Journal of Education and Educational Research
ISSN: 2957-9465 | Vol. 18, No. 2, 2026

Transformation and Reconstruction of Programming Course Teaching Mode in The Era of Large Language Model

Shan Hu, Xuesong Dai, Chun Tian

School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan, Liaoning, 114051, China

Abstract: The rapid development of large language models is reshaping the teaching mode of programming courses in universities. While AI tools enhance learning efficiency and enable personalized tutoring, they also pose a risk of students becoming overly dependent and core computer thinking deteriorating. The traditional "grammar-driven" teaching mode faces severe challenges. Based on a systematic analysis of the current status of AI-assisted programming course teaching, this paper deeply analyzes the dual dilemma of "teaching facilitation" and "thinking hollowing out", pointing out the necessity and urgency of teaching mode transformation. In response to the current dilemma, this paper proposes a "thinking-oriented, human-machine collaborative" teaching mode, systematically elaborating on the transformation path from four dimensions: reshaping teaching objectives, reconstructing classroom teaching, upgrading project practice, and innovating evaluation systems. Practical teaching research conducted in two consecutive "C Language Programming" courses shows that this mode can significantly improve students' algorithm design and problem-solving abilities without the aid of AI, while effectively cultivating students' collaborative literacy in mastering AI tools to complete complex engineering tasks. This paper provides an operable theoretical framework and practical reference for the teaching reform of programming courses in the intelligent era.

Keywords: Large Language Model; Programming Teaching; Teaching Mode Transformation; Human-computer Collaboration; Computational Thinking.

1. Introduction

Programming courses are the core vehicles for universities to cultivate computer thinking and engineering practice abilities. For a long time, "elaborate explanation of syntax, independent coding, debugging and error correction" have constituted the basic teaching mode of such courses. However, the explosive popularity of large language models (LLMs), represented by ChatGPT, DeepSeek, and others, is disrupting this tradition at an unprecedented speed. According to the "2025 Student Survey Report on Generative Artificial Intelligence", 92% of undergraduates admitted to using AI tools, an increase of 26% from the previous year [1]. In the field of programming learning, AI has deeply penetrated into various aspects such as code generation, debugging and error correction, and architecture design.

Faced with this technological flood, educators of programming in universities have been deeply pondering: should they adhere to tradition and simply prohibit the use, or should they fully liberalize and let things be? Simply prohibiting the use altogether is akin to covering one's ears to steal a bell, which is not only difficult to enforce but also leads to a serious disconnect between teaching content and the widely adopted "demand-driven programming" model in the industry [2]; however, full liberalization can easily lead students to develop a "tool dependency syndrome", skip the core cognitive process of "problem abstraction-algorithm design", and become "prompt-word engineers" who only know how to copy and paste[3]. Empirical research shows that students who rely on AI for a long time score significantly lower on average in exams without tools than those who program independently , and their knowledge transfer and innovation abilities are severely eroded.

Therefore, the core contradiction of current programming curriculum reform has shifted from "how to teach programming language syntax and processes well" to "how to ensure the effective construction and deepening of students' core computer thinking while fully utilizing AI to enhance efficiency." This is not only an adjustment of teaching methods, but also a fundamental transformation of teaching models. Based on a systematic analysis of the current dilemma, this article attempts to propose a new teaching model that shifts from "knowledge impartation" to "thinking empowerment," and verifies its effectiveness through teaching practice data, providing a feasible transformation path for programming education in the era of large models.

2. Current Status Analysis

Currently, AI-assisted tools have widely permeated the entire process of programming teaching, bringing about a significant "teaching facilitation" effect [8]. However, they have also given rise to a concern of "hollowing out thinking" that cannot be ignored, and these two constitute a sharp binary contradiction.

