Research on Experimental Teaching of Programming Courses Assisted by Large Language Models

Jianhua Zhao[1,*], Ning Liu[2]
[1]*School of Mathematics and Computer Application, Shangluo University, Shangluo, China*
[2]*Faculty of Economics and Management, Shangluo University, Shangluo, China*
**Corresponding Author*

Abstract: With the breakthrough development of generative artificial intelligence technology, the field of higher education is accelerating into a new stage of deep integration between large language models and traditional teaching. This paper analyzes the current status and existing problems of practical teaching in programming courses, and explores the methods and pathways for integrating large language models into the practical teaching system of programming courses. By leveraging large language models to construct professional knowledge graphs, assist personalized learning, build multimodal fusion interactive experimental scenarios, support comprehensive experimental project development, and reconstruct learning outcome evaluation, we aim to cultivate computer professionals who meet the requirements of the large-model era. Furthermore, the paper identifies issues that require attention in the educational application of large language models and proposes strategic recommendations.

Keywords: Large Language Model; Personalized Recommendation; Programming; Knowledge Graph; Industry-Education Integration; Experimental Teaching

1. Introduction

With the rapid advancement of artificial intelligence (AI) technology, the deep integration of AI and education has become an important direction for educational transformation. In recent years, relevant authorities have successively introduced policies to promote the application of AI in education, advocating the use of intelligent technologies to facilitate changes in teaching and learning methods and to enhance digital literacy and skills across society. On the one hand, at the policy level, there is encouragement to integrate AI technology into all aspects of teaching and learning, to explore new models of human-machine collaborative education such as intelligent learning companions and intelligent teaching assistants, and to shift educational goals from knowledge transmission to competency cultivation. On the other hand, targeted guidance documents have provided directional recommendations for the standardized application of generative AI in teaching and learning for educators [1]. These initiatives demonstrate that the convergence of AI and education is moving from theoretical exploration to practical implementation, becoming a significant driving force for advancing educational modernization [2].

Large language models are a class of language generation models based on the GPT (generative pre-trained transformer) architecture [3], with parameter scales typically at the billion (10^9) level or above. They are pre-trained on massive multimodal data including text, images, and audio through unsupervised or self-supervised learning, capable of generalizing to various downstream tasks beyond the pre-training data, and exhibiting emergent capabilities as the parameter and data scales increase. Currently, representative large language models include DeepSeek, Doubao, Qwen, PanGu-Coder, Gemini, Claude, and GitHub Copilot, which possess core capabilities such as natural language understanding, knowledge question answering, code generation and optimization, and logical reasoning.

The rapid development of large language model technology has accelerated the intelligent transformation of various industries. The demand for talent in some traditional positions, such as bank tellers, archival management, and junior accountants, has decreased, while the demand for new positions such as large model

research and development, algorithm research, intelligent software development, and digital operations has increased. Meanwhile, the development of large language model technology has also put forward new requirements for the competencies and literacy of computer professionals (see Table 1).

Table 1. Changes in Competency and Literacy Requirements for Computer Professionals in the Era of Large Language Models

Competency and Literacy Requirements	Traditional Era	Era of Large Language Models
Project development capability	Focused on programming skills, proficiency in a specific development language	Lower requirements for programming skills; emphasis on complex task analysis and comprehensive design capabilities
Interdisciplinary knowledge integration capability	No interdisciplinary knowledge requirements	Possession of interdisciplinary knowledge; compound talents
Team collaboration capability	Relatively low requirements	Relatively high requirements
Autonomous learning capability	Primarily teacher-guided learning; mastery of required professional knowledge content is sufficient	Increased weight of autonomous learning; expanded knowledge volume; higher requirements for interdisciplinary knowledge
AI application capability	Work completed independently by humans	Efficient human-machine collaborative work

2. Current Status of Experimental Teaching in Programming Courses in Chinese Universities

As the core carrier for cultivating computer professionals in universities, the teaching quality of programming courses directly determines students' technical proficiency, logical analysis, and innovation capabilities. These courses have high practical requirements, serving as both a means to consolidate theoretical knowledge and a platform to reflect students' programming abilities. However, the traditional experimental teaching system for programming courses in Chinese universities faces the following problems.

