WHIOCE PUBLISHING PTE. LTD.
PROVIDING
FIRST-CLASS SCIENTIFIC INFORMATION
FOR TOP SCHOLARS

2026 Volume 4, Issue 4
Educational Innovation Research
ISSN: 3029-1852 (Online)
ISSN: 3029-1844 (Print)

# Explorations into the Application of Generative Artificial Intelligence in Computer Programming Education at Universities

**Qichao Liu**
College of Artificial Intelligence, Dongguan City University, Dongguan 523419, Guangdong, China



**Abstract:** The prevalence of generative artificial intelligence tools, including Copilot and ChatGPT, has brought brand-new transformation directions for coding education within university computer majors. Traditional undergraduate coding courses are trapped by rigid unified teaching arrangements, insufficient personalized tutoring support, obsolete workshop projects, and oversimplified assessment standards, which cannot cultivate engineering talents matching industrial recruitment standards. Based on years of frontline teaching practice in programming modules, this paper reorganizes the core dilemmas of conventional coding pedagogy, explores how generative artificial intelligence improves classroom teaching efficiency, sorts out four complete teaching links covering pre-class preparation, in-class lectures, offline workshops, and post-course evaluation, and puts forward targeted management measures to address student over-reliance on intelligent tools. The research can provide practical reform references for the digital upgrading of university programming courses.

**Keywords:** Generative Artificial Intelligence; Undergraduate coding pedagogy; Curriculum reform; Engineering workshop teaching

***Online publication:*** April 26, 2026

## 1. Introduction

Most domestic universities still adopt outdated unified teaching modes for undergraduate programming courses. Teachers arrange identical theoretical explanations and offline coding tasks for all students without considering gaps in learners' prior coding experience. Restricted by limited faculty energy, individualized one-on-one guidance cannot be realized in large classes. Workshop resources are updated slowly; most training tasks remain repetitive basic exercises and lack real enterprise engineering scenarios. After-class technical consultation channels are limited, and students often spend hours debugging code without timely professional guidance. Besides, most assessment modes only calculate scores based on final papers and semester projects, ignoring the whole learning process including preview, classroom practice, and repeated error correction, making it difficult for teachers to capture students' persistent knowledge deficiencies [1].

Generative artificial intelligence can make up for multiple deficiencies of traditional coding teaching by providing adaptive learning paths, instant error diagnosis, and dynamically updated industrial training resources. This paper first summarizes four core bottlenecks of university traditional programming teaching, analyzes the practical teaching value

of generative artificial intelligence, builds a full-process classroom application framework covering pre-class, in-class, after-school, and assessment links, and further discusses targeted management strategies to prevent students' excessive dependence on intelligent tools in daily teaching activities [2].

## 2. Bottlenecks restricting the quality of traditional undergraduate coding courses

### 2.1. Uniform teaching arrangements fail to meet diversified learning demands

The mainstream teaching mode of university coding courses unifies teaching progress, knowledge explanation, and offline workshop tasks for the whole class, ignoring individual differences among learners. Some students have mastered basic programming logic before entering university and can quickly absorb new algorithm knowledge, while zero-foundation learners struggle with basic syntax writing and program logic sorting. Due to the limited number of teachers, tiered learning plans cannot be designed for students with different proficiency levels. Top learners lack challenging advanced development tasks, whereas students with weak foundations cannot follow the unified teaching rhythm. Individualized tutoring and stratified competency training are hard to implement, resulting in uneven learning outcomes among the whole cohort [3].

### 2.2. Outdated workshop content creates a gap between campus learning and industrial work

Programming is a discipline focusing on practical engineering operations, and teaching content needs to keep pace with the rapid iteration of industrial technical frameworks. However, most university textbooks update at a slow pace, and most demonstration cases are classic basic programs without real enterprise comprehensive development projects. Conventional offline workshops only train single knowledge points through simple coding exercises, lacking systematic multi-module engineering training. Without complete project development experience, students cannot form systematic full-cycle development thinking, and their independent troubleshooting and engineering practice abilities stay weak. After graduation, new graduates need long adaptation periods to be competent for enterprise development posts, forming an obvious separation between school education and industrial job demands [4].

### 2.3. Delayed feedback mechanism reduces students' independent learning motivation

Code debugging and logical error analysis are the core difficulties for novice coding learners. During classroom workshops and after-school homework, students frequently encounter syntax errors, logical loopholes, and abnormal program operation. Traditional teaching only reserves limited time for classroom Q&A, and after-school communication channels between teachers and students are narrow. Teachers cannot respond to massive student technical questions in real time. When students cannot solve coding bugs independently, long-term ineffective self-checking will weaken their learning enthusiasm, accumulate continuous knowledge blind spots, and hinder the formation of systematic programming ability.

