Journal of Contemporary Educational Research, 2026, Volume 10, Issue 8
http://ojs.bbwpublisher.com/index.php/JCER
Online ISSN: 2208-8474
Print ISSN: 2208-8466

Mechanisms and Reform Pathways for Integrating Generative AI into University Programming Education: A Task–Evidence–Assessment Chain Analysis

Haiping Zeng

School of Advanced Manufacturing Engineering, Guangxi Science & Technology Normal University, Laibin 546199, Guangxi, China

Copyright: © 2026 Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0), permitting distribution and reproduction in any medium, provided the original work is cited.

Abstract: The integration of generative artificial intelligence (GenAI) into university programming education creates opportunities for timely feedback and differentiated support, but it may also intensify cognitive outsourcing, obscure learning processes, and weaken assessment validity. Using policy-text analysis, purposive literature retrieval, and comparative case analysis, this study reviews authoritative guidance and six empirical studies of GenAI-supported programming learning. The available evidence—predominantly small-sample and short-duration—suggests that GenAI can alter debugging behavior, improve selected code-quality and self-efficacy indicators, and enrich learning support; however, access alone does not consistently improve performance or transfer. Educational effects are conditioned by task structure, cognitive engagement, instructional scaffolding, and assessment design. To explain these conditions, the paper develops a coordinated task–evidence–assessment chain: staged tasks define AI-use boundaries; version histories, tests, and selected dialogues create process evidence; and authentic assessment combines products, process records, oral defense, and transfer tasks. It further proposes outcome redesign, progressive authorization, structured prompting, traceable learning portfolios, assessment reform, and multilevel governance. The framework is a theory-informed design proposition that requires validation in authentic classroom settings.

Keywords: Generative artificial intelligence; Programming education; Teaching reform; Human–AI collaboration; Process assessment

Online publication: August 31, 2026

1. Introduction

Generative artificial intelligence (GenAI) can generate, explain, and revise code from natural-language prompts. It can intervene not only in implementation but also in requirements analysis, algorithm design, test construction, debugging, and reflection. In introductory programming, Python, object-oriented programming,

data structures, and algorithms, this capability challenges the conventional sequence of teacher explanation, student imitation, independent coding, and product-based grading.

China's *Education Powerhouse Development Plan (2024–2035)* calls for artificial intelligence to support educational transformation, while the Ministry of Education's "AI + Higher Education" initiative emphasizes innovation in talent development and teaching methods [1,2]. showing that programming has become a concrete setting for AI-enabled reform.

A 2025 survey of 1,041 full-time UK undergraduates found that 92% had used some form of AI and 88% had used GenAI in assessed work [3]. These findings cannot be directly generalized to China, but they suggest that avoidance alone is no longer a viable institutional response. In programming education, the completion of an assignment can no longer be treated as sufficient evidence of individual mastery.

Previous studies have examined the educational opportunities and risks of large language models [4,5], the refinement of AI-enabled educational scenarios [6], and the challenge of protecting teacher and student agency [7]. This paper asks a more specific question: under what objectives, task structures, and assessment constraints can AI support rather than replace computational thinking?

2. Research design and evidence base

This study combines policy-text analysis, purposive literature retrieval, and comparative case analysis. Searches and source verification were completed on 30 July 2026 using Chinese and English terms related to GenAI, programming education, introductory programming, experiments, and higher education. Journal websites, DOI records, government pages, and international-organization sources were checked. Empirical studies were included when their participants, samples, or data sources were explicit; product publicity, unverifiable sources, and studies limited to testing model answers were excluded. Six empirical studies were retained as the evidence base. Because the search was purposive rather than systematic, the resulting sample should not be treated as exhaustive.

Two university quasi-experiments provide the central evidence. A study of 82 students found that ChatGPT changed debugging and feedback-reading behaviors, but the between-group difference in programming performance was not statistically significant [8]. A 30-student Python study found that learners receiving structured prompts formulated more complex questions and obtained more precise feedback [9]. Given their limited samples and duration, these findings indicate that instructional scaffolding may condition educational effects; they do not establish a universal treatment effect.