2.1 Independent Experimental Content across Courses with No Closed Knowledge Loop

Modern higher education originated from the academic tradition of disciplinary specialization. University department structures, faculty assessments, and teaching evaluations are all centered on individual courses, with each professional course being relatively independent and weak inter-course knowledge correlations, usually taught by different instructors [4]. Due to constraints such as teaching hours, individual faculty thinking differences, and textbook resources, knowledge fragmentation and cross-course knowledge gaps easily occur. For example, in the introductory course C Language Programming, experimental content often focuses on memorizing basic syntax such as variables, selection, and loops. For later chapters on pointers, structures, and algorithm logic, instructors often stop at a superficial level due to insufficient class hours, high difficulty, and the assumption that subsequent courses will cover these topics. In advanced courses such as Data Structures and Algorithm Analysis, experimental content assumes students already possess modular programming thinking, directly diving into linked lists, trees, and sorting algorithms, lacking experimental projects that bridge and review pointers, structures, and algorithm logic from C language. This causes students to feel that algorithm difficulty suddenly increases, making programming difficult. In high-level courses such as Software Engineering, Mobile Application Development, and Advanced Architecture Technology, the focus is on development processes and framework usage, often neglecting the impact of data structures on system performance and the optimization role of algorithms in business logic. When working on projects, students only seek to make code run, ignoring code readability, scalability, and whether the solution is optimal.

Experiments in different courses are often designed around the core knowledge points of a single course, lacking explicit logical connections and transitions between experiments. The knowledge students acquire presents fragmented characteristics, and the knowledge system has not formed a complete closed loop.

2.2 Traditional Experimental Teaching Methods Struggle to Meet Students' Personalized Learning Needs

In traditional experimental teaching, instructors face the following problems when tutoring students: when the number of questions is high, teachers can only tutor students one by one, making it difficult to respond to student questions in a timely manner, resulting in low tutoring efficiency. Teachers' tutoring is limited by time and space, making it difficult to

accompany students throughout their learning beyond the classroom. Some students are introverted, shy to express themselves, and dare not ask questions when encountering difficulties, thus unable to obtain effective guidance and help. Programming experiment courses in Chinese universities generally adopt unified experimental content and schedules. This approach could meet the learning needs of most students in traditional teaching models. However, with the advent of the large-model era, students have more autonomous learning channels, and the differences in knowledge levels and learning progress among students have increased. This fixed-content, unified-progress experimental teaching method struggles to precisely match each student's learning needs [5].

2.3 Traditional Programming Software is Monotonous and Rigid, Lacking Interactivity and Engagement

Traditional programming environments are designed with a logic of function-first and visual minimalism, mostly featuring a fixed structure of menu bar + toolbar + code editing area + console. The experimental process also follows relatively fixed stages of coding-compiling-debugging-running. The only way to interact during the experiment is typing code, lacking interactivity and engagement. For long-term programmers, this easily leads to psychological fatigue. Additionally, traditional programming software generally only provides final running results. Although this can verify program correctness in general, the detailed execution process and the underlying algorithmic principles are not displayed in the running interface. Some students who are not good at deep thinking may not fully understand the algorithm design, only forming mechanical programming memory. This is unfavorable for subsequent knowledge learning and programming ability cultivation.

Many programming software products are designed to meet the production needs of enterprise developers, without targeted adaptation for student groups' cognitive characteristics, learning goals, and ability foundations. The rigid constraints of programming software on syntax make the programming process more cold and mechanical. Students often debug programs multiple times without success, generating frustration and negative emotions. For students, programming software is not only a production tool but also a learning tool, yet traditional software lacks scenario-based learning, question answering, and other interactive modes.

2.4 Experimental Content is Overly Basic and Insufficiently Aligned with Industry Needs

Current experimental content in university programming courses is limited by hardware environments, traditional teaching concepts, and students' comprehensive programming levels, focusing on basic problems (such as the Data Structures experiment project Bubble Sort, which is designed to verify the correctness of the bubble sort algorithm) [6]. Insufficient attention is paid to students' design and decision-making practical abilities, especially practical skills in emerging industrial software. The experimental system lacks comprehensive, innovative, and engineering-oriented project training. An industrialized project requires integrating multi-dimensional knowledge including design, algorithms, databases, software engineering, and front-end development. Basic experiments cannot allow students to experience the full process of requirements analysis to architecture design to module development to testing optimization, making it difficult to form engineering thinking characterized by decomposing problems from a system perspective and integrating knowledge. With the development of large language model technology, the speed of industrial iteration and technological updating has accelerated, and the gap between the competency literacy of computer professionals and industry demands has become increasingly apparent.

