

Review

A Scoping Review of Vibe Coding and AI-Assisted Programming in University Computing Education

Jing Tian 

NUS-ISS, National University of Singapore, Singapore 119615, Singapore; tianjing@nus.edu.sg

Abstract

Background: Generative AI coding tools are increasingly used in university computing education, including emerging practices commonly described as vibe coding. However, evidence on their educational effects is scattered across studies that differ in interaction patterns, instructional designs, assessment practices, and learning contexts. **Methods:** We conducted a scoping review using a PRISMA-guided process and synthesized 37 studies published between 2022 and July 2026. Five research questions examined student–AI interaction patterns, learning outcomes, task-level instructional strategies, assessment and academic integrity, and curriculum alignment with emerging professional practice. **Results:** The reviewed articles reported that students commonly relied on debugging-oriented prompts, common-case testing, and short edit-run cycles, while stronger performance was associated with more exploratory and evaluative interactions. Reported outcomes included conceptual learning, programming productivity, and affective benefits. Emerging instructional strategies included planning with hints, adaptive scaffolding, dual-mode assignments, and process visibility. **Conclusions:** The educational value of AI-assisted coding appears to depend less on tool access than on how student interaction is scaffolded, assessed, and connected to emerging competencies in specification, verification, testing, and critical evaluation. Therefore, computing education should treat AI-assisted programming not simply as accelerated code generation, but as a practice requiring deliberate reasoning and responsible oversight.

Keywords: vibe coding; AI-assisted coding; computing education; scoping review

1. Introduction

Artificial intelligence (AI) has moved quickly from research labs into industrial applications and higher education (Fu et al., 2025; Ng et al., 2025). Today’s foundation models write fluent text, perform multi-step reasoning, and help with tasks ranging from explanation and tutoring to writing computer programs. These capabilities present clear opportunities for teaching and learning (Küchemann et al., 2025; Tian, 2026). In computing education, the spread of generative AI, particularly *large language models* (LLMs), has given rise to a prompt-driven way of working now widely called *vibe coding*. The term comes from a February 2025 post by the AI researcher Andrej Karpathy (Karpathy, 2025). A programmer describes in natural language what the software should do and lets the AI model generate and debug the implementation. This is fundamentally different from earlier code-completion tools, which mostly predicted the next local fragment from surrounding context (GitHub, 2026).

The emergence of prompt-based programming has significant implications for university computing education. Students are increasingly using generative AI for coursework



Academic Editor: Yupei Zhang

Received: 9 August 2026

Revised: 14 September 2026

Accepted: 5 October 2026

Published: 9 October 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

activities that range from information gathering to program generation and assignment completion. Consequently, computing educators need to reconsider learning activities, instructional guidance, assessment design, and expectations concerning students’ independent programming competence.

To synthesize this emerging evidence, this study presents a scoping review of vibe coding and related AI-assisted programming practices in university computing education. Existing studies span student–AI interaction, instructional methods, and assessment settings, but this evidence has not yet been systematically mapped. A scoping review is appropriate because the terminology remains unstable. For example, *vibe coding*, *AI-assisted programming/coding*, *LLM-assisted programming/coding*, and *AI-assisted/LLM-assisted software development* are used in the literature. Although previous reviews have examined topics such as AI-supported programming resources, collaborative coding, reported educational effects, and implementation challenges (Bakharia & Abdi, 2024; Kaewsaiha et al., 2026; J. Liu & Li, 2024), the available evidence remains fragmented. Table 1 summarizes the differences between this review and three earlier reviews in terms of information sources, publication coverage, and analytical scope.

Table 1. The summarized differences between this paper and existing review papers.

Reference	Year	Is It a Systematic Review?	Period	Sources	Included Studies	Focus
(Bakharia & Abdi, 2024)	2024	No	From 2022 to January 2024	Programming books	25	Programming books
(J. Liu & Li, 2024)	2024	Yes	From 2022 to 27 November 2023	Web of Science, Springer Link Digital Library, Google Scholar	23	Collaborative programming approach
(Kaewsaiha et al., 2026)	2026	Yes	From 2023 to 2025	Scopus, IEEE Xplore, ACM Digital Library, ERIC Online Library	19	Reported landscape of usage, impact, challenge, and pedagogical strategies
Ours		Yes	From 2022 to July 2026	Scopus, IEEE Xplore, ACM Digital Library, Web of Science, arXiv	37	University computing education

This study makes two main contributions.

- First, it provides a systematic scoping review of vibe coding and related AI-assisted programming practices in university computing education. A review process is employed to ensure transparent study identification, screening, eligibility assessment, and reporting.
- Second, it develops an evidence-based synthesis structured around five research questions. It examines student–AI interaction, effects on programming learning outcomes, pedagogical and assessment design, academic integrity, and the alignment of computing curricula with evolving professional software-development practice.

The rest of this paper is organized as follows. Section 2 presents the five research questions and their rationales, the detailed literature search, and the identification, screening, and selection process. Section 3 presents the findings of the five research questions from the included articles. Section 4 presents the discussion and limitations, followed by the conclusions in Section 5.

2. Methods

2.1. Research Questions

To examine the use of vibe coding in university computing education, this paper formulates five *research questions* (RQs). The definitions and rationales of these RQs are presented as follows.

- RQ1 (Student–AI interaction): How do university students interact with AI coding tools, and how do these interaction patterns vary by prior programming competence? This question examines how students prompt, test, debug, revise, and evaluate AI-generated code.
- RQ2 (Learning outcomes): Under what conditions does AI-assisted coding support conceptual learning, programming performance, and affective outcomes in university computing education? This question distinguishes improvements in learning and competence from gains limited to task completion or productivity.
- RQ3 (Pedagogical strategy): Which task-level instructional strategies promote productive cognitive engagement during AI-assisted coding? This question focuses on interventions such as planning with hints, adaptive scaffolding, dual-mode assignment, and process visibility.
- RQ4 (Assessment and academic integrity): How are assessment and academic-integrity practices being redesigned to account for AI-assisted coding in university computing education? This question examines approaches such as process-based assessment, prompt-log submission, oral explanation, and activity monitoring.
- RQ5 (Curriculum and professional alignment): How are computing curricula responding to AI-assisted coding, and how well do the proposed competencies align with emerging professional practice? This question maps curricular responses related to code review, verification, and supervision of AI coding agents.

2.2. Search Strategy

This review was conducted in accordance with the updated methodological guidance for scoping reviews (Peters et al., 2020) and reported according to the Preferred Reporting Items for Systematic Reviews and Meta-Analyses Extension for Scoping Reviews (Tricco et al., 2018). The searches were conducted on 7 July 2026 across four bibliographic databases, Web of Science, Scopus, IEEE Xplore, and the ACM Digital Library, with arXiv searched as a complementary source. The database-specific search strategies are presented in Table 2. The protocol for this scoping review was preregistered on the Open Science Framework (OSF) on 20 August 2026. The registration protocol and search strategies can be accessed at https://osf.io/znkfd/overview?view_only=e5714d6a2fba4e8ab39e2f670dbc37d9.

Table 2. The database-specific search strategies.

