potential of NLP for microcontroller programming education. *Computer Applications in Engineering Education* (11 2024). doi:10.1002/cae.22778
- [83] Hemilis Joyse Barbosa Rocha, Patricia Cabral De Azevedo Restelli Tedesco, and Evandro De Barros Costa. 2023. On the use of feedback in learning computer programming by novices: a systematic literature mapping. *Informatics in Education* 22, 2 (2023), 209.
- [84] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *International Conference on Intelligent User Interfaces, Proceedings IUI*. Association for Computing Machinery, 491–514. doi:10.1145/3581641.3584037
- [85] Daniel Russo. 2024. Navigating the Complexity of Generative AI Adoption in Software Engineering. *ACM Transactions on Software Engineering and Methodology* 33 (6 2024). Issue 5. doi:10.1145/3652154
- [86] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, Vol. 1. Association for Computing Machinery, Inc, 27–43. doi:10.1145/3501385.3543957
- [87] Karim Shabani, Mohamad Khatib, and Saman Ebadi. 2010. Vygotsky's zone of proximal development: Instructional implications and teachers' professional development. *English language teaching* 3, 4 (2010), 237–248.
- [88] Tariq Shahzad, Tehseen Mazhar, Muhammad Usman Tariq, Wasim Ahmad, Khmaies Ouahada, and Habib Hamam. 2025. A comprehensive review of large language models: issues and solutions in learning environments. *Discover Sustainability* 6, 1 (2025), 27.
- [89] Sahil Sharma, Puneet Mittal, Mukesh Kumar, and Vivek Bhardwaj. 2025. The role of large language models in personalized learning: a systematic review of educational impact. *Discover Sustainability* 6, 1 (2025), 1–24.
- [90] Jinnie Shin, Laura Cruz-Castro, Zhenlin Yang, Gabriel Castelblanco, Ashish Aggarwal, Walter L Leite, and Bruce F Carroll. 2024. Understanding Optimal Interactions Between Students and A Chatbot During A Programming Task. In *2024 Winter Simulation Conference (WSC)*. IEEE, 3106–3117.
- [91] C. Estelle Smith, Kylee Shiekh, Hayden Cooreman, Sharfi Rahman, Yifei Zhu, Md Kamrul Siam, Michael Ivanitskiy, Ahmed M. Ahmed, Michael Hallinan, Alexander Grisak, and Gabe Fierro. 2024. Early Adoption of Generative Artificial Intelligence in Computing Education: Emergent Student Use Cases and Perspectives in 2023. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, Vol. 1. Association for Computing Machinery, 3–9. doi:10.1145/3649217.3653575
- [92] Barbara J Stevens. 1976. The teaching-learning process. *Nurse Educator* 1, 1 (1976), 9–12.
- [93] Xiaohang Tang, Sam Wong, Kevin Pu, Xi Chen, Yalong Yang, and Yan Chen. 2024. VizGroup: An AI-assisted Event-driven System for Collaborative Programming Learning Analytics. In *UIST 2024 - Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, Inc. doi:10.1145/3654777.3676347
- [94] Olle Ten Cate, Linda Snell, Karen Mann, and Jan Vermunt. 2004. Orienting teaching toward the learning process. *Academic Medicine* 79, 3 (2004), 219–228.
- [95] W Scott Terry. 2017. *Learning and memory: Basic principles, processes, and procedures*. Routledge.
- [96] Christian Maximilian Thurn and Simona Daguati. 2025. Optimizing the Learning Process in Higher Education: The Six Process Features of Learning. *Zeitschrift für Hochschulentwicklung* 20, 1 (2025), 87–108.
- [97] Neng Wang, Hongyang Yang, and Christina Dan Wang. 2023. Fingpt: Instruction tuning benchmark for open-source large language models in financial datasets. *arXiv preprint arXiv:2310.04793* (2023).
- [98] Hiroko Kawaguchi Warshawer. 2015. Productive struggle in middle school mathematics classrooms. *Journal of Mathematics Teacher Education* 18, 4 (2015), 375–400.
- [99] Wikipedia contributors. 2025. *Learning*. <https://en.wikipedia.org/wiki/Learning> Accessed: 2025-10-20.

[100] Greta Winograd and Jonathan P Rust. 2014. Stigma, awareness of support services, and academic help-seeking among historically underrepresented first-year college students. *Learning Assistance Review (TLAR)* 19, 2 (2014).