2.5 Lagging Learning Outcome Evaluation System

In traditional programming experiment courses, teachers' evaluation of student learning outcomes largely relies on experiment reports, which are entirely filled out by students and collected by class representatives after class before being submitted to the teacher. Therefore, experiment reports suffer from plagiarism, discrepancies between recorded and actual experimental processes, and insufficiently comprehensive feedback. Teachers' grading of experiment reports also faces problems such as insufficient energy, inconsistent evaluation standards, delayed feedback, and vague feedback [7]. Overall, the learning outcome evaluation system for experimental courses is lagging.

The above problems have long constrained the quality and efficiency of computer professional talent cultivation in China. Targeting the above experimental teaching status, this paper proposes methods and pathways for introducing large language models into the experimental teaching of computer programming courses, transforming the traditional teacher-student experimental teaching model into a teacher-student-machine ternary collaborative model, and cultivating computer professionals who meet industry demands in the digital intelligence era.

3. Methods and Pathways for Integrating Large Language Models into the Practical Teaching System of Computer Programming Courses

3.1 Using Large Language Models to Construct Professional Course-Group Knowledge Graphs and Optimize Experimental Projects

A knowledge graph is a network structure diagram created based on the internal connections of knowledge, enabling people to understand the world from the perspective of relationships [8]. Computer professional teaching teams can use talent training programs, course syllabi, textbooks, courseware, student experiment reports, assignments, examination papers, as well as industry needs, enterprise employment feedback, educational expert opinions, and school professional characteristics as foundational data. Leveraging large language models to assist in sorting out knowledge points and internal logical relationships across courses, they can establish professional course knowledge graphs. Using knowledge points-difficulty levels-experimental problems as association rules, a database is established, and visualization is achieved using tools such as Gephi and Matplotlib. Based on the professional knowledge graph, existing experimental problems are annotated with data labels and knowledge tags. During experimental teaching, problems can be dynamically extracted according to teaching needs (as shown in Figure 1). Based on the relationships between knowledge points in the knowledge graph, the experimental project library is optimized: outdated or highly repetitive problems are eliminated, and experimental projects corresponding to the knowledge points that need to be assessed are added. Large language models can also be used to analyze the latest industry needs and new technology applications, making experimental projects closer to market demands. After optimization, experimental projects place greater emphasis on the logical relationships between knowledge points, which is conducive to students systematically mastering professional knowledge systems and progressive learning, and promotes the comprehensive improvement of students' programming thinking, engineering practice, and innovation capabilities.

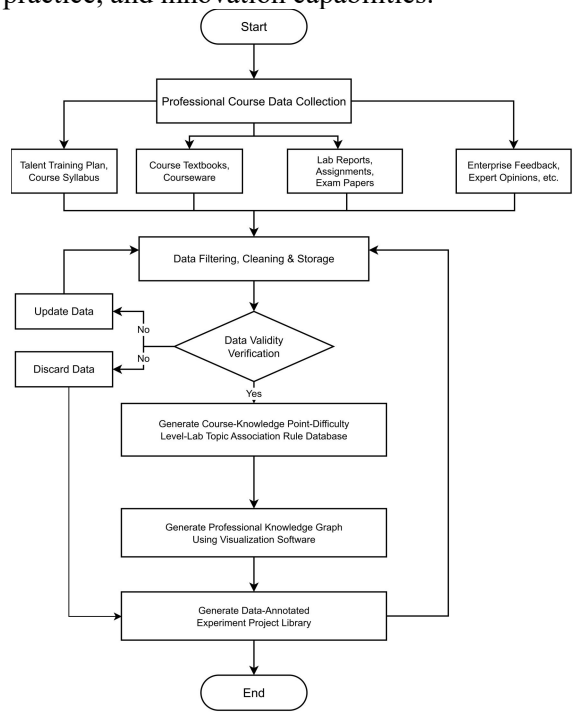

Figure 1. Flowchart of the Process for Generating a Knowledge Graph for Professional Course Clusters with the Assistance of Large Models

3.2 Large Language Models Provide Powerful Support for Students' Personalized Learning

Large language models possess powerful language understanding and knowledge retrieval capabilities, breaking through time and geographical constraints to provide students with all-weather companionship and tutoring. Students can obtain precise and efficient learning support anytime and anywhere, reconstructing the boundaries of personalized learning scenarios and helping students achieve autonomous, efficient, and sustainable learning.