Database	Search Syntax
Web of Science	TS=("vibe coding" OR "AI-assisted coding" OR "AI-assisted programming" OR "LLM-assisted coding" OR "LLM-assisted programming" OR "AI-assisted software development" OR "LLM-assisted software development") AND TS=("education" OR "course" OR "teaching")
Scopus	TITLE-ABS-KEY ("vibe coding" OR "AI-assisted coding" OR "AI-assisted programming" OR "LLM-assisted coding" OR "LLM-assisted programming" OR "AI-assisted software development" OR "LLM-assisted software development") AND TITLE-ABS-KEY ("education" OR "course" OR "teaching")
IEEE Xplore	("Abstract": "vibe coding" OR "Abstract": "AI-assisted coding" OR "Abstract": "AI-assisted programming" OR "Abstract": "LLM-assisted coding" OR "Abstract": "LLM-assisted programming" OR "Abstract": "AI-assisted software development" OR "Abstract": "LLM-assisted software development") AND ("Abstract": "education" OR "Abstract": "course" OR "Abstract": "teaching")
ACM Digital Library	[[Abstract: "vibe coding"] OR [Abstract: "AI-assisted coding"] OR [Abstract: "AI-assisted programming"] OR [Abstract: "LLM-assisted coding"] OR [Abstract: "LLM-assisted programming"] OR [Abstract: "AI-assisted software development"] OR [Abstract: "LLM-assisted software development"] AND [Abstract: or]] AND [[Abstract: "education"] OR [Abstract: "course"] OR [Abstract: "teaching"]]
arXiv	abstract="vibe coding" OR "AI-assisted coding" OR "AI-assisted programming" OR "LLM-assisted coding" OR "LLM-assisted programming" OR "AI-assisted software development" OR "LLM-assisted software development"; AND abstract="education" OR "course" OR "teaching"

2.3. Literature Identification, Screening, and Selection Process

The study-selection process comprised three stages: identification, screening, and final inclusion. First, during the identification stage, 183 records were retrieved, including 34 from Web of Science, 92 from Scopus, 16 from IEEE Xplore, 18 from the ACM Digital Library, and 23 from arXiv. After duplicate records were removed using *Digital Object Identifier* (DOI) information, 118 unique records remained. Second, the screening stage involved two steps based on inclusion and exclusion criteria presented in Table 3. The titles and abstracts of the 118 records were manually screened. Fifty records were excluded: a total of 10 were not full research articles, such as tutorials or editorials; 3 were review articles; 6 focused on pre-university education; and 31 examined non-computing disciplines in higher education. The remaining 68 articles were retrieved for full-text assessment. Thirty-one articles were excluded because they described tool development without a substantive teaching or learning context. Finally, following the screening and eligibility assessment, 37 studies were included in the review, as summarized in Table A1 and illustrated in Figure 1. The full list of articles is provided in Supplementary Table S1.

Table 3. Inclusion and exclusion criteria.

Inclusion Criteria	
Research articles, such as journal articles, conference articles, or preprints.	
Full text available.	
Published in English.	
Published between 2022 and July 2026.	
Exclusion Criteria	
Non-research articles, such as tutorials, abstracts, book chapters, etc.	
Articles examining non-higher education or non-computing disciplines.	
Articles solely on educational tool development without a teaching or learning context.	

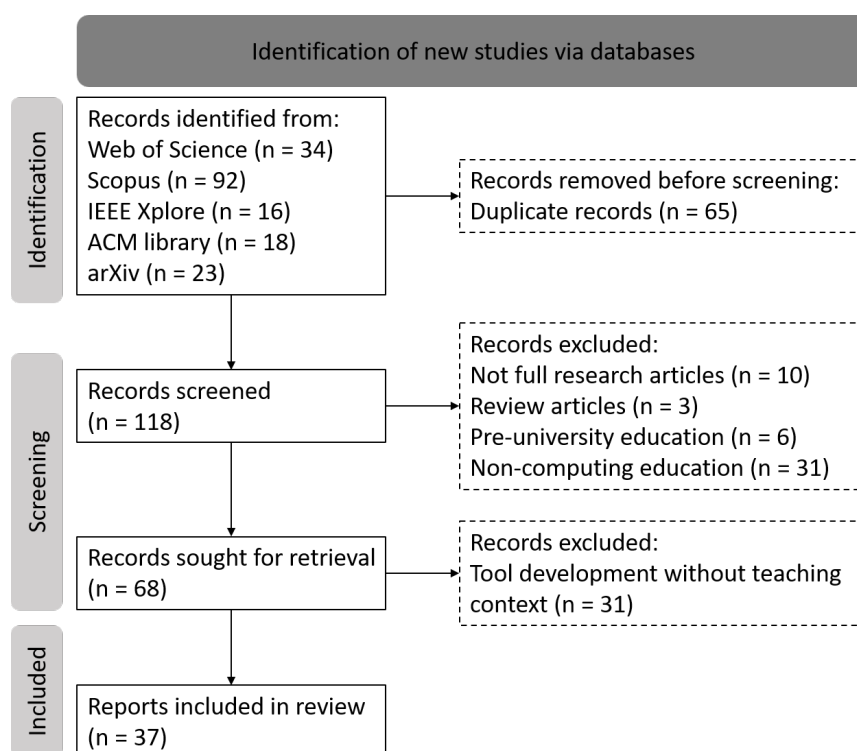


Figure 1. The PRISMA flow diagram for the literature identification, screening, and selection process used in this paper.

2.4. Literature Coding Process

To address the five research questions, we developed a structured data extraction table to enable consistent synthesis across 37 reviewed articles. For each included article, we extracted (i) educational context (education level, course level, participant type, sample size, prior programming experience), (ii) use of AI tools (tool name, model name, access mode, autonomy level, programming language and artifact), (iii) position in the programming cycle (planning and prompting, code generation, verification and evaluation), (iv) pedagogical strategy (role of AI, learning activity, role of instructor), and (v) outcomes (conceptual learning, productivity, participation and access). The definitions and suggested coding values are summarized in Table 4.

Table 4. Data extraction and coding scheme used for 37 included articles.

Item	Definition	Coding Value
Article_ID	Unique identifier	Digital object identifier (DOI) of the article
Year	Publication year	Four digits
Venue_Type	Publication type	Journal; Conference; Preprint
Education_Level	Learner level	Undergraduate; Postgraduate; Professional; Mixed
Course_Level	Course position	Introductory; Intermediate; Advanced; Multiple
Participant_Type	Who was studied	Students; Instructors; Practitioners; No human
Sample_Size	Number of participants	Integer
Prior_Programming	Programming experience	Beginner; Advanced; Mixed
Tool_Name	AI tool	Product or system name; or Unspecified
Model_Name	Model/version	Model and version; or Unspecified
Access_Mode	How AI is accessed	Web chat; Custom web platform; IDE extension; LMS integration; Multiple
Autonomy_Level	Degree of system autonomy	Generative; Advisory; Agentic; Mixed
Programming_Language	Language used	Record all programming languages used
Programming_Artifact	What was produced	Codebase; Full software system; Notebook; Programming exercise; Prototype; Web application
Intent_Prompting	Step in programming cycle	Yes; No
Code_Generation	Step in programming cycle	Yes; No
Verification_Evaluation	Step in programming cycle	Yes; No
Role_of_AI	Role of AI in learning	Tutor; Assistant; Collaborator; Code generator; Coding agent; Feedback provider; Evaluator; Mixed.
Learning_Activity	Task performed	Assistant; Code generator; Collaborator; Feedback provider; Tutor; Mixed
Instructor_Role	Instructor involvement	Assessor; Direct teaching; Facilitator; Monitor; Technical support; Mixed
Data_Sources	Evidence collected	Surveys; Prompt/chat logs; Grades; Code artifacts; Interaction logs; Videos; Pre/post tests; Documents; Hackathon data; Observations
Interaction_Behavior	Prompting, testing	Measured; Self-reported; Not examined
Conceptual_Learning	Knowledge understanding	Measured; Self-reported; Not examined
Productivity	Time, quantity produced	Measured; Self-reported; Not examined
Participation_Access	Accessibility	Measured; Self-reported; Not examined

A single-reviewer coding protocol was used. Before full coding, the initial coding framework was piloted on 10 included studies. Ambiguous or overlapping categories

identified during this pilot were clarified, and the coding definitions and values were revised accordingly. The resulting codebook was then applied consistently to all 37 studies, each of which was read in full before coding. Because the review used a single-reviewer protocol rather than independent parallel coding, inter-rater agreement was not calculated. The use of a predefined codebook, pilot coding, and explicit coding definitions was intended to improve consistency and transparency in the extraction and synthesis process.

