A Novel Teaching Method Based on AI Automated Programming: a Case Study of Python Programming

Feng Zhu[1], Zhou Zeng[2,*]

[1]Shanghai Urban Construction Vocational College, Shanghai, China
[2]Shanghai Open University, Shanghai, China

zhufeng_ujs@126.com, *zengzhou@sou.edu.cn

Abstract. In traditional Python programming education, students often struggle to bridge the gap between learning programming and implementing practical projects, making it hard to grasp project architecture and build functional applications. This study introduces an AI-based automated programming teaching method to optimize the teaching process and enhance practical programming skills. Centered on the "practice first, learn later" approach, the process includes setting project topics, analyzing requirements with the POE large language model foundation platform, decomposing modules via Scrum methods, generating and optimizing code with Cursor.AI, analyzing programs using UML diagrams from PlantUML, and extracting knowledge points from the code. AI tools facilitate automated code generation, program analysis, and quick understanding of logic through UML diagrams. Findings show that the method significantly improves students' ability to understand complex architectures, decompose functions, and solve practical problems. The "project-driven + AI-assisted" teaching model shortens the transition from theory to practice, enhancing students' programming skills and application capabilities. By integrating AI tools, this study presents an innovative "practice-to-theory" teaching approach for Python education. The tools and methods can be directly replicated by students, providing both theoretical significance and practical application value, and improving operability in real-world scenarios.

Keywords: Python programming education, AI-assisted programming, innovative teaching model, automated code generation, UML program analysis.

1 Introduction

In recent years, vocational colleges and open universities have successively launched artificial intelligence-related majors. However, students in these two types of institutions differ significantly in their learning backgrounds and goals. Most students in open universities are working professionals from non-IT fields who choose AI-related majors to enhance their competitiveness in their current jobs or achieve career transitions. On the other hand, students in vocational colleges often have IT-related backgrounds, and their primary goal is to prepare for future employment by acquiring the necessary

H. Kannan et al. (eds.), *Proceedings of the 5th International Conference on Internet, Education and Information Technology (IEIT 2025)*, Atlantis Highlights in Social Sciences, Education and Humanities 40,
https://doi.org/10.2991/978-94-6463-803-5_45

technical skills. Despite these differences in background and objectives, both groups share a common need: they hope to gain programming skills that will give them an advantage in employment, career transitions, and professional development. However, the programming learning outcomes for both groups have generally failed to meet expectations, directly affecting their ability to solve real-world problems and their overall competitiveness in the job market.

Currently, Python programming courses often suffer from a disconnect between theory and practice. While students may grasp basic syntax and complete some tasks in laboratory settings, they often struggle to solve real-world problems effectively. Employers frequently report that vocational graduates lack the programming skills required to meet job demands and need additional training before they can take on their roles. Similarly, working professionals from open universities find it challenging to write programs that can address complex practical needs when attempting to use Python in their work. The primary reason is the significant gap between mastering basic programming skills and being able to apply them effectively in real-world scenarios. Bridging this gap usually requires extensive project experience. However, both vocational and open university students often lack sufficient access to project opportunities and practical resources. This "learn but cannot apply" phenomenon severely limits students' ability to enhance their programming skills and widens the gap between academic training and industry requirements.

The rapid advancement of generative AI technology has brought new possibilities for addressing these issues. AI-powered automated programming tools integrate functions such as code generation, requirement analysis, error detection, and code optimization. These tools significantly lower the barriers to programming and improve both development efficiency and code quality. Research predicts that by 2030, AI-assisted programming tools will contribute over $1.5 trillion to global GDP, equivalent to adding 15 million "effective developers" to the workforce. Today, AI programming tools can assist users in tasks such as design, development, and optimization. Their core value lies in helping developers rapidly achieve functional requirements, improve productivity, and enhance code quality. The widespread adoption of this technology offers an efficient and economical solution for addressing the lack of practical programming experience among students.

This study designed and tested a Python programming teaching method centered on AI tools. Specifically, the research focuses on four key aspects: first, exploring how to use AI tools combined with mind maps for requirement analysis to help students quickly organize problems and clarify requirement logic; second, investigating how AI tools can assist students in generating code to directly implement functionalities; third, studying how to utilize AI tools to automatically generate UML diagrams, sequence diagrams, use case diagrams, and other visualization tools to help students better understand program structures and logic; and fourth, leveraging AI tools in conjunction with course content to guide students in reviewing code implementation details and connecting generated programs with course knowledge points to consolidate theoretical knowledge and deepen their understanding of real-world project implementation methods. This process creates an AI-enabled teaching model that spans from requirement analysis to code generation and program understanding to theoretical reinforcement. It

ensures that students develop programming skills while solving practical problems. This comprehensive teaching design not only compensates for students' lack of project experience but also helps them internalize programming thinking through hands-on learning, ultimately achieving a virtuous cycle between theory and practice.