A study of 38 novice Java students found fewer violations of several coding conventions in the ChatGPT-assisted group and lower median cyclomatic and cognitive complexity, with statistically significant between-group differences; because the study was limited to static analysis of a small sample, replication is needed [10]. An exploratory study involving 20 students and five faculty members reported generally positive evaluations of basic explanations and immediate support, while also identifying inconsistent responses, insufficient depth, and a continuing need for instructor guidance on complex problems [11]. A quasi-experiment with 55 undergraduates found stronger performance on selected code-quality and self-efficacy indicators in the GenAI group, but weaker results on several transfer measures; process-mining evidence further associated cognitive outsourcing and over-reliance with lower performance [12]. A self-report survey of 298 novice programmers documented patterns of use and student experience [13].

This is a theory-building and case-supported teaching-reform study. It does not claim to report an intervention conducted in the authors' own courses. Its purpose is to propose an explanatory framework and implementable design that can be tested in subsequent classroom research.

3. Theoretical foundations and analytical framework

When GenAI enters programming education, cognitive work is redistributed between the learner and the model. To prevent the proposed framework from becoming a list of disconnected techniques, this paper uses cognitive apprenticeship, self-regulated learning, and authentic assessment to explain task design, process visibility, and capability judgment. These perspectives are integrated into a cycle of task design, process evidence, authentic assessment, and feedback adjustment.

3.1. Cognitive apprenticeship: Scaffolding and fading in the task chain

Cognitive apprenticeship makes expert thinking visible through modeling, coaching, scaffolding, articulation, reflection, and exploration [14]. On this basis, the present study argues that complete AI-generated solutions may turn support into substitution for novices. The task chain should first require students to articulate problem analysis, algorithmic ideas, and output predictions; AI may then provide layered hints, error explanations, or testing suggestions. Support should gradually fade as competence develops.

3.2. Self-regulated learning: A process account of the evidence chain

Zimmerman conceptualizes self-regulated learning as a cycle of forethought, performance, and self-reflection [15]. Mapped to human–AI programming, problem analyses and pseudocode represent planning; commits, tests, and selected dialogues represent performance monitoring; and reasons for accepting, revising, or rejecting AI suggestions represent self-evaluation. The evidence chain therefore makes planning, monitoring, judgment, and reflection visible rather than merely surveilling students.

3.3. Authentic assessment: Validity of the assessment chain

Authentic assessment evaluates capability through tasks, contexts, outcomes, and standards that resemble professional practice [16]. Accordingly, this paper defines AI-era software-development capability as more than producing runnable code: it includes understanding requirements, reviewing generated code, designing tests, explaining technical choices, and accepting responsibility. Products, process portfolios, code walkthroughs, oral defense, and transfer tasks are therefore combined in the assessment chain.

3.4. Integration of the three perspectives

The three perspectives answer different questions. Cognitive apprenticeship explains how AI support should enter and fade. Self-regulated learning identifies which planning, monitoring, and reflection processes should be evidenced. Authentic assessment explains how multiple forms of evidence can support valid capability judgments. Together they yield a closed loop: the task chain regulates cognitive work and AI access; the evidence chain makes learning processes visible; the assessment chain judges capability and adjusts subsequent task difficulty and AI authorization.

4. Challenges arising from GenAI intervention in programming education

4.1. Goal displacement: Code production versus computational thinking

Programming education should develop problem decomposition, abstraction, algorithm design, testing, and transfer. Because GenAI rapidly produces plausible code, "generating a correct program" may displace these learning outcomes. A student who bypasses analysis and design may submit code that passes tests but remain unable to explain control structures, complexity, or how the solution transfers to a new problem.

UNESCO frames student AI competence around a human-centered mindset, AI ethics, AI techniques and applications, and AI system design [17]. These dimensions provide a normative reference for university programming courses, in which code review, output verification, and responsible use should become explicit learning outcomes.

4.2. Cognitive outsourcing and shallow debugging

Errors, hypotheses, failed attempts, and revisions are not simply inefficiencies; they contribute to concept formation. When AI immediately supplies complete code and fixes, students may lose opportunities to locate errors and build mental models. In the 82-student study, the ChatGPT group debugged and read feedback more often but also copied more code, while its performance advantage was not statistically significant [9]. Activity therefore cannot be equated with deep learning.

Large-language-model outputs are probabilistic and may contain plausible logical errors, omitted boundary conditions, insecure practices, or nonexistent dependencies [5,6]. Novices may over-trust fluent explanations. When "it runs" replaces "I can justify it," debugging can deteriorate into repeated copying of error messages and uncritical acceptance of revised code.