2.1. Teaching facilitation: efficiency improvement and personalized support

The positive impact of AI tools on programming courses is multifaceted. Firstly, they significantly lower the learning threshold and enhance basic learning efficiency. The intelligent code completion, real-time code generation, and debugging functions in integrated development environments free beginners from tedious grammar memorization and painful debugging processes [1]. Practice at Harbin Vocational and Technical University shows that the "snake" program, which used to take students a semester to complete, can be developed into more complex creative works within a week with AI assistance [4]. Secondly, they strongly support

the implementation of personalized teaching. Adaptive learning systems based on knowledge graphs can dynamically diagnose students' weak spots in knowledge, push differentiated learning paths and exercise resources, and achieve "precise targeting" learning [5]. Research shows that AI can handle about 72% of common questions and preliminary code reviews, freeing teachers from repetitive tasks and enabling them to focus more on high-level teaching design [6]. Finally, they significantly improve the efficiency of course resource construction and updating. Teachers can use large language models to quickly generate a full set of lesson plans including mind maps, interactive questions, and practical projects, and dynamically adjust teaching cases according to technological developments, effectively solving the pain point of lagging resource updates in traditional teaching[9].

2.2. Hollowing out of thinking: Fracture of the knowledge construction chain

However, beneath the prosperous appearance of facilitation, a deeper crisis is spreading. Traditional programming learning follows a complete cognitive loop of "problem analysis → abstract modeling →algorithm design → code implementation → debugging and verification". Each link in this chain corresponds to specific value in thinking training. The intervention of AI tools, especially the code generation function, brutally cuts off this chain - it directly maps "problems" to "codes", skipping the core links of problem analysis, abstract modeling, and algorithm design. Students may seem to have efficiently and conveniently completed teaching tasks, but their computer thinking, algorithm design ability, and problem modeling ability have not been effectively exercised [7].

2.3. Evaluating disability: systematic failure of traditional assessment methods

The crisis of "hollowing out of thinking" has further triggered the systemic failure of the traditional teaching evaluation system. When AI can generate code of a quality far surpassing that of ordinary students, it becomes challenging for teachers to accurately assess students' true proficiency solely based on final assignments or project outcomes. Research indicates that up to 94% of AI-generated content remains unrecognized by teachers, and there is an 83.4% likelihood that the grades earned for AIGC content will be significantly higher than those earned by human students[1]. This compels teachers to expend considerable effort as "academic detectives," utilizing unreliable AIGC detection tools or conducting one-on-one oral defenses to discern the authenticity of assignments and projects, further encroaching on time dedicated to teaching design. The dysfunction of the evaluation system has deprived teaching of precise evaluation criteria, making the reform of teaching models imperative.

3. Construction of New Teaching Model

The key to overcoming the aforementioned dilemma lies not in simply rejecting AI technology, but in systematically reconstructing teaching models to transform AI tools into means that serve the training and cultivation of computational thinking. To this end, this paper proposes a new teaching model centered on "thinking-oriented and human-machine collaboration". Its core framework, as shown in Figure 1, encompasses a deep transformation across four dimensions: teaching objectives, classroom instruction, project practice, and evaluation system.

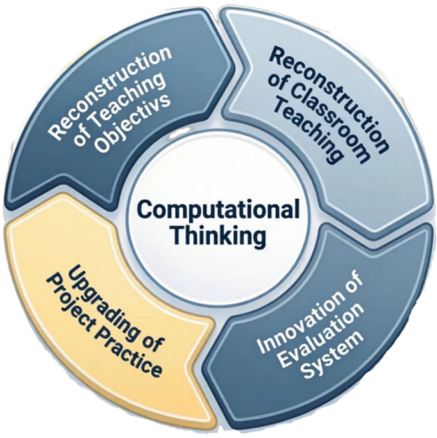


Fig 1. "Thought-based, Human-Machine Collaboration" Teaching Mode Framework Diagram

3.1. Restructuring Teaching Objective

Teaching objectives serve as the starting point for teaching models. In the era of AI, the goal of programming courses must evolve from a singular focus on "the ability to independently write code" to a more comprehensive objective of "possessing solid computer thinking skills and the ability to effectively utilize AI tools". This encompasses two specific levels: 1)Fundamental thinking skills: Core competencies such as problem analysis, abstract modeling, and algorithm design are essential qualities that AI cannot replace. They serve as the cornerstone for students to tackle unknown problems in the future.2) Tool collaboration capability: This refers to the human-machine collaboration ability to accurately describe requirements (Prompt Engineering), evaluate the quality of AI output, and optimize and refactor generated code.