This teaching method was tested during the spring semester of 2025, yielding notable results. Student interest in learning increased significantly, along with classroom attendance and completion rates for homework assignments. Additionally, students became more engaged in classroom interactions and demonstrated improved problem-solving abilities for real-world issues. However, the trial also revealed some challenges. For example, inconsistencies between AI-generated requirement analyses and code implementation occasionally emerged, placing higher demands on teachers' instructional skills. Furthermore, students who overly relied on AI tools sometimes neglected a deeper understanding of foundational concepts. These issues indicate that while AI tools can significantly enhance teaching effectiveness, proper guidance and regulation are essential in instructional design to mitigate potential negative impacts.

2 Related Work

In recent years, the rapid development of artificial intelligence (AI) technology has provided new possibilities for programming education. Particularly in courses such as Python programming, the introduction of AI technology has not only transformed teaching models but also enhanced teaching efficiency and improved students' learning experiences. Relevant studies have explored the application of AI in program-ming education from different dimensions.

Lu, J.[1] studied the application of intelligent programming assistants in Python courses at vocational colleges. The research found that AI technology effectively addresses issues such as low student interest, inefficient teaching, and challenges in personalized guidance. By using intelligent programming assistants, teachers can provide more targeted teaching support, enhancing students' learning enthusiasm and programming skills. Fan, X.Y., et al.[2] explored a teaching method that uses student profiling and dynamically adjusts teaching content to meet individual differences. The study designed tiered practice sessions and precise teacher interventions, which addressed difficulties such as accommodating individual differences, low learning efficiency, and insufficient practical skills in programming courses, significantly im-proving students' learning outcomes and programming abilities. Choi, S., Kim, H.[3] investigated the impact of a large language model-based programming learning environment (LPLE) on students' motivation and programming ability. The study found that LPLE significantly enhanced students' intrinsic motivation as well as their basic psychological needs (autonomy, competence, and relatedness). This, in turn, effectively improved their learning motivation and programming skills, offering a valuable supplement to programming education. Papakostas, C., et al.[4] introduced a rule-based chatbot system designed to offer personalized guidance and support in computer programming education, particularly for Java language. The research evaluated the system with 50 undergraduate students, gathering feedback on user satisfaction, query accuracy, response

time, and personalized guidance. Results showed outstanding performance in user satisfaction, while also identifying limitations in conversational depth and adaptability, which may hinder its engagement in more complex or nuanced discussions. This study highlights the potential of rule-based AI systems in personalizing programming education while addressing their current shortcomings. Wang, R.B., et al.[5] proposed a dual-chain iterative teaching design based on AI-generated content (AIGC), which includes a thinking chain and a problem chain. Teachers use AIGC to construct an initial chain, and students refine their knowledge systems and thinking capabilities through interactions with AI. The study found that this method effective-ly addressed issues such as insufficient thinking skills development, fragmented learn-ing, and challenges in personalized teaching in computer science education. In this process, AIGC acts as both a teaching assistant and a learning companion, injecting new vitality into teaching design and practice. Ye, X., Zhang, W., Zhou, Y., et al.[6] explored the integration of mind mapping and generative AI chatbot learning approaches to improve students' programming performance. This study found that GenAI chatbots positively impacted students' programming self-efficacy and that combining mind mapping with GenAI chatbots significantly enhanced programming learning outcomes. The progressive use of mind maps further amplified these positive effects, offering a structured and effective method for supporting programming education. These studies highlight the applications and advantages of AI technology in programming education.

Ma, S., Wang, J., et al.[7] designed DBox, a generative AI tool built on large language models (LLMs) that uses a step tree to provide personalized support during the problem conceptualization and implementation phases. Unlike tools that directly provide answers, DBox offers dynamic feedback and step-by-step prompts, encouraging independent thinking while enhancing algorithmic programming skills. Experimental results showed that DBox significantly improved learners' programming performance, cognitive engagement, and critical thinking abilities. Additionally, DBox achieved an evaluation accuracy of 98%-100% for code inputs and 86%-92% for natural language inputs. This approach highlights the potential of generative AI tools in scaffolding algorithmic programming learning by fostering deeper problem decomposition and improving educational outcomes.

Yeadon, W., Peach, A., et al.[8] conducted a study comparing the performance of two ChatGPT variants (GPT-3.5 and GPT-4) in a university-level physics programming course with human students. The research found that GPT-4, after prompt engineering optimization, achieved the highest performance with an average score of 81.1%, though it was still significantly lower than the human students' average score of 91.9%. Moreover, evaluators in blind assessments could distinguish AI-generated code from human-generated code with an average accuracy of 85.3%, indicating that AI-generated content still differs from human work in quality and style. Overall, the study highlights that while AI demonstrates strong capabilities in programming tasks, it currently cannot fully match the performance of human students.