### 2.4. Outcome-focused assessment lacks whole-process learning tracking

Traditional evaluation of programming courses mainly relies on final written examinations and semester comprehensive project scores, which only evaluate students' final learning results rather than track their performance throughout the whole learning cycle. The evaluation system ignores students' preview completion, classroom participation, workshop operation, and repeated error correction records, so teachers cannot accurately identify learners' logical thinking defects and knowledge blind spots, and cannot adjust teaching plans according to real learning data to carry out targeted tutoring.

## 3. Practical value of generative artificial intelligence in coding pedagogy

### 3.1. Adaptive learning logic supports tiered competency training

Equipped with big data analysis and adaptive matching algorithms, generative artificial intelligence records students'

exercise error data, coding proficiency, and knowledge mastery degree in real time, and formulates personalized learning paths for learners at different levels. For zero-foundation students, the system simplifies complex theoretical descriptions, splits programming logic step by step, and provides basic code demonstration fragments. For advanced learners, the platform pushes advanced algorithm research tasks and multi-module integrated development projects. This adaptive training mode completely breaks the uniform teaching defect of traditional classrooms, enabling each student to learn at a pace matching their own foundation and realizing real student-centered tutoring.

### 3.2. 24-hour instant error diagnosis improves workshop training efficiency

Intelligent tools driven by generative artificial intelligence such as GitHub Copilot and ChatGPT provide all-day technical consulting services for students. After learners upload error codes and runtime feedback information, the system quickly locates error sources, analyzes unreasonable logical structures, and provides optimized coding schemes with detailed grammar explanations. Compared with limited manual teacher consultation, generative artificial intelligence has faster response speed and wider coverage of technical problems, which can shorten students' ineffective trial-and-error time and relieve the pressure of teachers' after-class tutoring work.

### 3.3. Dynamic resource generation updates teaching content close to industrial frontiers

Generative artificial intelligence continuously collects emerging industrial technologies, mainstream programming frameworks, and enterprise real project cases, and automatically generates customized teaching materials and workshop tasks according to curriculum teaching objectives. Teachers can quickly replace outdated textbook cases through generative artificial intelligence and design comprehensive engineering training projects matching enterprise development scenarios, solving the problem of slow updates of traditional teaching materials. Diversified algorithm exercises, project development templates, and extended technical cases generated by generative artificial intelligence can broaden students' technical horizons and narrow the gap between campus coding training and enterprise post requirements.

## 4. Full-process application framework of generative artificial intelligence in coding teaching

### 4.1. Pre-class self-study: Intelligent preview guides students to sort out difficulties

High-quality pre-class preparation can significantly improve in-class teaching efficiency, and generative artificial intelligence can build a complete intelligent preview system for students' autonomous learning. Combined with core teaching difficulties of each unit, teachers use generative artificial intelligence to generate personalized preview materials, including simplified knowledge explanations, basic grammar demonstration codes, and lightweight programming exercises. Students complete preview tasks on the intelligent platform and submit confusing technical questions for instant feedback. Meanwhile, the system automatically aggregates all preview data, sorts out common learning difficulties of the whole class, and synchronizes analysis reports to teachers. Teachers adjust classroom teaching priorities based on preview data to focus on solving collective learning barriers, avoiding the formalistic, low-efficiency pre-class preview problem in traditional teaching.

### 4.2. In-class teaching: Optimize theoretical explanation and offline workshop links

In theoretical lecture sessions, teachers use generative artificial intelligence to visualize abstract algorithm logic and programming concepts. For difficult modules such as recursive programs, sorting algorithms, and object-oriented design, generative artificial intelligence splits complex logic through text decomposition and dynamic operation demonstration, helping students quickly grasp core knowledge points. During classroom interactive sessions, teachers generate real-time mini coding tasks and classroom quizzes via generative artificial intelligence, capture the overall learning state of the class based on student submission data, and flexibly adjust teaching progress according to real-time feedback.

In offline coding workshops, generative artificial intelligence acts as an auxiliary teaching assistant throughout the whole practice process. When students write code, the system monitors coding quality in real time and reminds learners of non-standard formats, syntax errors, and hidden logical defects synchronously. When students encounter unsolvable runtime faults, generative artificial intelligence decomposes problem-solving steps and guides independent error checking instead of directly outputting complete standard code. This teaching mode improves students' hands-on operation ability and cultivates their independent logical analysis and error diagnosis thinking. Teachers can also batch check all students' workshop progress and code quality via generative artificial intelligence, summarize common classroom problems for centralized explanation, and greatly improve the overall efficiency of offline workshop teaching.