3.2. Reconstruction of Classroom Teaching

During the Basic Knowledge and Core Algorithm Teaching Stage (Approximately the First Two-Thirds of the Course), the Classroom Should Strictly Follow the "No Laptop, No AI" Principle, and Classroom Education Should Strictly Achieve the Following Goals: 1) Strengthen Students' Understanding of Basic Programming Knowledge and Core Algorithms, Enabling Them to Master the Underlying Logic and Thinking Mode of Programming, 2) Improve Students' Logical Reasoning, Problem Analysis, and Hands-on Derivation Abilities, Cultivating the Quality of Solving Programming Problems Independently. To Achieve the Above Goals, the Design of Classroom Teaching is Carried Out in the Order of Theoretical Explanation, Whiteboard Derivation, and Logical Verification. Taking the Teaching of "Pointers and Linked Lists" in C Language as an Example: 1)Theoretical Explanation: The Teacher Uses Blackboard Writing to Explain in Detail the Definition of Pointers, the Concept of Memory Address, as well as the Structure of Linked Lists, the Composition of Nodes, and Other Basic Theories, Combined with Simple Schematics to Help Students Establish Preliminary Cognition.2)Whiteboard deduction: Students are required to leave the keyboard and draw a memory allocation diagram on paper or a whiteboard, manually simulating the process of pointer changes, node insertion, and deletion. For example, when explaining the insertion of a node in a linked

list, students need to draw the memory distribution of the original linked list, the memory allocation of the new node, and each step of pointer adjustment in sequence, clearly marking the address and data of each node.3)Logical verification: After students complete their deductions, they are divided into groups for explanation and mutual evaluation. The teacher focuses on commenting on common issues that arise in students' deductions, such as incorrect pointer assignments and memory leak risks, to help students correct logical deviations.

3.3. Project Practice Upgrade

In the comprehensive project phase of the latter part of the course, AI is formally introduced as a development assistant, but a structured collaboration process must be established to prevent students from using AI tools "mindlessly". The process includes: 1)Structured Requirements Analysis: Students must first complete the "Project Requirements and Architecture Design Specification", including functional module division, data structure selection rationale, interface definition, and boundary condition constraints. This is a top-level design phase that AI cannot replace.2)AI-assisted implementation and review: Based on the design document, use AI to generate code for each module.3) Mandatory manual review and optimization: Students are required to submit a "Code Review Report" explaining the core code logic paragraph by paragraph. They are encouraged to identify potential defects (such as memory leaks, unhandled exceptions, etc.) in AI-generated code and demonstrate their manual optimization results.4)Process transparency: Require students to submit "prompt log" and reflection, making the AI usage process completely transparent.

3.4. Innovation in evaluation system:

Table 1. Reconstructed hierarchical evaluation index system for programming courses

Assessment dimension	Weight	Assessment form and environment	Core assessment point	AI usage strategy
Basic Thinking Layer	60%	Closed-book written test / Offline computer test	Algorithm design logic, handwritten code ability	Strictly prohibited
Collaboration Practice Layer	40%	Open large project + Code review report + Prompt words log	System architecture design ability, AI tool mastery, code review and optimization ability	Allowed and encouraged, but the entire process must be recorded

Establishing a hierarchical, multi-dimensional, and process-oriented evaluation system that is compatible with new teaching objectives is crucial for ensuring the implementation of the model. The core lies in accurately distinguishing between "human abilities" and "machine contributions". The specific indicator system is shown in Table 1.