Sunday, A. O., & Deriba, F. G.[9] explored the application effects and potential problems of ChatGPT in computer programming education. Their study revealed that ChatGPT can simplify complex programming tasks, help students better understand challenging concepts, and improve beginners' task completion rates in introductory

programming courses. Additionally, ChatGPT was shown to enhance students' interest, motivation, and self-efficacy in programming. However, the research also pointed out its limitations, such as difficulties in handling non-text or non-code-related queries, as well as reliability and accuracy concerns. Overreliance on ChatGPT could negatively impact students' critical thinking, creativity, and problem-solving abilities. This highlights the need for balanced integration of ChatGPT into programming education to maximize its benefits while addressing its limitations.

Messer, M., et al.[10] analyzed automated grading and feedback tools (AATs) in programming education and found that 81% of the tools use fully automated methods, primarily focusing on correctness evaluation (66%) for object-oriented languages through dynamic techniques or static analysis. However, the tools showed shortcomings in feedback quality, maintainability, readability, and documentation evaluation. The study suggested focusing on automated assessments of these aspects, designing more open-ended assignments, and using open datasets to enhance evaluation verifiability and comparability.

Sun, D., et al.[11] investigated the effects of prompt-based learning (PbL) and unprompted learning (UL) using ChatGPT on university students' programming behaviors, interaction quality, and perceptions. The results showed that students in the PbL group were more likely to debug codes actively, pose complex questions, and receive more accurate feedback, while interactions in the UL group were relatively superfi-cial. The study recommended incorporating structured prompts into future teaching designs to optimize the programming learning experience. Sun, D., et al.[12] also ex-amined the impact of the ChatGPT-facilitated programming (CFP) model on stu-dents' programming behaviors and learning outcomes. Results indicated that students in the CFP group demonstrated more proactive debugging behaviors and interactions with AI, resulting in improved programming performance, although there was no statistically significant difference compared to the self-directed programming (SDP) model. Additionally, the CFP model significantly enhanced students' perceptions of ChatGPT, including its usefulness, ease of use, and intention to use. This research provides valuable insights for future teaching designs and the development of AI tools. Akçapınar, G., and Sidan, E.[13] explored the role of AI programming assistants in improving students' exam performance and influencing their acceptance of incor-rect information. The study found that AI assistants significantly improved students' average exam scores but also revealed that 92% of students accepted incorrect an-swers generated by AI, with 61% directly copying and pasting incorrect answers. This highlights the tendency of students to blindly trust AI-generated information, empha-sizing the need for caution when integrating AI tools into learning environments and recommending the development of specialized AI tools to better support student-AI interaction. These studies discuss the impact of AI tools on programming behaviors, interaction, and learning performance.

Güner, H., and Er, E.[14] examined how different teaching interventions influenced the dynamics of student interactions with ChatGPT and their learning outcomes. Through three experiments, the study found that teaching interventions significantly affected the types of AI interactions and post-test scores. Training in prompt design and AI usage effectively improved the quality of interactions and learning outcomes. The

study emphasized the importance of guiding students in effectively using AI, offering practical insights for integrating generative AI tools in education. Garcia, M.B.[15] conducted a rapid literature review and bibliometric analysis to explore the applications and effects of ChatGPT in programming education comprehensively. The research highlighted ChatGPT's significant potential in personalized tutoring, instant feedback, and code generation but also noted challenges such as academic dishonesty, reduced critical thinking, and technological limitations. Therefore, the study recommended the cautious and balanced use of ChatGPT in programming education while formulating policies and teaching strategies to maximize its benefits and mitigate potential drawbacks. These studies examine the dynamics of student interactions with ChatGPT.

The application of AI technology in programming education has become a re-search hotspot. Its advantages lie in improving teaching efficiency, optimizing learn-ing experiences, and supporting personalized teaching. However, the use of AI tools also brings challenges, such as the acceptance of incorrect information, risks to aca-demic integrity, and the potential decline of students' critical thinking skills. The Py-thon programming teaching method proposed in this study, which centers on auto-mated programming tools combined with other AI tools, achieves "learning by doing" and provides an efficient and economical solution to address the lack of practical programming experience among students. Nevertheless, the negative issues discussed in the literature regarding the use of AI technology remain unavoidable and require further exploration in the future.

3 A Novel Teaching Philosophy Based on Automated Programming

With the rise of AI-driven automated programming tools, programming education is evolving with new teaching concepts. These focus on human-machine collaboration, practice-driven learning, and efficiency-first approaches (see Figure 1). Below is a detailed explanation of these core concepts.