4.3. Misalignment between general tools and differentiated learning

Programming classes include complete beginners, students with prior experience, and advanced learners. General-purpose models often return similarly complete answers to all three groups: weak students may not understand them, while advanced students receive insufficient challenge. Moreover, programming knowledge has prerequisite dependencies that open-ended conversation does not automatically respect.

Teachers must decide when AI is permitted, design scaffolds, and validate outputs. The UNESCO teacher framework shows that relevant competence extends beyond tool operation to AI pedagogy, ethical judgment, and professional learning.

4.4. Assessment invalidity and invisible learning processes

Assessment based only on source code, laboratory reports, and final projects loses validity when code, comments, test reports, and reflections can all be generated. If teachers inspect only whether a program runs and whether documentation is polished, they may evaluate model output and editing skill rather than student understanding.

Available survey evidence also cautions that AI-detection tools are unreliable. In a domain where code is highly structured and reusable, detection alone cannot sustain assessment validity. A more defensible design moves from asking whether AI was used to requiring students to explain when, why, and how it was used, supplemented by walkthroughs, oral explanation, transfer tasks, and version-history checks.

4.5. Governance risks

Programming tasks may contain personal information, real datasets, unreleased code, credentials, or third-party intellectual property. Uploading such materials to public models may create privacy, copyright, and security risks. UNESCO guidance emphasizes ethical impact assessment, transparency, and accountability [18]. Universities must translate these principles into course-level rules for permitted use, disclosure, data handling, and responsibility.

5. The task–evidence–assessment chain mechanism

The educational effects of GenAI emerge from the interaction of tools, tasks, learner behavior, and assessment. The proposed mechanism positions AI as a constrained cognitive scaffold rather than an answer generator: the task chain defines cognitive activities and access boundaries; the evidence chain records problem solving; and the assessment chain uses multiple sources to judge capability and adjust later tasks and permissions.

5.1. Task chain: Progressive authorization

The task chain follows problem understanding, solution expression, implementation, testing and debugging, and explanation and transfer. During concept formation, no-AI or limited-AI tasks require independent pseudocode, flowcharts, code fragments, and output predictions. During skill development, AI may explain errors, generate candidate tests, or provide hints. In projects, code assistants may support implementation, but students retain responsibility for architecture, interfaces, security, and verification.

The objective is to preserve productive difficulty, not to increase low-value labor. Mechanical syntax lookup can be delegated while conceptually indispensable reasoning is retained. Time saved should be redirected to algorithm comparison, code review, testing, and explanation.

5.2. Evidence chain: Traceable human–AI learning

The evidence chain includes problem analyses, pseudocode, commits, tests, selected prompts and responses, modification notes, and reflection. Students need not submit every conversation; they should select interactions that materially influenced the solution and explain why suggestions were accepted, modified, or rejected. Repository history shows iteration, automated assessment shows testing and repair, and oral explanation checks understanding.

Such evidence can reduce copy-and-submit behavior and support more specific diagnosis. It also enables value-added assessment by focusing on development from the initial to the final solution rather than only on the polish of the final artifact.

5.3. Assessment chain: Capability-based judgment

The assessment chain combines product, process, and authentic assessment. Product assessment addresses correctness, efficiency, maintainability, and security. Process assessment addresses decomposition, prompting, testing, feedback use, and iteration. Authentic assessment uses timed independent coding, walkthroughs, oral defense, and variant tasks. Students who can explain and verify their work may receive broader AI access; those showing dependency return to stepwise scaffolds.

The 30-student quasi-experiment found that the structured-prompt group formulated more complex

questions and received more precise feedback, whereas unprompted interaction was more often superficial [9]. Prompt quality may therefore serve as candidate process evidence of problem representation, although its assessment validity requires testing across larger samples and different courses.

5.4. Core concepts, propositions, and boundaries

The task chain is a sequence of learning activities organized by the cognitive order of programming problem solving and graded AI access. The evidence chain is a continuous set of materials showing analysis, interaction, iteration, and judgment. The assessment chain uses products, process evidence, defense, and transfer to judge genuine capability and adjust later tasks and permissions.

Four propositions follow for later testing: progressive authorization should preserve cognitive engagement better than unrestricted access or blanket prohibition; process evidence should mediate valid capability judgment; defense and transfer tasks should discourage superficial copying; and stronger coordination among the three chains should make efficiency gains more likely to become transferable capability. These are theoretical propositions, not causal effects established by this paper.