[101] Jialu Zhang, José Pablo Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. 2024. PyDex: Repairing Bugs in Introductory Python Assignments using LLMs. *Proceedings of the ACM on Programming Languages* 8 (4 2024). Issue OOPSLA1. doi:10.1145/3649850

[102] Haiyan Zhao, Fan Yang, Bo Shen, Himabindu Lakkaraju, and Mengnan Du. 2024. Towards uncovering how large language model works: An explainability perspective. *arXiv preprint arXiv:2402.10688* (2024).

[103] Zhi Zhou, Jiang-Xin Shi, Peng-Xiao Song, Xiao-Wen Yang, Yi-Xuan Jin, Lan-Zhe Guo, and Yu-Feng Li. 2024. Lawgpt: A chinese legal knowledge-enhanced large language model. *arXiv preprint arXiv:2406.04614* (2024).

[104] Barry J Zimmerman and Dale H Schunk. 2011. Self-regulated learning and performance: An introduction and an overview. *Handbook of self-regulation of learning and performance* (2011), 15–26.

A List of Primary Studies

Table 3. List of primary studies

Title	Citation
VizGroup An AI Assisted Event Driven System for Real Time Collaborative Programming Learning Analytics	[93]
Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming	[40]
Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study	[53]
The Power of Context: An LLM-based Programming Tutor with Focused and Proactive Feedback	[60]
The underlying potential of NLP for Microcontroller Programming Education	[83]
ChatGPT in Data Visualization Education: A Student Perspective	[41]
Evaluating LLM-generated Worked Examples in an Introductory Programming Course	[39]
Enhancing commit message quality in software capstone projects with generative AI	[62]
Iris: An AI-Driven Virtual Tutor For Computer Science Education	[10]
ProgMate: An Intelligent Programming Assistant based on LLM	[49]
Leveraging ChatGPT for Adaptive Learning through Personalized Prompt-based Instruction: A CS1 Education Case Study	[3]
Understanding the Impact of Using Generative AI Tools in a Database Course	[63]
Concepts for Teaching Software Development in the Age of AI-Tools	[14]
LLM-Driven Feedback for Enhancing Conceptual Design Learning in Database Systems Courses	[79]
The Effects of Generative AI on Computing Students' Help-Seeking Preferences	[35]
Early Adoption of Generative Artificial Intelligence in Computing Education: Emergent Student Use Cases and Perspectives in 2023	[91]
Navigating the Complexity of Generative AI Adoption in Software Engineering	[85]
The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development	[84]
An LLM-Driven Chatbot in Higher Education for Databases and Information Systems	[61]
Student Perspectives on Using a Large Language Model (LLM) for an Assignment on Professional Ethics	[32]
Insights from Social Shaping Theory: The Appropriation of Large Language Models in an Undergraduate Programming Course	[64]
Can Small Language Models With Retrieval-Augmented Generation Replace Large Language Models When Learning Computer Science	[51]
Desirable Characteristics for AI Teaching Assistants in Programming Education	[21]
Navigating Compiler Errors with AI Assistance - A Study of GPT Hints in an Introductory Programming Course	[66]
It's not like Jarvis, but it's pretty close!" - Examining ChatGPT's Usage among Undergraduate Students in Computer Science	[13]
Accessible and Reliable AI Coding Tutors: Augmenting Large Language Models with Retrieval-Augmented Generation for Java Programming.	[36]
Integrating AI Tutors in a Programming Course	[54]
Feedback-Generation for Programming Exercises With GPT-4	[9]
AI-Tutoring in Software Engineering Education: Experiences with Large Language Models in Programming Assessments	[29]
Enhancing CS1 Education through Experiential Learning with Robotics Projects	[11]
An Empirical Study on How Large Language Models Impact Software Testing Learning	[58]
Student-AI Interaction: A Case Study of CS1 students	[7]
Unrestricted Use of LLMs in a Software Project Course: Student Perceptions on Learning and Impact on Course Performance	[44]
LLMs Still Can't Avoid Instanceof: An Investigation Into GPT-3.5, GPT-4 and Bard's Capacity to Handle Object-Oriented Programming Assignments	[19]
Enhancing Accessibility in Software Engineering Projects with Large Language Models (LLMs)	[6]
Teaching Refactoring to Improve Code Quality with ChatGPT: An Experience Report in Undergraduate Lessons	[57]
PyDex: Repairing Bugs in Introductory Python Assignments using LLMs	[101]
Incorporating Generative AI into Software Development Education	[69]
Integrating Conversational Large Language Models into Student Learning: A Case Study of ChatGPT in Software Engineering Education	[52]
Integrating LLMs into Database Systems Education	[74]
Influence of Large Language Models on Programming Assignments - A user study	[73]
From Automation to Cognition: Redefining the Roles of Educators and Generative AI in Computing Education	[26]
ChatGPT in Computer Science Curriculum Assessment- An analysis of its successes and shortcomings	[75]
Call me Kiran ChatGPT as a Tutoring Chatbot in a Computer Science Course	[77]
Automatic Generation Of Programming Exercises And Code Explanations Using Large Language Models	[86]
Understanding Optimal Interactions Between Students and a Chatbot During a Programming Task	[90]