3.1 Core Principle 1: Human-AI Collaboration as the Core—Recognize Your Role: If AI Replaces Programmers, Then You Are the Project Manager

With the development of AI-driven automated programming tools (e.g., ChatGPT, GitHub Copilot), the traditional model where programmers write code from scratch is being transformed. AI can efficiently generate code, perform automatic debugging, and optimize algorithms, shifting the focus of programming from specific code details to a higher-level emphasis on thinking and planning. For students, the emphasis of learning programming should shift from being "code executors" to "project managers," focusing on project planning, requirement analysis, resource coordination, and progress control. In this process, students need to clearly define project goals, use AI to generate code frameworks, and optimize or adjust them to complete complex projects beyond their current programming capabilities. By effectively leveraging AI tools, students can not only improve their learning efficiency but also master programming skills and project

management capabilities more efficiently, thereby transforming their approach to learning.

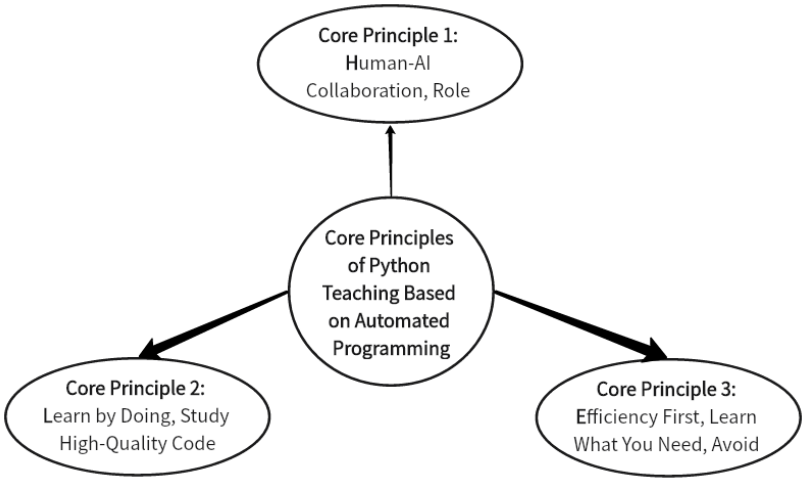

Fig. 1. Core Teaching Philosophy of Automated Programming

3.2 Core Principle 2: Learn by Doing—Just Like Learning a Native Language: Use It First, Then Learn It, and Practice Makes Perfect

In Python learning, the traditional "learn theory first, then practice" model is often time-consuming and inefficient. In contrast, the "learn by doing" philosophy advocates "practice first, learn later," where students engage in practical projects by directly using Python to solve problems and gradually understand related programming concepts through practice. This approach is analogous to learning a native language: people accumulate rules naturally through continuous communication rather than memorizing grammar from the start. Similarly, students can begin with a specific task or goal, directly leveraging AI tools (e.g., ChatGPT, GitHub Copilot, Cursor AI) to generate code and iteratively develop complex functionalities. Automated programming tools provide high-quality, efficient code that adheres to coding standards and incorporates various programming techniques. Through repeated practice, students can learn from these excellent code examples and deepen their understanding of Python's core concepts. This project-driven approach not only enables students to quickly get started with Python but also helps them internalize theoretical knowledge through practice, thereby enhancing their ability to solve real-world problems.

3.3 Core Principle 3: Efficiency First—Learn What You Use, Learn It Quickly, and Apply It Immediately

In real-world engineering, efficiency is paramount. The learning content should align closely with students' practical needs, avoiding unnecessary time waste and ensuring that students can learn and apply their knowledge quickly. Traditional learning methods

often start with a large amount of foundational knowledge, much of which is forgotten if not used for an extended period, leading to low learning efficiency. Guided by the "efficiency first" philosophy, students, with the support of AI, can directly start implementing projects while learning only the programming knowledge, techniques, and relevant algorithms essential for the project. This approach allows them to learn and apply their knowledge immediately. This demand-driven learning methodology not only significantly improves learning efficiency but also enables students to master Python's practical application capabilities more quickly and deeply.

4 Teaching Plan

The teaching plan, as shown in Figure 2, is designed based on a 3-hour practical class and consists of the following six steps:

Step 1: Define the project and solve real-world problems. Teaching begins with a meaningful project designed to meet students' needs, such as data analysis, automated script development, or simple application design. Project requirement documents or related data (e.g., spreadsheets, PDFs) are used as inputs to help students clarify the specific problems to be solved.

