International Journal of Multidisciplinary Research and Publications
ISSN (Online): 2581-6187

Constructing and Implementing a "Three Perspectives, Four Integrated Stages" Teaching Model for C Programming with Large AI Models

Qian Zhang[1], Banglei Zhao[1*]
[1]Anhui University of Finance and Economics, Bengbu, 233030, China
Email address: bangleizhao@163.com

Abstract*—This paper proposes a "three perspectives, four integrated stages, and human‑AI collaboration" model for C programming education. Memory, code, and debugging perspectives guide content redesign, while diagnostic guidance, theory‑practice integration, project transfer, and intelligent assessment structure the learning process. The model facilitates personalized learning, coding, debugging, and assessment by integrating knowledge graphs, AI assistants, code-centric large language models, virtual simulations, and learning analytics. The approach aims to develop student's programming, problem-solving, and reflective abilities while mitigating AI hallucinations, overreliance, unreliable test oracles, and academic integrity risks through teacher guidance and human verification.*

***Keywords*—** *large AI models; C programming; human‑AI collaboration; three perspectives in programming; teaching reform; knowledge graph*

I. INTRODUCTION

C programming is a foundational course in computer science and related disciplines in both vocational and undergraduate education. It plays an important role in developing students' computational thinking, algorithmic reasoning, programming skills, and awareness of engineering practices. Traditional instruction, however, continues to face several structural challenges. Course content often emphasizes syntax and elementary algorithms, with insufficient connections to professional practice and authentic projects. Practical sessions frequently follow a linear "teacher demonstration‑student imitation" pattern, providing limited opportunities for students to independently analyze problems and debug complex programs. Differences in students' prior knowledge make a uniform pace inadequate for personalized learning, while conventional assessment methods provide only a partial picture of process-oriented learning, innovation, and engineering competence [1,2]. At the same time, large language models (LLMs), including ChatGPT and DeepSeek, have demonstrated capabilities in code generation, code explanation, error localization, and test-case generation, creating new opportunities for programming education [3, 4].

The introduction of AI models into the classroom does not automatically resolve these challenges. Students may treat an LLM as an answer-generating machine and copy its code without critical review, weakening independent reasoning and debugging skills. Models may also produce inaccurate explanations, logically flawed programs, or unreliable test expectations. Without pedagogical guidance and assessment constraints, these risks may encourage superficial learning in which students appear competent but cannot independently explain or reproduce their solutions [5]. One study of LLM-based unit-test generation for C programs reported a line coverage rate of 91.71%, while the accuracy of test oracles was only 50.05% [6]. This result suggests that LLMs can be effective programming assistants but should not be treated as inherently reliable arbiters of correctness.

The integration of AI into C programming therefore needs to move beyond simply adding tools to existing lessons. It requires a systematic redesign of content, teaching processes, assessment, and risk management. Drawing on research into AI-supported C programming education, the "three perspectives" approach, curriculum-based values education, and unit-test generation, this paper proposes a teaching model based on "three perspectives, four integrated stages, and human-AI collaboration" and discusses its implementation, with the aim of informing the intelligent transformation of similar courses.

II. TEACHING CHALLENGES AND THE RATIONALE FOR INTEGRATING LARGE AI MODELS

A. Key challenges in conventional C programming instruction

First, course content is often insufficiently connected to professional contexts. Conventional courses are typically organized around variables, expressions, control structures, arrays, functions, pointers, and structures, with exercises designed mainly to verify isolated concepts. Students may therefore struggle to recognize how C is used in embedded systems, the Internet of Things, and software engineering [5]. Second, practical instruction often depends on teacher demonstrations and student imitation. When errors occur, students may need individual teacher assistance or time-consuming independent searches. A comparative study by Chen Yufeng et al. [6] reported a first-attempt code correctness rate of 51.2% and an average debugging time of 24.3 minutes per problem under conventional instruction. These findings indicate weaknesses in the write-debug-revise cycle. Third, individualized support remains limited: teachers may find it difficult to diagnose every student's code in depth, and learning platforms may not recommend resources

Qian Zhang and Banglei Zhao, "Constructing and Implementing a "Three Perspectives, Four Integrated Stages" Teaching Model for C Programming with Large AI Models" *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 9, Issue 6, pp. 1-4, November 2026.

based on specific knowledge gaps. Finally, attendance, classroom quizzes, and final programming tests alone cannot adequately measure students' design, debugging, collaboration, and innovation skills.

B. Opportunities and limitations of large AI models

Large AI models offer several potential benefits for C programming education. They can explain syntax and concepts, reducing the cognitive burden on beginners; generate code skeletons and comments to illustrate the relationship between requirements, algorithms, and implementation; explain compiler errors and possible logical defects; and assist with code review and the generation of unit and boundary tests. Zhao Chaoyang et al. [7] note that instruction tuning and reinforcement learning from human feedback can improve models' alignment with human intentions, although reliability remains dependent on training data, feedback, and task characteristics. AI is therefore best positioned to support learning, practice, and formative assessment rather than replace teachers' responsibility for values education, process supervision, or final judgments.