The framework is most applicable to introductory programming, object-oriented programming, data structures, and project courses that generate versions, tests, and oral evidence. Effects may vary with prior knowledge, class size, instructor feedback capacity, platform conditions, and model version. The framework cannot replace professional judgment and is unsuitable when data or code cannot be safely uploaded.

6. Reform pathways

6.1. Dual outcomes: Programming capability and AI literacy

Course outcomes should include problem decomposition and algorithmic expression; implementation, testing, and debugging; reading and reviewing human- or AI-generated code; structured human–AI collaboration; and recognition of hallucination, privacy, copyright, and security risks. Introductory courses should emphasize independent reasoning, while software engineering should emphasize collaboration, architectural trade-offs, and responsibility.

6.2. Red–amber–green authorization

AI permissions should be specified by task. Red tasks prohibit GenAI for diagnostic work, timed coding, and individual verification. Amber tasks permit conceptual explanation, error analysis, and test generation but prohibit complete solutions. Green tasks permit extensive assistant use while emphasizing requirements, design choices, review, integration, and reflection. Every assignment should state its permission level, disclosure requirements, and prohibited practices.

Authorization should develop with competence: novices think before seeking help; intermediate learners collaborate and verify; advanced learners conduct engineering review of AI outputs. Permission never removes responsibility for correctness, security, or intellectual property.

6.3. Structured prompting as a problem-solving scaffold

Prompts should support learning rather than answer production. For example: “Do not provide complete code; ask questions that help me locate the error; give one level of hint at a time; require me to predict the revised output.” Another prompt might request comparison of two algorithms and three boundary conditions

that the student must independently test. Such templates should gradually fade.

Error-review activities can ask students to identify logical defects, security vulnerabilities, redundancy, or incorrect complexity claims in AI-generated code. This uses generative capacity while keeping judgment and explanation with the learner.

6.4. Traceable learning portfolios

Code repositories, online judges, and learning platforms can support lightweight portfolios containing task analyses, key versions, test results, representative dialogues, and reflections. Minimum submission requirements prevent documentation from becoming a ritual burden. Instructors can sample rather than inspect every record and investigate signals such as abrupt version jumps, unexplained code, or mismatches between tests and implementation.

Data minimization should apply: remove personal information and credentials, and do not upload restricted code or sensitive data. Institutions should provide approved or locally deployed tools where feasible.

6.5. Product + process + defense + transfer assessment

A four-part structure is recommended. Products demonstrate functionality and engineering quality; portfolios show problem solving and collaboration; oral defense checks understanding and decisions; and transfer tasks require students to modify requirements, fix a new defect, or explain unfamiliar code under time constraints. A project course might provisionally use 35% product, 25% process, 20% defense, and 20% transfer, whereas introductory courses may assign more weight to independent no-AI tasks. These weights are design suggestions, not empirically optimal parameters.

Rubrics should replace “Did the student use AI?” with “Did the student use AI appropriately?” Strong performance includes defining the problem, constraining the model, verifying outputs, explaining human revisions, and transferring the method. Weak performance includes direct copying, inability to explain, missing tests, and concealed use.

6.6. Faculty capacity and multilevel governance

Faculty development should focus on task redesign, prompt scaffolds, code review, and interpretation of process evidence. Course teams can build graded task banks, error-case libraries, and common rubrics. Institutions should recognize the workload involved and provide technical and legal support.

At course level, AI rules should be published; at program level, authorization should be coordinated across the curriculum; and at institutional level, tool approval, data protection, ethical review, and appeals should be established. Ministry initiatives already identify educational agents, educational graphs, and digital practice teaching as important directions. Small pilots should precede course-cluster expansion, and reports should include null and unsuccessful outcomes as well as positive results.

7. Conclusion

GenAI lowers the cost of code production but not the importance of problem solving, logical verification, and responsible judgment. Programming education should move from merely writing code to defining problems, reviewing code, validating systems, and accepting responsibility in AI-supported work. The task–evidence–

assessment chain preserves cognitive engagement through progressive authorization, restores learning visibility through process evidence, and directs collaboration through authentic assessment. Its proposed weights, authorization thresholds, and effects require validation in real courses.

Current evidence remains preliminary. The 82-student study found no significant between-group difference in programming performance [8], while the 30-student study mainly identified differences in interaction behavior, feedback quality, and perceptions [9]. These findings suggest that scaffolding may be an important condition, but they do not prove universal effectiveness. Future multi-course and longitudinal studies should use independent programming tests, transfer tasks, process logs, and interviews, and should report effect sizes, model versions, and prompting conditions.