2.5. Synthesis of Results

We organized the evidence along two dimensions. The first separated the evidence on the learner side from instructor-side design decisions. The second placed evidence within three stages of AI-assisted programming, including (i) intent formation and prompting, (ii) generation, and (iii) verification and evaluation. The resulting synthesis comprised five domains corresponding to the five RQs: interaction behavior, learning and affective outcomes, task-level instructional strategies, assessment and integrity, and curriculum and industry alignment.

The first descriptive result examines the publication profiles of the included studies. Thirty-three of 37 studies were published in 2025 or 2026; 19 were conference papers, 11 were journal articles, and 7 were preprints, as summarized in Table 5. In terms of study participants, 24 studies involved university students, four involved industry practitioners, and two involved faculty members.

Table 5. Annual distribution of the included articles by publication type.

Year	2022	2023	2024	2025	2026 (Till July)	Total
Journal	1	1		2	7	11
Conference		2	10	7	19	
Preprint				3	4	7
Total	1	1	2	15	18	37

The second descriptive result examines the educational level, course level, and prior programming experience represented in the reviewed studies, as shown in Figure 2. The evidence base is concentrated in undergraduate education: 24 of the 37 studies (64.9%) focused on undergraduate learners. Five studies (13.5%) involved practitioners, two (5.4%) focused on postgraduate learners, one (2.7%) included participants from mixed educational levels, and five (13.5%) did not report the educational level. A similar concentration is evident at the course level. Thirteen studies (35.1%) examined beginner-level courses, compared with five (13.5%) examining advanced courses and four (10.8%) spanning multiple course levels. However, 15 studies (40.5%) did not explicitly report the course level. Prior programming experience was also incompletely documented: 14 studies (37.8%) did not report participants' prior experience. Among the remaining studies, eight involved beginners, eight involved learners with mixed experience levels, four involved participants with advanced experience, and three involved professional programmers. These distributions suggest that the available evidence is weighted toward undergraduate and relatively introductory learning contexts.

The third descriptive result examines the terminology used to characterize AI-supported programming practices. Among the 37 studies, 22 used the broader framing of *AI-assisted coding*, two used *LLM-assisted coding*, and 13 explicitly adopted the term *vibe coding*. These terms reflect different degrees of human involvement and delegation to AI systems. Chang et al. (2025) conceptualizes AI-assisted coding from two dimensions: the level of LLM involvement and the degree of human engineering effort. Within this framework, *vibe coding* is prompt-driven and requires relatively limited human intervention.

AI-assisted coding retains more active human participation in the software design and implementation, with AI augmenting rather than replacing the developer's work. Sapkota et al. (2025) distinguishes vibe coding, where a human developer uses natural language to guide AI models to generate code and tests the output, from agentic coding, where multiple AI agents perform autonomous code development actions within an execution environment.

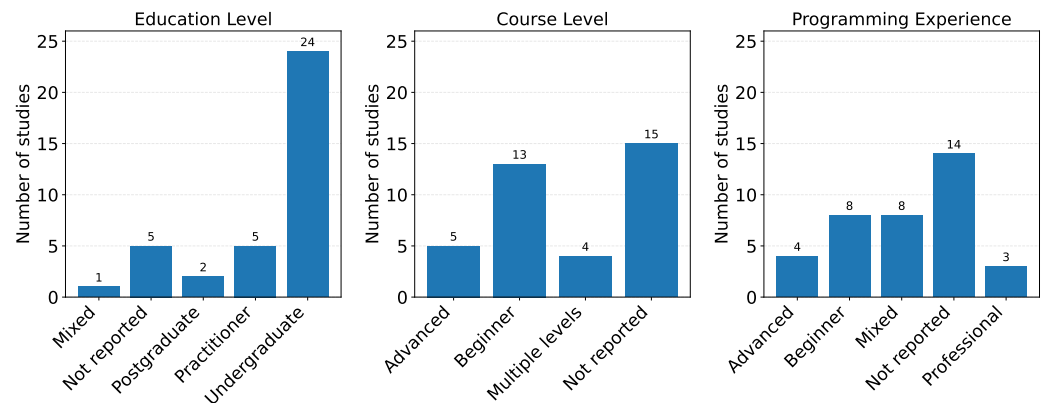


Figure 2. Distribution of education level, course level, and prior programming experience used in the included studies.

The fourth descriptive result examines the empirical evidence used to support the findings of the reviewed studies, as shown in Figure 3. Of the 37 studies, 31 (83.8%) reported at least one identifiable form of empirical evidence. Surveys were the most frequently used evidence source, appearing in 21 of the 31 evidence-reporting studies (67.7%). Prompt or chat logs were used in 10 studies (32.3%), followed by grades in eight (25.8%), code artifacts in six (19.4%), and interaction logs in five (16.1%).

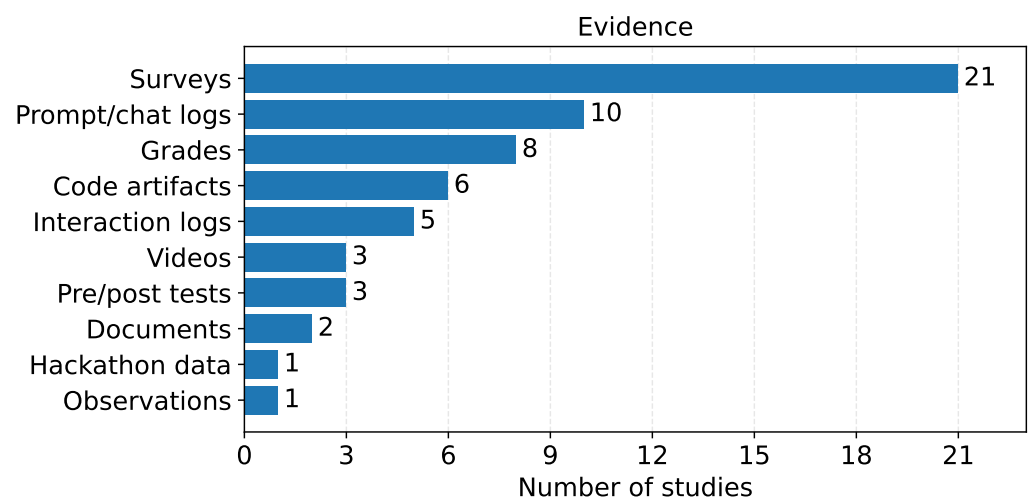


Figure 3. Frequency of evidence types reported in the included studies. Evidence categories are non-mutually exclusive because individual studies may use multiple evidence sources.

3. Results

3.1. Student–AI Interaction Patterns (RQ1)

The reviewed studies suggest that learning outcomes depend less on whether students use AI coding tools than on how they interact with them. As summarized in Table 6, recurring interaction patterns can be organized across four learner contexts: beginner, advanced, mixed-level, and practitioner.