The proposed reform adopts collaboration rather than substitution. Teachers define learning objectives, establish prompt-use conventions, guide learning processes, and make final assessment decisions. AI provides immediate feedback, recommends resources, diagnoses code, and assists with test generation. Students remain responsible for defining problems, selecting solutions, reviewing code, verifying results, and reflecting on improvements. The"teacher-student-machine" model discussed by Xu Yan et al. provides a useful reference for this collaborative relationship [8].

III. Overall Design of the "Three Perspectives, Four Integrated Stages" Model

A. Three perspectives: redesigning course content

Wang Zhenhua et al. [9] proposed incorporating the perspectives of memory, code, and debugging into C programming instruction. This paper adopts these perspectives as the organizing principles for linking individual knowledge points with core programming competencies.

The memory perspective emphasizes the mechanisms of memory allocation, access, and management underlying data types, arrays, pointers, structures, and strings. An AI model can explain pointer-related risks or array-boundary violations in natural language, but teachers should guide students to verify such explanations against memory models, variable lifetimes, and address-access mechanisms rather than merely memorize conclusions.

The code perspective emphasizes modular decomposition, interface design, and code optimization, with functions as basic units. AI-generated code may run correctly while remaining poorly structured, difficult to read, or hard to maintain. Students can compare AI-generated code with manually refactored versions and assess function decomposition, naming conventions, coupling, reusability, and maintainability. The goal is to move beyond asking whether code runs to evaluating whether it is well designed.

The debugging perspective emphasizes test-first practices, boundary testing, breakpoint-based tracing, and post-error reflection. AI can help generate test cases and interpret error messages, but students must determine whether expected outputs are correct in light of requirements and programming semantics. They should understand that high test coverage does not necessarily imply accurate test oracles. Effective debugging requires evidence-based hypotheses, verification, and documentation of corrective actions.

B. Four integrated stages: redesigning the teaching process

Building on the "four-in-one" model and the pre-class exploration, in-class practice, and post-class inquiry cycle, this paper organizes instruction into four connected stages: diagnostic guidance, theory-practice integration, project transfer, and intelligent assessment.

Diagnostic guidance: Before class, knowledge graphs, preparatory quizzes, and AI-based learning analytics identify knowledge gaps and generate learner profiles. Teachers review AI-generated preparatory materials and design tasks that require students to assess, verify, and explain AI responses.

Theory-practice integration: During class, project-based tasks guide students in using AI for code scaffolding, syntax checks, and error analysis. Teachers focus on requirements analysis, algorithm design, memory concepts, and debugging strategies, while helping students critically review and revise model outputs.

Project transfer: After class, group projects and industry-oriented cases require students to use AI for code review, test generation, and optimization analysis. Students also submit code-iteration records and reflective reports.

Intelligent assessment: Learning behavior, AI interaction, and learning outcomes are considered together. Online judge systems, peer assessment, AI-use records, and teacher spot checks support a feedback loop of assessment, recommendations, feedback, and improvement.

IV. Implementation Pathways

A. Course Content Redesign and Pre-class Preparation

Course content is organized into five modules: basic syntax, algorithmic thinking, core techniques, project practice, and interdisciplinary integration. Knowledge graphs connect these modules with the three perspectives of memory, code, and debugging. Before class, teachers provide microlectures, knowledge graphs, and diagnostic quizzes through platforms such as the Chaoxing AI Workbench. AI assistants support concept explanations and question answering, while students verify AI-generated responses through textbooks, code compilation, and experimental evidence. This approach promotes targeted preparation, critical thinking, and evidence-based learning while mitigating hallucinations and content bias

B. In-class Practice and AI-assisted Debugging

Classroom activities integrate theory and practice through authentic programming tasks, following the workflow of requirements analysis, logical design, implementation, and testing. AI tools can assist with code generation, error

Qian Zhang and Banglei Zhao, "Constructing and Implementing a "Three Perspectives, Four Integrated Stages" Teaching Model for C Programming with Large AI Models" *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 9, Issue 6, pp. 1-4, November 2026.

diagnosis, and test suggestions, but students must independently examine problems, justify corrections, and compare program behavior before and after modifications. Virtual simulation helps visualize memory addresses, data changes, and execution paths, particularly when learning pointers and structures. Teacher-guided code reviews further encourage students to improve modularity, readability, boundary handling, and logical correctness.

C. Project Transfer and Integrated Assessment

After class, group projects extend learning to applications such as student-information systems, intelligent lighting, and grade-management software. AI can support initial scaffolding, code review, and test-case generation, while students remain responsible for refactoring, execution-based verification, and documenting AI contributions. Assessment combines diagnostic quizzes, coding and debugging records, project outcomes, peer review, and practical examinations. AI-use rules should vary by task: independent tests measure individual competence, whereas comprehensive projects may permit responsible AI assistance. Oral defenses, process records, and code-similarity checks help strengthen accountability and academic integrity while evaluating programming skills, problem-solving, innovation, and teamwork.