Methodologically, the framework also supplies a falsifiable research agenda. Subsequent studies can compare authorization sequences, examine whether process evidence mediates performance and transfer, and test whether multimodal assessment moderates the relationship between AI assistance and valid capability judgments. Such designs would distinguish immediate productivity gains from durable learning, identify boundary conditions across learner levels and course types, and convert the three-chain model from a normative proposal into an empirically refinable teaching theory.

Disclosure statement

The author declares no conflict of interest.

References

[1] Central Committee of the Communist Party of China, State Council, 2025, Outline of the Plan for Building a Powerful Nation in Education (2024–2035), viewed July 30, 2026, https://www.moe.gov.cn/jyb_xwfb/gzdt_gzdt/s5987/202501/t20250119_1176166.html

[2] Department of Higher Education, Ministry of Education, 2025, Notice on Soliciting the Third Batch of Typical Application Scenario Cases of "Artificial Intelligence + Higher Education," viewed July 30, 2026, https://www.moe.gov.cn/s78/A08/tongzhi/202506/t20250605_1193095.html

[3] Freeman J, 2025, Student Generative AI Survey 2025, Oxford: Higher Education Policy Institute, viewed July 30, 2026, https://www.hepi.ac.uk/reports/student-generative-ai-survey-2025/

[4] Kasneci E, Sessler K, Küchemann S, et al., 2023, ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education. Learning and Individual Differences, 103: 102274.

[5] Tlili A, Shehata B, Adarkwah MA, et al., 2023, What if the Devil Is My guardian angel: ChatGPT as a Case Study of Using Chatbots in Education. Smart Learning Environments, 10: 15.

[6] Yang X, Zeng J, Li X, 2025, Scenario Refinement and Implementation Practice of the Deep Integration of Artificial Intelligence and Education: Based on Exploratory Multi-Case Analysis Method. Open Education Research, 31(1): 82–92.

[7] Tian Y, Lu L, 2025, Artificial Intelligence Empowering Higher Education: Subjectivity Dilemmas and Path Reconstruction. Journal of Chengdu University of Technology (Social Sciences Edition), 33(5): 96–111.

[8] Sun D, Boudouaia A, Zhu C, et al., 2024, Would ChatGPT-Facilitated Programming Mode Impact College Students' Programming Behaviors, Performances, and Perceptions? An Empirical Study. International Journal of Educational Technology in Higher Education, 21: 14.

[9] Sun D, Boudouaia A, Yang J, et al., 2024, Investigating Students' Programming Behaviors, Interaction Qualities and Perceptions Through Prompt-Based Learning in ChatGPT. Humanities and Social Sciences Communications, 11: 1447.

[10] Haindl P, Weinberger G, 2024, Does ChatGPT Help Novice Programmers Write Better Code? Results from Static Code Analysis. IEEE Access, 12: 114146–114156.

[11] Ahmed Z, Shanto SS, Jony AI, 2024, Potentiality of Generative AI Tools in Higher Education: Evaluating ChatGPT's Viability as a Teaching Assistant for Introductory Programming Courses. STEM Education, 4(3): 165–182.

[12] Li S, Liu J, Dong Q, 2025, Generative Artificial Intelligence-Supported Programming Education: Effects on Learning Performance, Self-Efficacy and Processes. Australasian Journal of Educational Technology, 41(3): 1–25.

[13] Scholl A, Kiesler N, 2024, How Novice Programmers Use and Experience ChatGPT When Solving Programming Exercises in an Introductory Course, 2024 IEEE Frontiers in Education Conference. Washington, DC: IEEE.

[14] Collins A, Brown JS, Newman SE, 1989, Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics, Resnick LB. Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser. Hillsdale, NJ: Lawrence Erlbaum Associates, 453–494.

[15] Zimmerman BJ, 2002, Becoming a Self-Regulated Learner: An Overview. Theory Into Practice, 41(2): 64–70.

[16] Gulikers JTM, Bastiaens TJ, Kirschner PA, 2004, A Five-Dimensional Framework for Authentic Assessment. Educational Technology Research and Development, 52(3): 67–86.

[17] Miao F, Holmes W, 2023, Guidance for Generative AI in Education and Research, Paris: UNESCO.

Publisher's note

Bio-Byword Scientific Publishing remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.