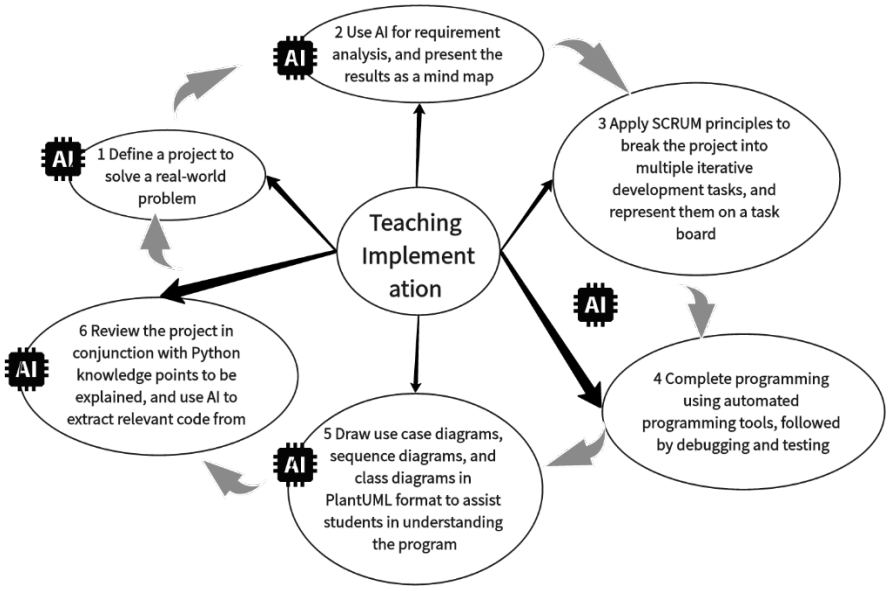

Fig. 2. Teaching Plan.

Step 2: AI-assisted requirement analysis and mind map generation. Leveraging AI tools (e.g., ChatGPT, Claude, Deepseek), the project requirements are thoroughly analyzed to extract key information from the input data. The analysis includes the following elements: project objectives, public data structures, function lists (inputs,

processing logic, outputs), UI design requirements, and project structure requirements. Based on this analysis, tools supporting formats like PlantUML, Markdown, or JSON are used to generate editable mind maps that visually present the project structure and logic. This process helps students quickly understand project requirements and prepares them for subsequent task decomposition and development.

Step 3: Task decomposition and iterative development planning. In this step, students break the project into multiple subtasks based on the function lists and program structure requirements in the mind map, using SCRUM principles and managing them via task boards. Task decomposition follows these principles: tasks should be simple and independent, easy to implement and test; core functions take high priority; and each project's iteration complexity is kept within a reasonable range. Task decomposition is the only step that does not use AI tools. This not only improves the success rate of automated programming but also cultivates students' project planning abilities.

Step 4: AI-automated programming and debugging/testing. Based on the requirement analysis results in the mind map and the task board, students write prompts for programming tasks and use AI automated programming tools (e.g., GitHub Copilot, ChatGPT) to complete code writing. The specific process includes: Writing prompts: Based on subtask descriptions, students generate clear programming requirement prompts. Automated programming: Prompts are input into automated programming tools, which generate code and run the program automatically. Debugging and testing: With the assistance of automated programming tools, students address issues that arise during execution, fix code defects, and ensure the completeness and correctness of the functionality. Through this process, students can efficiently complete complex functionalities with the help of AI and gradually master programming techniques.

Step 5: Generate UML diagrams to aid understanding. To help students understand the structure and flow of the project code, AI tools are used to generate UML diagrams in PlantUML format, including: Use case diagrams: To show the user interaction model of the project functions. Sequence diagrams: To describe the execution flow of the program. Class diagrams: To illustrate the class structure and related relationships in the project. These UML diagrams provide a clear visualization of the program architecture and logic, enabling students to quickly grasp the overall design and execution process of the code.

Step 6: Code review and knowledge point explanation. After completing the project, students review the code in conjunction with the teaching objectives of the Python course. AI is used to extract relevant code snippets from the project for analysis and explanation. Teaching focuses include: Python syntax, such as variables, data types, control structures, and other foundational knowledge. Relevant algorithms and technologies, such as regular expressions, data processing libraries (e.g., pandas), or the use of specific frameworks. Advanced programming techniques: Project code is used to explain advanced techniques for solving real-world problems. This step not only reviews the content students practiced in the project but also deepens their understanding of relevant knowledge points through code explanations.

The above teaching plan fully implements the three core principles: human-AI collaboration, learning by doing, and efficiency first.

5 Case Study of Teaching Plan Implementation

The following practical case demonstrates the implementation of project development and the integration of teaching content through six steps, as shown in Figure 3, with the goal of "developing a Python-based scientific calculator simulation."