- **Beginner.** Pădurean et al. (2026) found that students typically requested an initial solution, executed it, and when it failed, entered short edit-run loops. This pattern suggests that verification often remains limited to observable program behavior rather than systematic inspection of the generated code. Menon (2025) compared eleven recommended behaviors against an ideal interaction profile and found that students struggled most with preliminary exploration, followed by integrating AI-generated output with their own code and verifying the resulting solution. D. Liu et al. (2026) further showed that greater expertise did not eliminate verification problems but changed their form: beginners displayed a Trust-but-Can't-Verify pattern, whereas more experienced students showed a Boilerplate Blindspot, accepting generated scaffolding without close inspection. Across 732 remediation episodes, Lai and Hsueh (2026) found that 78.0% consisted of pure retesting, while 86.3% of the clarification attempts that did occur appeared in the first round.
- **Advanced.** Rinja et al. (2026) identified significant differences in interaction patterns between the highest-performing and lowest-performing quartiles. High-performing students engaged predominantly in exploratory cycles, whereas lower-performing students repeatedly moved directly from an assignment prompt to an AI-generated solution. Isa et al. (2026) reported that providing detailed specifications for code generation was positively correlated with final grade, while pasting unprocessed console errors was negatively correlated with performance. Rinja et al. (2026) explained this distinction through instrumental and executive help-seeking, arguing that the model tends to respond to the intent expressed in the prompts, such that prompts that delegate responsibility are more likely to produce executable answers than to support deeper reasoning.
- **Multiple levels.** In (Geng et al., 2026), 63.6% of coded interactions involved executing the prototype. Among these interactions, 91.7% tested common cases, only 2.2% tested edge cases, and none involved unit testing. Prompting accounted for 20.6% of all actions, of which 61.0% applied to debugging, while directly engaging with course code or execution logs accounted for 7.4%.
- **Practitioner.** Chou et al. (2026) observed a similar relationship between expertise and interaction behavior outside the formal classroom setting. More experienced coders used more prescriptive prompts and intervened more actively when the generated solution required correction.

Across these contexts, a consistent pattern emerges: stronger performance is associated with more deliberate specification, exploration, verification, and intervention, whereas weaker interaction is characterized by direct solution delegation, repeated execution, and limited inspection of generated code.

Table 6. Findings of student–AI interaction studies (RQ1).

Reference	Interaction Behavior	Evidence
(Pădurean et al., 2026)	Prompt-run-edit loop	Interaction logs, Code artifacts, Surveys (355 students, 13,305 events)
(Menon, 2025)	Exploration, verification	Surveys ($n = 50$)
(D. Liu et al., 2026)	Inquiry, transactional delegation	Interaction logs, Surveys ($n = 41$)
(Lai & Hsueh, 2026)	Pure retesting	Interaction logs, Grades (366 students, 732 episodes)
(Rinja et al., 2026)	Exploration, assignment-to-solution	Prompt/chat logs, Grades (110 students, 19,418 turns)
(Isa et al., 2026)	Code generation, debugging, and explanation	Prompt/chat logs, Grades, Surveys (147 students, 448 logs)
(Geng et al., 2026)	Prototype testing	Videos ($n = 19$), Code artifacts
(Chou et al., 2026)	Prescriptive prompts and corrective intervention	Videos ($n = 20$), Prompt/chat logs

3.2. Learning Outcomes of AI-Assisted Coding Tools (RQ2)

Across the 37 included articles, three broad categories of learning outcomes were reported: conceptual learning ($n = 21$), programming productivity ($n = 13$), and affective outcomes ($n = 9$). These categories were not mutually exclusive, as some studies reported outcomes across multiple dimensions.

The most consistently reported conceptual learning benefits appeared in the beginner-level courses, when AI-assisted coding tools were designed to scaffold students' reasoning rather than provide complete solutions. Hou et al. (2026) randomly assigned 122 undergraduates to an AI-supported intervention group ($n = 62$) or a control group ($n = 60$). Students in the intervention group shifted from trial-and-error behavior toward more deliberate planning. Both the study's planning measure and debugging precision were positively associated with post-test performance. Similarly, He and Li (2025) used programming tests and questionnaires to assess programming proficiency and self-efficacy. Students in the AI-assisted group ($n = 38$) increased their mean proficiency score from 24.34 to 34.34, whereas those in the human pair-programming group ($n = 42$) showed no comparable improvement. Kerimbayev et al. (2026) reported statistically significant gains in academic performance and increased intrinsic motivation among students in the experimental group.

A second reported outcome was increased programming productivity. Kuser and Szabo (2025) assessed students' confidence and found significant self-reported improvements in productivity. However, Tadimalla et al. (2025) reported that although 76% of participants considered AI-assisted coding tools beneficial, 51.2% reported no improvement in their understanding of loops and arrays, compared with 41.5% who perceived an improvement. Lower confidence was particularly evident among students who believed that they had become overly dependent on AI-assisted coding tools.

The third group comprised affective and access-related outcomes, particularly students' ability to translate ideas into functional prototypes. In Gama et al. (2025), 31 students with beginner-level and mixed technical experience transformed their ideas into working prototypes within a single day. The study reported that cross-disciplinary participation and diversity of backgrounds supported authentic hackathon learning experiences. In a multi-national hackathon organized by Chen et al. (2026), participants gave the AI-supported approach a mean helpfulness rating of 4.68 on a five-point Likert scale. Leon van Bokhorst (2026) found that the accessibility gains were largest among participants with the least technical experience.

3.3. Task-Level Instructional Strategies Using AI-Assisted Coding Tools (RQ3)

The instructional strategies reported in the included articles can be grouped into beginner-level and advanced-level course contexts. Table 7 organizes these strategies across three stages of AI-assisted programming, including (i) prompting and planning, (ii) code generation, and (iii) verification and evaluation.

Table 7. Task-level strategies reported in the included studies (RQ3).

Strategy	Programming Stage	Evidence
Planning with hints	Prompting, planning	Interaction logs, Grades, Surveys
Adaptive scaffolding	Code generation	Surveys, Prompt/chat logs, Pre/post tests, Code artifacts
Dual-mode assignment	All stages	Interaction logs, Surveys
Process visibility	Verification, evaluation	Surveys, Prompt/chat logs, Code artifacts

3.3.1. Beginner-Level Courses

Directing AI support toward planning rather than immediate code generation was an effective task-level instructional strategy. Phung et al. (2025) provided 102 students

with planning, debugging, and optimization hints and found that planning hints attracted the greatest engagement and were the only hint type consistently associated with higher performance. Although students requested debugging assistance more frequently as task difficulty increased, their demand for planning support did not rise accordingly. [Hou et al. \(2026\)](#) implemented a similar approach through proactive prompts that encouraged students to plan, execute their code, and compare the output with their expectations rather than using AI primarily as a code-generation tool.

Adaptive scaffolding that withholds complete solutions while prompting students to reason can also benefit beginner programmers. [Tran-Le et al. \(2026\)](#) implemented a reasoning-first prompting approach in which the AI provided guidance without immediately revealing a complete solution. Beginners demonstrated improvement on unassisted post-tests, whereas advanced students benefited mainly through reduced effort. Similarly, [Adhikari et al. \(2024\)](#) found that explanation features were valued most by participants with the least programming experience and least by those with the greatest expertise.

A dual-mode assignment in which students first attempt a task independently and then complete it with AI assistance may help balance programming comprehension with the development of AI literacy. [Liang and Chen \(2026\)](#) asked students to write code independently before completing the same task in collaboration with an AI-assisted coding tool. This design exposed the trade-off between developing an independent understanding of code and learning how to use AI-assisted coding tools effectively. Based on observed student behaviors, [D. Liu et al. \(2026\)](#) proposed a comparable approach involving planned fading, AI-free subtasks, and a compulsory phase where students critique the AI-generated output.