### 4.3. After-school workshop: Tiered task distribution consolidates students' coding ability

After-school practical training is the key stage to improve students' comprehensive programming ability, and generative artificial intelligence builds a long-term tiered after-school training mechanism. Combined with students' in-class performance and recorded knowledge weaknesses, the platform distributes differentiated homework and workshop tasks: basic consolidation exercises for students with weak foundations, comprehensive ability-improving projects for intermediate learners, and innovative development and advanced algorithm research tasks for top students, realizing accurate layered homework distribution.

Generative artificial intelligence also supports students' independent project development. Learners describe project functional requirements in natural language, and generative artificial intelligence generates basic code frameworks and overall development ideas; students independently complete code polishing, fault debugging, function optimization, and iterative upgrades to finish practical projects including lightweight management systems, front-end website development, and data analysis programs. The intelligent platform powered by generative artificial intelligence automatically marks after-school homework and workshop projects, accurately marks error code fragments, analyzes the root causes of defects, and provides targeted optimization schemes. It generates exclusive personal learning reports for each student to help learners identify their own knowledge blind spots and carry out targeted supplementary training, steadily strengthening independent coding practice ability.

### 4.4. Multi-dimensional evaluation: Build a full-cycle formative assessment mechanism

Generative artificial intelligence breaks the single result-oriented evaluation mode of traditional teaching and constructs a comprehensive assessment framework integrating process tracking, final achievement detection, and competency judgment. Process evaluation comprehensively evaluates students' overall learning performance based on full-cycle data recorded by generative artificial intelligence, including pre-class preview completion, classroom interactive participation, offline workshop operation, after-school exercise submission, and repeated error correction records. Outcome evaluation combines final examinations and complete semester project deliverables to test students' overall knowledge mastery. Competency evaluation analyzes code standardization, logical rationality, innovative design, and troubleshooting efficiency through the algorithm of generative artificial intelligence to judge learners' core programming thinking and practical operation capabilities.

Furthermore, the intelligent system driven by generative artificial intelligence automatically generates class-wide learning analysis reports and individual student growth archives. Teachers optimize subsequent teaching plans by analyzing teaching effects reflected in data reports, while students carry out targeted ability improvement according to their personal learning weaknesses. This forms a complete closed-loop teaching chain of "teaching arrangement–autonomous learning–multi-dimensional assessment–teaching optimization".

## 5. Conclusion

The integration of generative artificial intelligence into university programming teaching is not a simple superposition

of technical tools, but an all-round innovation covering educational concepts, classroom organization forms, and student evaluation systems. In actual daily teaching, universities and frontline teachers need to fully recognize potential risks brought by generative artificial intelligence, including students' excessive dependence on content produced by generative artificial intelligence, biased output logic of generative artificial intelligence, academic integrity hidden dangers, and insufficient deep integration between generative artificial intelligence and curriculum design. Universities can maximize the teaching advantages of generative artificial intelligence and avoid negative teaching impacts by issuing standardized usage specifications for generative artificial intelligence, establishing multi-layer content review mechanisms, and organizing systematic digital teaching ability training for teachers. In future curriculum reform practice, educators should continuously deepen the integrated application of generative artificial intelligence and programming courses, so as to cultivate high-quality computer engineering talents with solid independent coding ability, innovative development awareness, and rigorous critical thinking.

## Funding

Quality engineering construction project of Dongguan City University, "Software Engineering–Provincial First-Class Major Development", and the Young Teacher Development Fund of Dongguan City University (Project No.: KY2025005001)

## Disclosure statement

The author declares no conflict of interest.

## References

[1] Song H, Lin M, 2023, Teachers' Work Transformation in the Era of ChatGPT/Generative Artificial Intelligence: Opportunities, Challenges and Responses. Journal of East China Normal University (Educational Sciences), 41(7): 78–90.

[2] Li Y, Gou J, Wang J, et al., 2023, Teaching Design for Cultivating Higher-Order Thinking Abilities Based on Programming Learning Platforms. Computer Knowledge and Technology, 19(8): 113–116.

[3] Wang Z, 2025, AI Large Language Model-Driven Pedagogical Reform and Practice for the Python Programming Course: A Case Study in the Electronic Information Engineering Program. Journal of Computer and Communications, 13(8): 205–221.

[4] Xu G, Cai J, Jiang B, et al., 2023, ChatGPT/Generative Artificial Intelligence and Future Vocational Education. Journal of East China Normal University (Educational Sciences), 41(7): 64–77.