Step 1: Define the project and clarify objectives. The project input material is a photograph of a real calculator, requiring students to simulate the main functions of the calculator, including basic operations (addition, subtraction, multiplication, division) and scientific calculations (e.g., trigonometric functions, logarithms, exponents). Through this real-world scenario, students clearly understand the functional goals to be achieved, which provides the foundation for subsequent requirement analysis.

Step 2: AI-assisted requirement analysis and mind map generation. Function identification and extraction: Upload the calculator photograph to ChatGPT. Using its image processing and language analysis capabilities, ChatGPT automatically identifies every button's function on the calculator and provides explanations. Students then manually review the AI-identified functions for feasibility and finalize the function list. Requirement analysis and design output: Based on the function list, ChatGPT further extracts global data structures, UI design requirements, and program structure suggestions.

Mind map generation: ChatGPT outputs the requirement analysis results in PlantUML format, providing the code for generating a mind map. Students input the code into PlantUML-compatible AI tools to produce editable mind maps for subsequent task decomposition and development.

Step 3: Task decomposition and iterative development planning. Students break down the functions in the mind map into seven subtasks, with each task covering several functional modules of the calculator buttons. Due to the large number of button functions, the functions of every two rows of buttons are grouped into one iterative section to control the complexity of implementation. The entire development process is divided into five iterations, gradually achieving full functionality.

Step 4: AI-based automated programming and debugging. Prompt design: Based on the mind map and the functional requirements of each iteration, students write detailed programming task prompts and input them into automated programming tools (e.g., Cursor.AI). The prompts include functional descriptions, input-output requirements, and implementation details to ensure the accuracy of AI-generated code. Code generation and debugging: Using Cursor.AI, students generate code to incrementally implement the functionality of each iteration. With AI assistance, students debug the code and resolve issues encountered during execution, ensuring the functionality's correctness and completeness. Through this process, students can quickly complete complex code implementation while gaining a deeper understanding of Python applications during debugging.