3.3.2. Advanced-Level Courses

For advanced students, adaptive scaffolding and the gradual withdrawal of support can strengthen the ability to collaborate productively with AI-assisted coding tools. [Bai et al. \(2026\)](#) implemented a scaffolded sequence comprising the establishment of shared understanding, continuous diagnosis of learner needs, calibrated assistance, and the gradual fading of support. The intervention demonstrated the pedagogical value of structured scaffolding for developing prompt-engineering proficiency and supporting student–AI collaboration during complex problem-solving activities.

Process-oriented instructional designs can also make AI-assisted programming activity more visible and provide opportunities for targeted feedback. Studies using process telemetry [Al-Khalifa \(2026\)](#) offered design rationales and reported favorable student perceptions, although evidence of their effects on learning outcomes remained limited. At the code-transformation level, [Khairnar et al. \(2025\)](#) found that LLMs were effective at localized refactoring tasks, such as improving identifier names and extracting methods, but rarely recommended broader structural changes, such as reallocating functionality across classes.

3.4. Assessment and Integrity (RQ4)

Assessment design emerged as a central mechanism for shaping how students engage with AI-assisted coding tools. Among the reviewed articles, studies that examined instructional design in depth consistently connected the use of generative AI with changes to assessment. This is consistent with the broader principle that assessment strongly influences learning behavior. Rather than evaluating only the final program, these studies emphasized evidence of process, reasoning, and critical engagement with AI-generated output.

Process-oriented grading encouraged students' attention to shift from producing functional code toward conceptual understanding and reflective practice. [Al-Khalifa \(2026\)](#) allocated 20% of the grade to code output, without assessing coding style, 30% to the prompt log, and the remaining marks to critical reflection. By explicitly rewarding conceptual

engagement, the assessment encouraged students to explain and evaluate their use of AI rather than focus solely on task completion. Participants rated engagement, conceptual learning, and assessment fairness between 4.4 and 4.6 on a five-point scale. However, the study also acknowledged an important validity concern: students may have used LLMs to produce the reflections that were treated as evidence of conceptual understanding.

Three broad assessment strategies were identified, each working through a different mechanism. The first strategy was to resist trivial generation by allowing AI use while designing sufficiently complex high-stakes tasks. Lubarda et al. (2025) adopted this approach by constructing problems that could not be completed successfully through straightforward code generation alone. The second strategy was to segregate independent and AI-assisted performance. Liang and Chen (2026) assessed AI-free and AI-supported phases of the same programming task separately, thereby distinguishing students' unaided competence from their ability to collaborate with an AI-assisted coding tool. The third strategy was to diversify the evidence used for evaluation. Cao (2025) proposed oral examinations, video-based code explanations, incremental version submissions, learning journals, and context-specific tasks to reduce reliance on the final code artifact.

Students' perceptions of academic integrity were influenced by what assessments rewarded and how AI-assisted coding use was documented. Tadimalla et al. (2025) found that nearly two-thirds of students considered the acceptability of AI use to depend on the specific assignment context. When assessments rewarded transparency about the development process, students tended to regard AI as a legitimate learning aid. Adhikari and Lamichhane (2025) required students to record and submit keystroke data alongside their assignments to document the coding process. Although intended to deter plagiarism, this approach received limited student acceptance. The authors consequently recommended that process data remain under student control and be presented as a resource for learning rather than as instructor-controlled surveillance. Faculty concerns about integrity were substantial, although support for prohibition remained limited. In a survey of 110 faculty members, Rose et al. (2023) found that 73.7% expected generative AI to increase plagiarism, 56.3% anticipated reductions in critical thinking, and 45.5% expected programming skills to decline. Nevertheless, only 2.7% supported banning the technology.

3.5. Curriculum and Professional Alignment (RQ5)

This research question examines how computing curricula respond to AI-assisted coding as it becomes embedded in emerging professional practice. The findings are structured from three perspectives: changes in professional practice, curricular approaches that have been empirically evaluated, and proposed curricular adaptations.

- Changes in professional practice. The adoption of AI-assisted coding tools is shifting professional programming competence from code production toward verification, oversight, and system-level judgment. Based on 57 practitioner videos, Chang et al. (2025) found that AI-assisted coding tools substantially accelerated code generation while shifting the primary bottlenecks toward code review, testing, security evaluation, and architectural reasoning. Similarly, De La Cruz et al. (2025) described a transition toward more strategic forms of human-AI collaboration, accompanied by growing concerns about software security and the potential erosion of programming skills.
- Evaluated curricular approaches. Empirical examinations of curricular approaches indicate that lower barriers to code generation create both opportunities and new instructional challenges. Rapid prototyping can broaden participation, but it may also reduce the practice through which novice programmers traditionally develop foundational expertise. Curricula therefore need to address not only AI-enabled productivity but also recurring inefficiencies and failure modes in AI-assisted workflows.

Chou et al. (2026) found that more than 20% of session time was spent waiting for model responses and that students frequently issued redundant prompts. Starace (2026) translated common AI-generated failure modes into an instructional case based on ten recurring bug patterns grouped into four broader categories.

- Suggested changes. Proposed curricular changes emphasize competencies in specification, verification, and structured human–AI collaboration. Liao (2026) proposed a five-module sequence covering model behavior and red-teaming, prompt and context engineering, vibe coding through a bug-hunting verification laboratory, agentic multi-role workflows, and specification-driven development. Student capstones were evaluated through four artifacts: a formal specification, a verification suite, an interaction log, and a working prototype. This design treats the specification, development process, verification evidence, and human–AI interaction as important learning outcomes rather than evaluating only the completed application. Chang et al. (2025) recommended that computing curricula should place greater emphasis on reviewing, testing, securing, and reasoning about AI-generated software.

4. Discussion

4.1. Summary of Key Findings Across the Five RQs

Regarding RQ1 on student–AI interaction, the reviewed articles indicate that students' use of AI-assisted coding tools is often reactive rather than strategically planned (Pădurean et al., 2026). Across multiple studies, students relied heavily on debugging prompts, tested mainly common cases, and engaged in short edit-run cycles (Geng et al., 2026; Lai & Hsueh, 2026; Menon, 2025). Interaction patterns varied with prior competence and were associated with course performance. Higher-performing students generally demonstrated more exploratory, specification-rich, and evaluative behaviors, whereas lower-performing students were more likely to delegate tasks directly to the AI-assisted coding tools (D. Liu et al., 2026; Rinja et al., 2026).

Regarding RQ2 on learning outcomes, the evidence was more consistent for engagement, productivity, and task completion than for deeper conceptual development. Several studies reported improvements in conceptual learning when AI support was structured around reasoning, planning, and guided debugging rather than direct solution generation (He & Li, 2025; Hou et al., 2026; Kerimbayev et al., 2026). Other studies reported gains in perceived productivity, participation, accessibility, motivation, or reduced barriers to prototyping (Chen et al., 2026; Gama et al., 2025; Kusper & Szabo, 2025; Leon van Bokhorst, 2026).

Regarding RQ3 on task-level instructional strategies, the reviewed studies identified several approaches for structuring productive interaction with AI-assisted coding tools. Planning with hints encourages students to formulate an approach rather than immediately requesting complete code. Adaptive scaffolding provides calibrated assistance while withholding full solutions when appropriate. Dual-mode assignments separate independent and AI-assisted phases so that students experience both AI-free problem solving and AI-supported work. Process-visibility strategies make the interaction process visible and verifiable. Open-ended AI assistance tended to favor students with stronger prior knowledge, whereas more constrained and explanatory configurations appeared particularly helpful for beginners (Adhikari et al., 2024; Hou et al., 2026; Liang & Chen, 2026; D. Liu et al., 2026; Phung et al., 2025; Tran-Le et al., 2026). Effective instructional design may therefore require support to be calibrated to learner expertise and gradually withdrawn as students develop greater evaluative independence (Al-Khalifa, 2026; Bai et al., 2026).