Table 4. Classification of Custom LLM-Based Learning Tools Across Four Dimensions

Tool	Functions	Interaction Mode	Pedagogical Approach	Timing
Progmate [49]	Evaluates work, Tracks progress, Recommends resources, Personalizes content	System-initiated	Adaptive scaffolding, Error identification	Post-submission, Meta-cognitive
AITutor [29]	Evaluates work, Provides hints, Explains concepts	Learner-initiated, One-shot	Solution withholding, Error identification	During-practice, Post-submission
MoodleBot [61]	Generates problems, Answers questions, Explains concepts, Recommends resources	Learner-initiated, Conversational	Direct instruction, Worked examples	Pre-instruction, During-practice, Meta-cognitive
CodeTutor [53]	Answers questions	Learner-initiated, Conversational	Direct instruction	During-practice
Feedback Generator [79]	Evaluates work (EERD diagrams), Provides feedback	Learner-initiated, Iterative	Error identification, Rubric-guided	During-practice, Post-submission
Iris [10]	Provides hints, Answers questions, Evaluates work	Learner-initiated, Conversational	Socratic questioning, Solution withholding, Adaptive scaffolding	During-practice
RAGMAN [54]	Answers questions, Provides guidance	Learner-initiated, Conversational	Solution withholding, Direct instruction	During-practice
ContextTutor [60]	Evaluates work, Provides feedback, Tracks progress	Learner-initiated, Dual feedback	Error identification, Reflective feedback, Solution withholding	During-practice, Post-submission
DB Tutor [74]	Answers questions, Generates examples, Explains results, Tracks performance	Learner-initiated, Conversational	Direct instruction, Worked examples	During-practice, Meta-cognitive
Code Help [21]	Answers questions, Provides guidance, Explains concepts	Learner-initiated, Conversational	Solution withholding, Uses pseudocode, Positive tone	During-practice
LetsCommit [62]	Generates commit messages	Learner-initiated, One-shot	Worked examples, Direct instruction	During-practice
AI Defenders [58]	Answers questions, Explains concepts, Provides code examples	Learner-initiated, Conversational, Context-aware	Structured responses, Direct instruction, Worked examples	During-practice
RAG AI Tutor [36]	Answers questions, Provides best practices	Learner-initiated, Conversational	Direct instruction, Worked examples	During-practice
Programmer's Assistant [84]	Answers questions, Provides assistance	Learner-initiated, Conversational, Context-aware	Direct instruction	During-practice
Coding Steps [40]	Generates code, Autocomplete, Error detection, Documentation	Learner-initiated, One-shot	Direct instruction, Context-aware	During-practice
ClueBot [73]	Answers questions, Validates solutions	Learner-initiated, Conversational	Direct instruction, Error identification	During-practice
PyDex [101]	Repairs programs, Generates patches	System-initiated	Error correction, Few-shot learning, Minimal intervention	During-practice, Post-submission
VizGroup [93]	Analyzes discussions, Tags messages, Summarizes topics, Visualizes patterns	System-initiated	Formative assessment, Meta-cognitive, Reflective feedback	During-practice, Meta-cognitive
Adaptive Prompt System [3]	Generates explanations/examples/exercises, Personalizes content	Learner-initiated, Context-maintained	Adaptive scaffolding, Direct instruction, Context-aware	Pre-instruction, During-practice, Meta-cognitive
Exercise Generator [86]	Generates exercises, Creates explanations	System-initiated	Worked examples, Direct instruction	Pre-instruction
WorkedGen [39]	Generates worked examples, Interactive explanations, Answers questions	Learner-initiated, Interactive	Worked examples with fading, Scaffolded learning, Interactive feedback	During-practice
GPT Hint System [66]	Generates error hints, Educational explanations	System-initiated, One-shot	Error identification, Solution strategy, Educational concepts, Solution withholding	During-practice