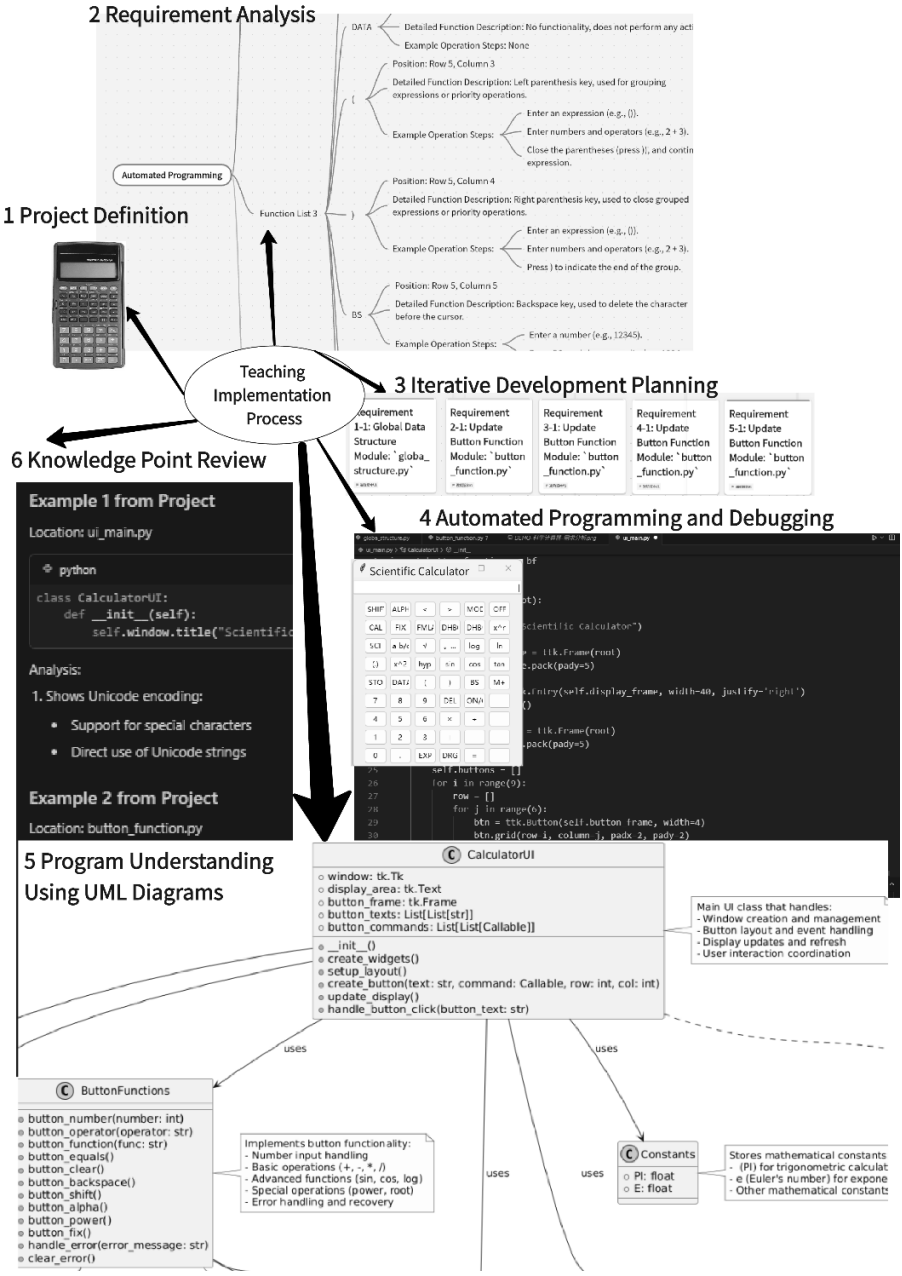

Fig. 3. Case Study of Teaching Plan Implementation

Step 5: UML diagram generation and program understanding. To help students quickly understand the code's logical structure and execution flow, ChatGPT is used to generate PlantUML-format text for class diagrams, sequence diagrams, and use case

diagrams. The text is then entered into the PlantUML website to generate UML diagrams. These UML diagrams clearly display the program's composition and logic, enabling students to quickly grasp the overall design of the project.

Step 6: Code review and knowledge point explanation. Based on the teaching objectives of the Python course, ChatGPT is used to extract code snippets related to knowledge points from the implemented code for analysis and explanation. For example, when explaining dictionary comprehensions, the code snippet periods = {f"{i}-{i+3}": 0 for i in range(0, 24, 3)} is selected, guiding students to understand multiple knowledge points, such as dictionary comprehensions, formatted strings, the use of the range function, and the relationship between loop variables and dictionary creation. For each knowledge point, due to the limited code snippets, ChatGPT generates additional examples based on the snippets to help students reinforce knowledge from different perspectives. Through this approach, students not only understand the knowledge points but also learn efficient programming techniques.

Through this practical class project, students can efficiently complete the full process of project development—from requirement analysis to code implementation—with AI assistance, while mastering core Python knowledge and application skills through practice.

6 Teaching Effectiveness Evaluation

A new teaching method was applied to the students of the Spring 2025 Artificial Intelligence program. The learning outcomes were evaluated through a questionnaire survey, and the findings are summarized as shown in Table 1.

Table 1. Survey on Teaching Effectiveness of the New Teaching Method.

Heading level	Example	Font size and style
Learning Approach	Preference for the "practice first, learn later" approach; perception of the method's relevance to real-world needs	78%
Learning Efficiency	Reduction in learning time; more efficient mastery of knowledge points; avoidance of irrelevant content	56% (learning time was not reduced)
Improvement in Programming Skills	Acquisition of more efficient programming techniques through AI-generated code; faster mastery of Python core knowledge	76%
Project Achievement	Confidence and sense of accomplishment from completing projects	95%
Project Completion	Ability to independently reproduce the course's practical case studies	65% (uncertainty in AI-generated code)
Learning Threshold	Ease of use of AI tools; ability to quickly adapt to the new learning method	45%

Knowledge Point Difficulty	Difficulty level of knowledge points; ability to digest knowledge points in each lesson	34% (understanding high-quality programs is challenging)
Task Complexity	Appropriateness of task workload; ability to complete tasks within the allotted time; clarity of task decomposition	23% (task complexity was not reduced)

The new teaching method, centered on project-driven and AI-assisted learning, effectively enhanced students' interest in learning and practical abilities. A total of 78% of students preferred the "practice first, learn later" approach and found it more relevant to real-world needs; 95% of students felt confident and accomplished in completing projects, recognizing the practical application skills they had gained. However, certain areas require improvement.

Only 45% of students found AI tools easy to use, indicating initial difficulties in adaptation. Only 65% of students were able to successfully reproduce the course's practical case studies, with the uncertainty of AI-generated code leading to debugging challenges for some. Furthermore, only 34% of students felt that the difficulty level of the knowledge points was reasonable, with most finding the knowledge points too challenging to understand and master. Only 23% of students believed the task complexity was appropriate, with most considering it too high due to difficulties in requirement analysis and code debugging.

Additionally, 56% of students reported limited improvement in learning efficiency, suggesting that the systematic explanation of knowledge points and transferability of skills need to be enhanced. Overall, the new method is fast-paced and effective but requires further refinement to lower the learning threshold, optimize the explanation of knowledge points, and improve task decomposition. These improvements will better meet students' needs and enhance the overall teaching effectiveness.