Computer science students' learning has distinctive personalized rhythms and randomness. Programming practice, experiment debugging, and knowledge exploration often require large amounts of fragmented time, and

learning needs may arise at any time. Traditional tutoring models, centered on classroom Q&A, cannot meet students' diverse and random learning needs. Through the technical advantage of uninterrupted operation, large language models completely break through time limitations, becoming always-online tutoring teachers beside students. Students can initiate tutoring requests at any time according to their own learning habits and needs. Large language models can quickly identify problems, analyze causes, provide detailed debugging ideas and code modification suggestions, help students resolve doubts in a timely manner, and avoid the accumulation of doubts affecting learning progress. For students conducting autonomous learning and experimental projects during holidays, large language models can accompany them throughout the process, from project planning, code writing, to debugging optimization and problem troubleshooting, providing guidance at any time to ensure orderly progress of autonomous learning and improve the timeliness of autonomous learning.

For example, when students first learn C language programming, a common error is not distinguishing between Chinese and English punctuation marks, which can lead to compilation errors: [Error] stray '\273' in program. However, students learning C language for the first time often do not understand the meaning of this compilation hint, and may thus fall into learning stagnation and confusion. In the large language model environment, students can input the compilation hint information into the question box, and the large language model can quickly provide an explanation: it is because the code contains invisible non-ASCII characters, and prompt the possible causes of the error: containing Chinese symbols / full-width characters. With such detailed prompts, students can quickly find the cause of the error and correct it by comparing with their own source code.

With multiple programming samples, large language models can analyze students' programming habits, high-frequency errors, and knowledge weak points, establishing exclusive learning profiles for students. Combined with students' learning progress and mastery, the model can push targeted practice problems and knowledge summaries, guiding students to consolidate weak links. For example, some students are good at basic syntax application but struggle to grasp object-oriented programming concepts such as encapsulation, inheritance, and polymorphism, while others have shortcomings in algorithm design and optimization. Through differentiated experimental projects, students can gradually deepen their knowledge understanding and improve programming abilities along learning paths suitable for themselves.

3.3 Constructing Multimodal Fusion Interactive Experimental Scenarios to Enhance Students' Programming Interest

The realization of multimodal interaction depends on the deep integration of computer vision (CV), natural language processing (NLP), automatic speech recognition (ASR), and machine learning technologies [9]. Currently, multiple generative large language models support multimodal interactions including text, voice, and vision. Introducing multimodal interaction during programming experiment courses can enrich experimental forms and enhance students' programming interest.

Taking the data structure course experiment as an example, completing linked list element insertion and deletion operations is an important content. In traditional programming environments, students can only see the final form of the linked list, but how the linked list changes is not displayed by traditional programming software. In the large language model-assisted experimental environment, students can input key prompt words such as demonstrate animation of linked list element insertion process into the large language model. The model can then initiate internet content retrieval mode and display relevant videos, allowing students to seamlessly switch to video learning. After resolving the knowledge difficulties, students can continue to complete programming tasks.

In collaborative working mode, large language models can analyze program code in real time, provide error prompts and optimization suggestions, and analyze the causes of errors, improving students' programming efficiency and enhancing the programming experience.

3.4 Improving the Efficiency of Comprehensive Experimental Project Development and Promoting Industry-Education Integration

Large language models, through natural language understanding, code generation, logical

reasoning, and knowledge fusion capabilities, significantly improve the efficiency of application-oriented project development, ensuring code quality and reducing time costs. For example, GitHub Copilot can automatically generate code based on developers' requirement descriptions, reducing manual coding workload. Research reports show that GitHub Copilot can help developers effectively increase programming speed by 55%, reducing average programming time from nearly 3 hours to nearly 1 hour [10].

In the software testing phase, large language models can automatically generate test cases to verify code correctness, reducing manual review workload. This allows students to be freed from simple repetitive programming tasks and focus more on cultivating comprehensive abilities such as system design and algorithm optimization.

Large language models can also quickly call IDE plugins, open-source datasets, enterprise knowledge bases, and cloud storage, reducing the difficulty of comprehensive software project development. For example, developing intelligent chatbot projects, drone equipment, and various industrial control software allows students to master skills required by cutting-edge industry demands, shortening the gap between campus learning and industry needs, and promoting industry-education integration.

3.5 Large Language Models Reconstruct Learning Outcome Evaluation Methods to Improve Evaluation Validity

Large language models can reconstruct learning outcome evaluation methods, achieving comprehensive, precise, and timely scoring and feedback.