Regarding RQ4 on assessment and academic integrity, the reviewed articles show that assessment emerged as a central mechanism through which educators can shape students' use of AI-assisted coding tools. Three broad strategies were identified (Cao, 2025; Liang &

(Chen, 2026; Lubarda et al., 2025): resist trivial generation by designing tasks that cannot be solved adequately through straightforward AI output; segregate independent and AI-assisted phases to distinguish unaided competence from AI-supported performance; and diversify assessment evidence through oral explanations, process records, incremental submissions, or reflective artifacts. Process-oriented assessment can make students' reasoning and interaction with AI more visible, but it also introduces new validity concerns when reflective or explanatory artifacts can themselves be AI-generated. When process transparency and critical reflection were explicitly rewarded, students were more likely to regard an AI-assisted coding tool as a legitimate learning resource (Al-Khalifa, 2026).

Regarding RQ5 on curriculum and professional alignment, practitioner studies and curriculum proposals converge on an emerging shift in programming competence. As AI-assisted tools accelerate code production, greater emphasis is placed on reviewing generated output, testing, security analysis, debugging, and system-level reasoning (Chang et al., 2025; De La Cruz et al., 2025). Session-level analyses also reveal repeated prompting, response delays, and iterative probing of stochastic outputs (Chou et al., 2026). These findings imply that computing curricula should extend beyond prompt construction and code generation to develop students' capabilities in specification, verification, architectural reasoning, security, and responsible oversight.

4.2. Limitations

This scoping review paper has several limitations. First, the object of study is evolving rapidly and does not yet have a stable or consistently applied definition. The term "vibe coding" is used to describe different tools, levels of AI autonomy, prompting practices, instructional constraints, and learning environments. This conceptual variation complicates comparisons across studies and limits the precision with which findings can be generalized.

Second, the evidence base is highly time-sensitive. Seven of the included sources were preprints and had not necessarily undergone peer review at the time of the search. Given the rapid development of AI-assisted coding tools and educational practices, the composition and strength of the evidence base may change as these studies are revised or formally published.

Third, the included studies were heterogeneous in their research designs, participant populations, outcome measures, and educational contexts. Many relied on self-reported perceptions, short interventions, or observational data. These differences prevented direct comparison of effect sizes and limited the extent to which causal conclusions could be drawn.

Finally, the review focused on English-language journal articles, conference papers, and selected preprints published between 2022 and July 2026. Relevant studies published in other languages, indexed outside the selected sources, or reported in other publication formats may therefore have been omitted.

5. Conclusions

This scoping review synthesized 37 studies examining vibe coding and related AI-assisted programming practices in university computing education. Across the reviewed literature, students commonly interacted with AI-assisted coding tools through debugging-oriented prompts, common-case testing, and short edit-run cycles. The reviewed evidence indicates that AI-assisted coding can support conceptual learning, productivity, participation, and motivation under particular instructional conditions. The review further identifies a set of emerging instructional and assessment strategies for university computing education. This includes planning-focused prompts, adaptive scaffolding, dual-mode assignment, and transparent interaction logs. The reviewed studies emphasize assessment designs that

move beyond final code artifacts toward combinations of unaided performance, oral or reflective explanation, interaction records, and other forms of process evidence.

Overall, the findings suggest that the increasing capability of AI-assisted coding tools requires a corresponding shift in how programming competence is conceptualized. Future research should evaluate these instructional designs through larger samples, comparative studies, and authentic programming tasks. Such evidence is necessary to determine when vibe coding supports the development of programming expertise rather than merely accelerating the production of functional software.

Supplementary Materials: The following supporting information can be downloaded at <https://www.mdpi.com/article/10.3390/aieduc2040034/s1>. Table S1: List of articles included in review. File S1: The PRISMAScR Checklist.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset available on request from the author.

Acknowledgments: During the preparation of this manuscript/study, the author used [GPT-5], a large language model developed by OpenAI (San Francisco, CA, USA) for the purpose of improving the clarity of the writing. The author has reviewed and edited the output and takes full responsibility for the content of this publication.

Conflicts of Interest: The author declares no conflicts of interest.

Appendix A. Overview of Included Articles

Table A1. The 37 included studies and their brief descriptions.

Reference	Description
(Gama et al., 2025)	Reports an educational hackathon at a public university studying how beginner and mixed-experience undergraduates engage with vibe coding, using observations, an exit survey, and semi-structured interviews.
(He & Li, 2025)	Uses a quasi-experimental design with undergraduates in a C programming course to compare a hierarchical LLM-based AI-assisted pair programming approach against human-human pair programming on performance and self-efficacy.
(Hou et al., 2026)	Conducts a controlled trial with computer science undergraduates in a Python course, pairing a Jupyter-based autonomous AI agent that triggers process-oriented prompts with fine-grained log analysis to study metacognitive regulation and self-regulated learning.
(Chou et al., 2026)	Studies vibe-coding videos, including live-streamed coding sessions and opinion videos, to characterize how practitioners engage in vibe coding.
(Chen et al., 2026)	Investigates a month-long international online vibe coding hackathon with three difficulty tracks, using a mixed-methods design to study how participants across skill levels engage with vibe coding.
(Pasalkar et al., 2025)	Presents the design and development of an AI coding support system offering structured hints, debugging assistance, conceptual explanations, and gamification features.
(Adhikari & Lamichhane, 2025)	Proposes quantitative summaries and interactive visualizations of students' coding process, such as execution history, AI usage, copy-paste events, and code-evolution playback, to support reflection and academic integrity.
(Liang & Chen, 2026)	Introduces a dual-mode assignment pattern in which students first complete an independent self-coded solution and then repeat the same task with an AI coding assistant.
(Chang et al., 2025)	Conducts a qualitative analysis of curated YouTube videos of industry practitioners to characterize how LLM coding tools are used in professional practice and draw implications for software engineering education.

Table A1. Cont.

Reference	Description
(Bai et al., 2026)	Reports a two-study investigation, applying a three-phase scaffolding strategy to develop postgraduate students' prompt engineering skills in an AI-assisted coding and computational thinking course.
(Cao, 2025)	Reviews the literature to examine how AI-generated content as an auxiliary teaching tool reshapes teaching objectives, content delivery, instructional methods, and evaluation in university programming education.
(Geng et al., 2026)	Uses qualitative coding of screen recordings and artifacts to study how introductory and advanced students interact with a vibe coding platform to build a web application.
(Paludo & Montresor, 2024)	Describes the design of the reflective AI programming Lab, where student groups solve programming tasks using exclusively AI-generated code to foster metacognition and reflective learning.
(Isa et al., 2026)	Analyzes students' LLM chat logs across two offerings of a senior-level web programming course via open coding to identify prompt-response pair types and relate interaction patterns to academic performance.
(Al-Khalifa, 2026)	Describes implementing a vibe coding pedagogical approach in a senior undergraduate natural language processing course, using LLMs for code generation with reflection-based assessment, and analyzes end-of-course feedback.
(Lai & Hsueh, 2026)	Analyzes students' remediation episodes to examine how learners move through the remediation branch of an LLM-assisted programming course and whether clarification interactions relate to subsequent coding-process efficiency.
(Tran-Le et al., 2026)	Presents an LLM-assisted platform applying a cognitive-forcing strategy, and pilots it with beginner and advanced college programmers under assisted and unassisted conditions to examine incidental learning.
(Tsai et al., 2022)	Develops an AI-assisted programming module and compares learning expectancy, motivation, and effectiveness across multi-ethnic groups.
(Pădurean et al., 2026)	Examines how natural-language prompting and manual code editing are combined and how their usage varies with task complexity and student ability.
(Rose et al., 2023)	Surveys professors to understand their perspectives on generative AI Chatbots and SQL Query Builder and how they perceive these tools will affect academia.
(Adhikari et al., 2024)	Implements real-time AI-generated explanations of code and compiler error messages within the process feedback platform and surveys students across two institutions on their engagement.
(Kerimbayev et al., 2026)	Proposes and experimentally tests the MicroVibe Learning model, which integrates microlearning and vibe coding with generative AI support, comparing an experimental group of first-year students against a traditional lecture-practical format.
(Zigler, 2026)	Proposes a three-phase context-engineering preparation methodology for agentic coding and reports its application during a competitive hackathon.
(Bussari, 2025)	Combines retrieval-augmented generation, multimodal AI services, and explainability modules with drag-and-drop design tools.
(Starace, 2026)	Presents a teaching case set in a software company, where students analyze and classify bug patterns in LLM-generated code and evaluate testing, code review, and governance strategies.
(Lubarda et al., 2025)	Investigates the feasibility of permitting generative AI tools on high-stakes assessments in an introductory programming course for engineers, integrating AI-assisted programming and metacognitive instruction alongside a permitted-use project and final exam.
(Phung et al., 2025)	Studies a hint system designed around the metacognitive phases with 102 students in an introductory data science programming course.
(Menon, 2025)	Surveys undergraduates to examine the frequency of recommended AI-support behaviors, such as planning queries and verifying AI-generated output, and proposes low-overhead instructional scaffolds.
(De La Cruz et al., 2025)	Explores IT professionals' lived experiences of human–AI collaboration, shifting professional identities, and security concerns in LLM-augmented software engineering.
(Tadimalla et al., 2025)	Studies an AI Literacy course using pre-task and post-task assessments, a chatbot-assisted programming task on loops and arrays, and a reflection survey to examine student interactions, attitudes, and perceptions of AI tools.