7 Discussion

This paper proposes a new AI-tool-centered, project-driven teaching method for Python. By integrating AI-based requirement analysis, code generation, debugging optimization, and visualization tools, it constructs a full-chain teaching model from requirement analysis to code implementation and theoretical reinforcement. This method effectively enhances students' learning interest and practical abilities. However, the study also revealed several issues. These issues stem primarily from the following reasons: Students are not familiar with the operation of AI tools and lack systematic guidance. The uncertainty of AI-generated code increases debugging difficulty. The gradient design of task decomposition and knowledge point explanation is insufficiently detailed, failing to fully account for students' varying ability levels.

To address these problems, future research could focus on the following directions: Optimize training for the use of AI tools by designing step-by-step teaching modules to help students quickly become proficient. Incorporate specialized training focused on debugging and understanding AI-generated code, considering its unique characteristics. Enhance the design of task decomposition and knowledge point explanations, reducing

complexity and strengthening systematic approaches to better meet the needs of students at different levels.

8 Conclusion

A Python programming teaching method based on AI automated programming was proposed, with the core concept of "practice first, then learn." By integrating multiple AI tools, the teaching process was optimized, significantly enhancing students' learning interest and practical abilities. This method was particularly effective in helping students understand the overall architecture of complex projects, perform functional decomposition, and implement solutions, while achieving a closed-loop teaching model from requirement analysis and code generation to UML visualization, offering a new approach to traditional teaching.

By introducing AI automated programming tools, the method addressed the disconnect between theory and practice in traditional Python teaching, providing an efficient and economical pathway for students in vocational colleges and open universities, with wide-ranging practical applications.

The study identified two key challenges: the uncertainty in the accuracy and debuggability of AI-generated code, which poses the greatest risk to the successful implementation of the teaching approach, and the difficulty in crafting high-quality prompts for complex projects, which is a major obstacle for both students and teachers. Therefore, future research should focus on these two critical issues to ensure that the AI-automated programming teaching model can be more efficiently and reliably applied in practical educational scenarios.

Acknowledgments

This paper is funded by the (AA-02-2025-01-16-02) Research on SLAM Visual Dataset Construction and Key Technologies for Autonomous Mobile Platforms in Home Environments.

References

1. Lu, J. (2024). Application of artificial intelligence technology in curriculum teaching in vocational colleges: A case study of Python programming course. University Education, 18, 89–92.
2. Fan, X. Y., Yao, L., & Cui, X. L. (2024). Exploration of a Python teaching model for cultivating personalized programming ability. Computer Education, 12, 182–187.
3. Choi, S., & Kim, H. (2025). The impact of a large language model-based programming learning environment on students' motivation and programming ability. Education and Information Technologies, 30, 8109–8138.
4. Papakostas, C., Troussas, C., Krouska, A., & Sgouropoulou, C. (2024). A rule-based chatbot offering personalized guidance in computer programming education. In Sifaleras, A., & Lin,

F. (Eds.), Generative Intelligence and Intelligent Tutoring Systems (Vol. 14799). Springer, Cham.
5. Wang, R. B., Li, M. H., Song, W., et al. (2024). Role positioning and functional extension of AIGC in empowering basic computer education: A teaching design and practice based on dual-chain iteration. Computer Education, 10, 159–163+168.
6. Ye, X., Zhang, W., Zhou, Y., et al. (2025). Improving students' programming performance: An integrated mind mapping and generative AI chatbot learning approach. Humanities and Social Sciences Communications, 12, 558.
7. Ma, S., Wang, J., Zhang, Y., Ma, X., & Wang, A. Y. (2025). DBox: Scaffolding algorithmic programming learning through learner-LLM co-decomposition. In Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '25). Yokohama, Japan.
8. Yeadon, W., Peach, A., & Testrow, C. (2024). A comparison of human, GPT-3.5, and GPT-4 performance in a university-level coding course. Scientific Reports, 14, 23285.
9. Sunday, A. O., & Deriba, F. G. (2023). Enhancing computer programming education using ChatGPT: A mini review. ACM.
10. Messer, M., Brown, N. C., Kölling, M., & Shi, M. (2023). Automated grading and feedback tools for programming education: A systematic review. ACM Transactions on Computing Education, 24, 1–43.
11. 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.
12. 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.
13. Akçapınar, G., & Sidan, E. (2024). AI chatbots in programming education: Guiding success or encouraging plagiarism. Discover Artificial Intelligence, 4(87).
14. Güner, H., & Er, E. (2025). AI in the classroom: Exploring students' interaction with ChatGPT in programming learning. Education and Information Technologies.
15. Garcia, M. B. (2025). Teaching and learning computer programming using ChatGPT: A rapid review of literature amid the rise of generative AI technologies. Education and Information Technologies.