First, the evaluation dimensions are broadened, no longer limited to experiment reports, but interacting with students throughout the process to grasp their learning status. Large language models can conduct multidimensional data collection of students' experimental process data, including code writing time, number of modifications and debugging attempts, code compilation pass rate, active learning time, and departure time (as shown in Figure 2). Second, large language models' evaluation of programming code is not limited to right or wrong, but can also analyze deep-level issues in program code: whether there are logical loopholes, whether the algorithm is optimal, and the degree of coding standardization. Large language models can also generate detailed learning outcome analysis reports through data analysis (such as class common errors and high-frequency knowledge weak points), providing data support for teachers' teaching and differentiated experimental content setting [11].

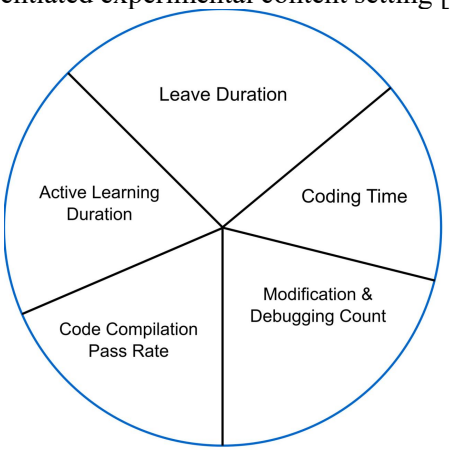

Figure 2. Schematic Diagram of Large Language Model Collecting Multidimensional Learning Data

In the grading of student experiment reports and practical assignments, large language models can quickly complete the grading work and provide timely feedback to students. At the same time, they greatly reduce teachers' repetitive workload, allowing teachers to devote more energy to teaching preparation and student tutoring, thereby promoting teaching interaction and improving teaching effectiveness.

4. Issues to Consider in the Application of Large Language Models to Assist Experimental Teaching in Programming Courses

Large language models represent a great technological innovation of the information age, bringing tremendous changes to various industries and the education sector. Applying large language models to experimental teaching in programming courses can greatly improve the current teaching status and student learning outcomes. However, there are also some issues that require attention.

4.1 Over-Reliance on Large Language Models

Some students may exhibit learning inertia, skipping critical learning steps such as thinking, debugging, and trial-and-error, directly copying code generated by large language models without understanding its principles. Therefore, in the teaching process, teachers need to strengthen guidance to help students establish a

correct understanding of large language models, treating them as crutches in the learning process rather than express trains. In daily tutoring and questioning, questions that detect active thinking processes should be set to prevent students from over-relying on large language models and weakening their active thinking abilities.

4.2 Technical Limitations of Large Language Models

The core principle of large language models is statistical pattern matching rather than true understanding of semantics and the real world. They generate outputs by learning the association probabilities of words and sentences in massive data, unable to establish mappings between concepts and reality like humans. Therefore, when user instructions are vague, ambiguous, or contain complex constraints, large language models may provide unsatisfactory answers. Limited by training data, they suffer from insufficient knowledge timeliness and data hallucination issues.

Programming languages and tools in computer science update rapidly, so attention should be paid to timely updates of large language model versions. In application, if suspected knowledge errors are found, multi-source data cross-checking or experimental verification should be conducted to ensure knowledge correctness.

4.3 Teacher Role Transformation and Competency Enhancement

The integration of large language models into education brings both opportunities and challenges to teachers' roles. On the one hand, the introduction of large language models reduces teachers' burdens in lesson preparation, teaching, tutoring, and assignment grading. On the other hand, large language models can directly face students, with their massive knowledge systems and second-level response efficiency being astounding, challenging teachers' traditional status as knowledge transmitters and problem solvers [12].

As teachers, they should actively adapt to the changes of the era, transform their role positioning, and shift from knowledge transmitters to learning guides and enablers. In practical teaching of programming courses, they should guide students to correctly use large language models to improve learning efficiency and professional programming levels. They should leverage the learning analytics functions of large language models to more precisely grasp students' learning status and knowledge difficulties, providing targeted tutoring and explanations. When suspected errors are found in content generated by large language models (such as code bugs or knowledge deviations), teachers should act as the final reviewers to ensure students receive correct knowledge answers. The large language model era puts forward higher requirements for teachers' professional literacy, requiring teachers to continuously learn, expand and update their knowledge systems, and quickly keep pace with information technology development and industrial changes.