Table A1. Cont.

Reference	Description
(D. Liu et al., 2026)	Uses a constructivist grounded-theory design across an AI-enabled and a human pair-programming Java course section, drawing on interaction logs, concept maps, and interviews to build a model of cognitive scaffolding and offloading.
(Rinja et al., 2026)	Conceptualizes vibe coding as help-seeking and analyzes students' interaction turns to compare top-performing and low-performing students.
(Leon van Bokhorst, 2026)	Presents a practitioner report documenting patterns from vibe coding challenges run across four student cohorts.
(Khairnar et al., 2025)	Integrates LLM-assisted refactoring with structured prompts into a course project on a long-lived open-source project to teach students to identify and address code smells, evaluated through student feedback.
(Sapkota et al., 2025)	Presents a taxonomy that compares vibe coding and agentic coding paradigms across conceptual foundations, execution models, feedback loops, safety mechanisms, debugging strategies, and tool ecosystems.
(Kusper & Szabo, 2025)	Develops educational material for practicing vibe coding within project-based learning, and reports a preliminary analysis of confidence and productivity hypotheses.
(Liao, 2026)	Describes instructional experiments across several courses that inform the design of a new course structured around five modules spanning stochastic engines, prompt/context engineering, vibe coding, agentic coding, and specification-driven development.

References

- Adhikari, B., Dhakal, S., & Jha, A. (2024). Maximizing student engagement in coding education with explanatory AI. In *Frontiers in education conference* (pp. 1–9). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Adhikari, B., & Lamichhane, N. (2025). Coding process visualizations for improved learning and academic integrity in CS1. In *World congress in computer science, computer engineering & applied computing* (Vol. 2261 CCIS, pp. 275–282). Springer. [CrossRef]
- Al-Khalifa, H. (2026). From code-centric to concept-centric: Teaching NLP with LLM-assisted vibe coding. In *International workshop on teaching natural language processing* (pp. 11–18). Association for Computational Linguistics. [CrossRef]
- Bai, S., Yeung, S. S., & Lo, C. K. (2026). Enhancing the effect of AI-assisted learning: The use of scaffolding strategies to develop students' prompt engineering skills. *Interactive Learning Environments*, 1–22. [CrossRef]
- Bakharia, A., & Abdi, S. (2024). Shaping programming and data science education: Insights from GenAI technical book trends. In *IEEE international conference on advanced learning technologies* (pp. 116–120). IEEE. [CrossRef]
- Bussari, S. (2025). Vibe coding: An AI-augmented flow paradigm for software development. In *International conference on computer and applications* (pp. 1–6). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Cao, H. (2025). Experiences and insights gained from AI-assisted programming instruction in higher education. In *International conference on AI-enabled education* (pp. 159–166). Association for Computing Machinery. [CrossRef]
- Chang, H.-F., Shirazi, M., Cao, L., & Mobasser, S. K. (2025). Coding with AI: From a reflection on industrial practices to future computer science and software engineering education. *arXiv*, arXiv:2512.23982. [CrossRef]
- Chen, A. J., Cao, Y., Shao, M., Karri, R., & Shafique, M. (2026). Code for all: Educational applications of the vibe coding hackathon in programming education across all skill levels. *arXiv*, arXiv:2604.22747. [CrossRef]
- Chou, Y.-H., Jiang, B., Chen, Y. W., Weng, M., Jackson, V., Zimmermann, T., & Jones, J. A. (2026). Building software by rolling the dice: A qualitative study of vibe coding. *Proceedings of the ACM on Software Engineering*, 3(FSE), FSE077. [CrossRef]
- De La Cruz, E., Le, H., Meduri, K., Nadella, G. S., & Gonaygunta, H. (2025). Redefining the programmer: Human-AI collaboration, LLMs, and security in modern software engineering. *Computers, Materials and Continua*, 85(2), 3569–3582. [CrossRef]
- Fu, Q., Zhao, Y., Jia, Z., & Zheng, Y. (2025). Large language models (LLMs) in programming learning: The current research state and agenda. *IEEE Transactions on Learning Technologies*, 18, 942–961. [CrossRef]
- Gama, K., Calejario, F., Jackson, V., Nolte, A., Morais, L. A., & Garcia, V. (2025). Can you feel the vibes?: An exploration of novice programmer engagement with vibe coding. *arXiv*, arXiv:2512.02750. [CrossRef]
- Geng, F., Shah, A., Li, H., Mulla, N., Swanson, S., Raj, G. S., Zingaro, D., & Porter, L. (2026). Exploring student-AI interactions in vibe coding. In *Australasian computing education conference* (pp. 45–54). Association for Computing Machinery. [CrossRef]
- GitHub. (2026, July 1). *What is vibe coding?* Available online: <https://github.com> (accessed on 2 August 2026).