V. Reported Outcomes and Reflection

Existing studies provide preliminary evidence for integrating large AI models into C programming education. Yang Lan et al. [10] developed a "four-in-one" model using the Chaoxing AI Workbench and DeepSeek, reporting that the end-of-semester pass rate increased from 73.79% to 85.36%; 82% of students reported that targeted learning helped deepen their understanding and improve efficiency. In a comparative experiment, Chen Yufeng et al. [6] reported that, in the AI-assisted group, first-attempt code correctness increased from 51.2% to 86.1%, average debugging time per problem decreased from 24.3 to 8.6 minutes, and active classroom participation increased from 53.5% to 90.7%. These results suggest that immediate AI feedback and virtual verification may shorten trial-and-error cycles and strengthen learning confidence.

These figures should nevertheless be interpreted as outcomes reported by the cited studies; their generalizability depends on research design, sample characteristics, teaching conditions, and measurement methods [11]. Benefits do not arise automatically from using AI. They depend on systematic instructional design, effective constraints on model outputs, and sustained development of students' independent reasoning and reflective capacity. Without the three perspectives and appropriate assessment controls, AI may reinforce rote practice and copying, ultimately weakening genuine programming competence. Future implementations should therefore use stronger comparison designs and process-data analysis to evaluate the model more rigorously [12].

VI. Conclusion

Large AI models provide new technological opportunities for reforming C programming education, but the central issue is not which model is selected. It is whether course content, instructional processes, and assessment can be redesigned around core programming competencies. The proposed model uses the memory, code, and debugging perspectives to organize learning; connects the full teaching cycle through diagnostic guidance, theory–practice integration, project transfer, and intelligent assessment; and clarifies responsibilities through collaboration among teachers, AI, and students. Previously reported practice results suggest that this approach may help improve instructional efficiency, debugging skills, and student participation. At the same time, hallucinations, overreliance, and academic-integrity risks must be addressed through human verification, explicit usage policies, process-oriented assessment, and improved teacher AI literacy. Future work should investigate deeper integration of code-oriented LLMs with knowledge graphs, virtual simulation, and learning analytics, and develop a more operational and empirically verifiable framework for evaluating AI-supported teaching. Such work could provide a more robust evidence base for the intelligent transformation of programming courses.

This work was financially supported by the Special Research Project for Curriculum Ideology and Politics of Anhui University of Finance and Economics. (Grant No.: acszjyzd2025010）

References

[1] Ye, R., Fan, J., Bernuy, A. Z., Karnalim, O., Denny P., Leinonen, J., Liut M. "Correctness, Convergence, and AI-Generated Code Detection: A Longitudinal Study of Student and Large Language Model Code in Introductory Programming," Unpublished Manuscript, 2026.

[2] Choi, S., & Kim, H. "The impact of a large language model-based programming learning environment on students' motivation and programming ability," Education and Information Technologies, 2024.

[3] Diamantopoulos, A., Sintoris, C., Demetriadis, S. "Feedback to Students and Instructors of a Programming Course Through a Large Language Model," Conferences of the Greek Scientific Association of Information and Communication Technologies in Education, 1159-1168, 2026.

[4] Sheese, B. E., Liffiton, M., Jaromír, J., Paul Denny. "Patterns of Student Help-Seeking When Using a Large Language Model-Powered Programming Assistant," Proceedings of the 26th Australasian Computing Education Conference, 2024.

[5] Hsu, H. P. "From Programming to Prompting: Developing Computational Thinking through Large Language Model-Based Generative Artificial Intelligence," TechTrends: Linking Research & Practice to Improve Learning, 69(3), 2025.

[6] Chen, Y., Lin, Y., Xun, W. "Research on the Reform and Practice of AI-Deeply Empowered Teaching Mode of C Language Programming in Higher Vocational Colleges," Intelligent Manufacturing Research and Application, 2(8), 163-165, 2025.

[7] Zhao, C., Zhu, G., & Wang, J. "The Inspiration Brought by ChatGPT to Large Language Models and the New Development Ideas of Multi-modal Large Models," Data Analysis and Knowledge Discovery, 7(3), 26-35, 2023.

[8] Xu, Y., Nan, Y., Xu, J. "Ideological and Political Education in C Programming Courses for Large Language Models," Computer Education, 11, 115-118, 2025.

[9] Wang, Z., Geng, N., & Sun, H. "Research on the Reconstruction of C Language Teaching Content Integrating the "Three Views of Programming"," Journal of Higher Education, 14, 39-42, 2024.

Qian Zhang and Banglei Zhao, "Constructing and Implementing a "Three Perspectives, Four Integrated Stages" Teaching Model for C Programming with Large AI Models" *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 9, Issue 6, pp. 1-4, November 2026.

[10] Yang, L., Liu, W., & Li, G. "Exploration of the Reform Path of AI-Empowered C Language Programming Course," Computer Knowledge and Technology, 19, 169-171, 2025.
[11] Wysokińska, A., Duch, P., Bartosik, B., Fabijanska A. "Can AI Code? A Head-to-Head Showdown Between Large Language Models and Student Programmers in C," IEEE Access, 14, 50908-50928, 2026.
[12] Zhao, Q., Zhang, L., Liu, F., "Peer-aided repairer: Empowering large language models to repair advanced student assignments," Empirical Software Engineering, 31(2), 33, 2026.