4.4 Data Security and Privacy Protection

Data security and privacy protection are important issues to consider in the application of large language models. In the process of using large language models, users often need to authorize access permissions or upload private data to cloud servers, posing risks of data being cached, stolen, or leaked. For example, leakage of student grades, behavior trajectories, identity information data (such as ID numbers, addresses, phone numbers), experimental data, and business information may cause serious losses to schools, related enterprises, and individuals. Therefore, in the application of large language models, attention should be paid to avoiding exposure of private data, and sensitive data should be filtered or desensitized.

Privatized deployment of open-source large language models is an effective solution to data leakage problems [13,14]. Deploying models on campus intranets or dedicated servers, all data interactions are completed locally without uploading to external platforms, fundamentally avoiding leakage risks caused by cross-boundary data transmission. Management departments can fully control model access permissions, input-output filtering rules, without worrying about external attackers stealing internal data through prompt injection or inversion attacks. At the same time, unnecessary network interfaces can be closed to isolate public network attacks.

5. Conclusion

Applying large language models to experimental teaching in programming courses, through professional knowledge graphs, real-time knowledge Q&A, multimodal experimental scenarios, industry-oriented comprehensive

projects, and process-oriented multidimensional evaluation methods, allows each student to develop progressively along a path suitable for themselves. It enables teachers to transform from knowledge transmitters to learning companions, representing a key pathway for universities to actively adapt to the development of the era, deepen industry-education integration, and improve talent cultivation quality. The deep integration of large language models and the education field requires the collaborative evolution of policies, technologies, and educational concepts, ultimately achieving a new ecosystem of human-machine collaborative education.

Acknowledgments

This work was supported by the 2024 Project of the 14th Five-Year Plan for Educational Science of Shaanxi Province (Grant No. SGH24Y2302), Scientific Research Program Funded by Education Department of Shaanxi Provincial Government (Program No. 24JR059), and Key Research and Development Program of Shaanxi (Program No. 2025CY-YBXM-002).

References

[1] Zhang X X, Sun J. "Artificial Intelligence +" Cultural Construction: A New Vision for the Future of Pubishing: A Perspective on the "Opinions on Deepening the Implementation of the 'Artificial Intelligence+' Action Plan". Science-Technology & Publication, 2025, 44(10): 38-49.

[2] Gu X Q. Integration and Transcendence: The Role Reconstruction and Practical Path for Teachers in the Era of Artificial Intelligence — Interpretation and Reflection Based on the "Guidelines for Teachers' Application of Generative Artificial Intelligence (First Edition)". People's Education, 2025, (24): 41-44.

[3] Yang X N, Zhu W X, Li J. Research review on large language models in software testing. Technology Innovation and Application, 2026, 16(03): 68-72, 76.

[4] Cui Y Q, Quan P P. The modernity of university discipline and its transcendence. Journal of East China Normal University (Educational Sciences), 2019, 37(02): 73-80.

[5] Zhang L. Strategic exploration of AI-driven personalized teaching in higher education from the perspective of digital intelligence empowerment. Journal of Jilin TV & Radio University, 2025, (05): 139-142.

[6] Tang L J. Innovation of new quality applied talent training model under the background of digital centralized knowledge production. Heilongjiang Researches on Higher Education, 2025, 43(12): 61-68.

[7] Zhao M Y, Cai X J, Shen Z Y. Research and practice on student ability evaluation in the era of artificial intelligence. China University Teaching, 2025, (11): 61-68.

[8] Li X. Construction and application research of knowledge graph for online courses in smart education. Journal of Guangzhou Open University, 2025, 25(01): 48-55, 71.

[9] Li B Y, Bai Y, Zhan X N, et al. The technical features and aromorphosis of artificial intelligence generated content (AIGC). Documentation, Information & Knowledge, 2023, 40(01): 66-74.

[10] Jin Y L, Wang X C. How artificial intelligence empowers curriculum reform. Basic Education Curriculum, 2026, (02): 4-10.

[11] Yan H L. Research on innovation path of university teaching management mode in the digital intelligence era. Teaching of Forestry Region, 2025, (12): 47-50.

[12] Liu Y, Yang X H. Research on the crisis and identity reshaping of teachers' roles in the AIGC era. Journal of Dali University, 2025, 10(03): 40-44.

[13] Tang B W, Zhou C L. Discussion on security protection technology of large models. Secrecy Science and Technology, 2025, (05): 5-10.

[14] Wu Y, Li W J, Liu X, et al. Global evolution trends and prospects of smart education research. Journal of Xi'an Jiaotong University (Social Sciences), 2025, 45(06): 37-45.