- He, H., & Li, X. (2025). A comparative study on the effectiveness of generative AI tools and pair programming in college students' programming learning. In *International conference on distance education and learning* (pp. 224–228). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Hou, X., Liu, Z., Wang, G., & Shi, N. (2026). Bridging awareness and behavior: Decoding implicit metacognitive behaviors in AI-assisted programming via fine-grained log analysis. *Frontiers in Psychology*, 17, 1790196. [CrossRef] [PubMed]
- Isa, H.-Y., Weston, M., Wellyanto, M. R., Karna, I., Talton, J. O., & Kumar, R. (2026). From code generation to conceptual learning: Student use of LLMs in a web programming course. In *Chi conference on human factors in computing systems* (pp. 1–13). Association for Computing Machinery. [CrossRef]
- Kaewsaiha, P., Klaysung, C., & Norasedsophon, S. (2026, March 29–April 1). *Vibe coding and AI-assisted programming in undergraduate computer education: A systematic review of pedagogical benefits and foundational skill RISKS* [Paper presentation]. International Academic Multidisciplinary Research Conference (pp. 33–44), Paris, France. Available online: <http://icbtsproceeding.ssru.ac.th/index.php/ICBTSPARIS2026/article/view/2720> (accessed on 7 August 2026).
- Karpathy, A. (2025, February 2). *There's a new kind of coding i call "vibe coding", where you fully give in to the vibes*. Available online: <https://x.com/karpathy/status/1886192184808149383> (accessed on 2 August 2026).
- Kerimbayev, N., Menlibay, Z., & Shadiev, R. (2026). MicroVibe Learning: Synergy of vibe coding and microlearning in modern programming education. In *Proceedings of SPIE—The international society for optical engineering* (Vol. 14183, p. 141830T). SPIE. [CrossRef]
- Khairnar, A., Rajaju, A., & Gehringer, E. F. (2025). Teaching code refactoring using LLMs. In *Frontiers in education conference* (pp. 1–8). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Kusper, G., & Szabo, C. (2025). Vibe coding in education. In *International conference on emerging elearning technologies and applications* (pp. 506–511). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Küchemann, S., Avila, K. E., Dinc, Y., Hortmann, C., Revenga, N., Ruf, V., Stausberg, N., Steinert, S., Fischer, F., Fischer, M. R., Kasneci, E., Kasneci, G., Kuhr, T., Kutyniok, G., Malone, S., Sailer, M., Schmidt, A., Stadler, M. J., Weller, J., & Kuhn, J. (2025). On opportunities and challenges of large multimodal foundation models in education. *npj Science of Learning*, 10(1), 11. [CrossRef] [PubMed]
- Lai, L.-C., & Hsueh, N.-L. (2026). Help-seeking in LLM-assisted learning: Behavioral pathways and their limited association with subsequent coding process efficiency. *Electronics*, 15(12), 2706. [CrossRef]
- Leon van Bokhorst, K. S. (2026). Vibe coding before the trend. *arXiv*, arXiv:2605.07751. [CrossRef]
- Liang, J., & Chen, X. (2026). Coding Twice: Integrating independent and AI-assisted programming assignments in CS education. In *ACM technical symposium on computer science education* (pp. 1417–1418). Association for Computing Machinery. [CrossRef]
- Liao, W. (2026). Teaching software development in the GenAI era: Experiences and course design. *Journal of Computing Sciences in Colleges*, 41(8), 258–272.
- Liu, D., Fan, G., & Pan, L. (2026). Tool, tutor, or crutch?: A grounded theory of cognitive scaffolding and offloading in AI-assisted programming education. *International Journal of STEM Education*, 13(1), 10. (Correction in 2026, *International Journal of STEM Education*, 13, 14). [CrossRef]
- Liu, J., & Li, S. (2024). Toward artificial intelligence-human paired programming: A review of the educational applications and research on artificial intelligence code-generation tools. *Journal of Educational Computing Research*, 62(5), 1165–1195. [CrossRef]
- Lubarda, M. V., Lubarda, V. A., Schurgers, C., Phan, A., Baghdadchi, S., & Qi, H. (2025). Permitting GenAI use on assessments in an introductory programming course. In *Frontiers in education conference* (pp. 1–5). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Menon, P. (2025). Quantifying shortfalls in students' AI-supported programming practices. *Issues in Information Systems*, 26(1), 458–471. [CrossRef]
- Ng, D. T. K., Chan, E. K. C., & Lo, C. K. (2025). Opportunities, challenges and school strategies for integrating generative AI in education. *Computers and Education: Artificial Intelligence*, 8, 100373. [CrossRef]
- Paludo, G., & Montresor, A. (2024). Fostering metacognitive skills in programming: Leveraging AI to reflect on code. In *International workshop on artificial intelligent systems in education*. CEUR-WS. Available online: https://ceur-ws.org/Vol-3879/AIXEDU2024_paper_13.pdf (accessed on 7 August 2026).
- Pasalkar, J., Gajbhiye, T., Ovhal, O., & Hanwate, S. (2025). CodeBuddy—An AI-powered code learning buddy. In *IEEE international conference on information and communication technology* (pp. 1–5). Institute of Electrical and Electronics Engineers Inc. [CrossRef]
- Pădurean, V.-A., Gotovos, A., Ghosh, A., Denny, P., Leinonen, J., Luxton-Reilly, A., Prather, J., & Singla, A. (2026). Interleaving natural language prompting with code editing for solving programming tasks with generative AI models. In *ACM conference on innovation and technology in computer science education* (pp. 273–279). Association for Computing Machinery. [CrossRef]
- Peters, M. D. J., Marnie, C., Tricco, A. C., Pollock, D., Munn, Z., Alexander, L., McInerney, P., Godfrey, C. M., & Khalil, H. (2020). Updated methodological guidance for the conduct of scoping reviews. *JBI Evidence Synthesis*, 18(10), 2119–2126. [CrossRef] [PubMed]

- Phung, T., Choi, H., Wu, M., Singla, A., & Brooks, C. (2025). Plan more, debug less: Applying metacognitive theory to AI-assisted programming education. In *International conference on artificial intelligence in education* (p. 17). Springer. [\[CrossRef\]](#)
- Rinja, D., Oliveira, E. A., Lopez-Pernas, S., Saqr, M., Specht, M., & Misiejuk, K. (2026). Unpacking vibe coding: Help-seeking processes in student-AI interactions while programming. *arXiv*, arXiv:2604.27134. [\[CrossRef\]](#)
- Rose, K., Massey, V., Marshall, B., & Cardon, P. (2023). IS professors' perspectives on AI-assisted programming. *Issues in Information Systems*, 24(2), 178–190. [\[CrossRef\]](#)
- Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic AI. *arXiv*, arXiv:2505.19443. [\[CrossRef\]](#)
- Starace, L. L. L. (2026). When AI codes, who tests? A teaching case on risk, quality, and governance in LLM-assisted software development. *Journal of Systems and Software*, 241, 112989. [\[CrossRef\]](#)
- Tadimalla, S. Y., Maher, M. L., Najjar, N., & Simon, C. (2025). The impact of AI tools on the student programming experience: Interactions, attitudes & perceptions. In *Frontiers in education conference* (pp. 1–10). Institute of Electrical and Electronics Engineers Inc. [\[CrossRef\]](#)
- Tian, J. (2026). Vision-language models in teaching and learning: A systematic literature review. *Education Sciences*, 16(1), 123. [\[CrossRef\]](#)
- Tran-Le, B.-T., Thomas, P., Stiffler, N. M., & Nguyen, T. N. (2026). How does LLM-powered coding assistance shape incidental learning? Exploring cognitive forcing strategies in programming education. In *AAAI conference on artificial intelligence* (Vol. 40, pp. 40880–40888). Association for the Advancement of Artificial Intelligence. [\[CrossRef\]](#)
- Tricco, A. C., Lillie, E., Zarin, W., O'Brien, K. K., Colquhoun, H., Levac, D., Moher, D., Peters, M. D. J., Horsley, T., Weeks, L., Hempel, S., & Akl, E. A. (2018). PRISMA extension for scoping reviews (PRISMA-ScR): Checklist and explanation. *Annals of Internal Medicine*, 169(7), 467–473. [\[CrossRef\]](#) [\[PubMed\]](#)
- Tsai, C.-W., Ma, Y.-W., Chang, Y.-C., & Lai, Y.-H. (2022). Integrating multiculturalism into artificial intelligence-assisted programming lessons: Examining inter-ethnicity differences in learning expectancy, motivation, and effectiveness. *Frontiers in Psychology*, 13, 868698. [\[CrossRef\]](#) [\[PubMed\]](#)
- Zigler, A. (2026). Mise en place for agentic coding: Deliberate preparation as context engineering methodology. *arXiv*, arXiv:2605.05400. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.