4. Practical Effect

4.1. Experimental Design

To verify the effectiveness of the aforementioned model, the author conducted a quasi-experimental study with two parallel classes from the 2025 grade majoring in Computer Science and Technology as the subjects. The experimental class (n=42) adopted the new model of "thinking-based, human-machine collaboration", while the control class (n=40) followed the traditional teaching method supplemented with free access to AI-assisted homework. There were no significant differences in the students' entry levels, textbooks, and class hours between the two classes. The experimental period lasted for one semester (16 weeks).

4.2. Comparative Analysis of Core Data

During the semester, a closed-book exam with internet access restricted (out of 100 points) was conducted, focusing on the ability to manually implement core algorithms such as multiple loops, pointers, and linked lists. The score comparison is shown in Table 2.

Table 2. Comparative analysis of final closed-book exam scores between the experimental class and the normal class

Group	Mean	Excellence Rate (≥85)	Pass Rate (≥60)	Pointer Score Rate
Experimental Class (n=42)	79.5	25.7%	91.8%	76.3%
Normal Class (n=40)	65.3	12.5%	75.0%	41.8%

Data analysis indicates that the experimental class significantly outperforms the control class in both average score and score rate on high-order questions (pointers). This suggests that the new model not only enhances overall performance. Furthermore, during the blind review assessment of the final comprehensive project, students in the experimental class demonstrated superior performance in modular design, code robustness, and readability.

5. Conclusion and Outlook

Large language models are not the terminator of programming education, but a powerful catalyst that drives its transformation from a "knowledge impartation" to a "thinking empowerment" mode. Facing the impact of AI, educators should not adopt an ostrich policy of prohibition, but should actively seek change by reconstructing teaching objectives, teaching processes, and evaluation systems, shaping AI as a "cognitive lever" for cultivating students' high-order thinking abilities. The "thinking-oriented, human-machine collaboration" model proposed in this article, by decoupling and reorganizing the deep training of computer thinking and the efficient utilization of AI tools, preliminarily demonstrates the feasibility of cultivating new engineering talents who understand both underlying logic and are adept at using advanced tools in the intelligent era.

Looking ahead, further exploration is needed in the following aspects to deepen this model: first, how to utilize AI technology to construct a more precise and personalized thinking training path; second, how to design more complex multi-agent collaborative teaching scenarios to simulate real

industrial environments.

References

[1] Wang Huiyong, Tang Shijie. Opportunities and Challenges Faced by Programming Course Teaching in the Context of AI Assistance [J]. Information and Computer, 2020(8): 251-253.

[2] Ren Jing, Lu Yao, Tang Jintao, et al. Reshaping Programming Teaching Thinking and Cultivating Collaborative Ability in the Age of Intelligence [J]. Computer Education, 2026(2): 88-92.

[3] Hamsa Bastani, Osbert Bastani, et al. Generative AI without guardrails can harm learning: Evidence from high school mathematics[J]. Proceedings of the National Academy of Sciences of the United States of America .2025(122),.

[4] Harbin Vocational and Technical University. AI Empowers the Construction of a Smart Education Ecosystem [N]. Heilongjiang Daily, 2026-01-22.

[5] Zuo Chunting, Xiao Pingping, Zhang Shuyan, et al. AI-Assisted Personalized Teaching of "Python Programming" for Internet of Things Engineering Major [J/OL]. Communication and Information Technology, 2026.

[6] Leng Xiangguang, Xiong Bolin, Xu Jianjun, et al. Construction of Computer Programming Course Resources Assisted by Large Language Models [J]. Journal of Higher Education, 2025(34): 21-24.

[7] Qi Yue, Wu Xiaoguang, Wang Yiqin. Reconstruction and Practical Research of Human-Computer Collaborative Teaching Mode - Taking Programming Courses as an Example [J]. Modern Distance Education, 2026(1): 48-52.

[8] Gao Honghao, Chen Zhangjin. Artificial Intelligence Empowers the Teaching Reform of Programming Courses [J]. Computer Education, 2024(7): 41-43.

[9] Xiao Le, Shan Xin, Deng Miaolei. Computer Programming Curriculum System Based on AI-Assisted Teaching [J]. Computer Education, 2024(7): 134-139.