World Journal of Educational Studies
Print ISSN: 2959-9989
Online ISSN: 2959-9997
DOI: https://doi.org/10.61784/wjes3194

GENERATIVE AI-ASSISTED TEACHING FOR MICROCONTROLLER PROGRAMMING AND DEBUGGING: MODEL DESIGN AND PRACTICE

Yuan Tan[1], QingYuan Zhen[2], XinDi Wang[3], HongLiang Gao[1*]
[1]*School of Electrical Engineering and Automation, Hubei Normal University, Huangshi 435002, Hubei, China.*
[2]*School of Physics and Electronic Information, Jiangsu Second Normal University, Nanjing 211200, Jiangsu, China.*
[3]*School of Artificial Intelligence and Computer Science, Hubei Normal University, Huangshi 435002, Hubei, China.*
[*]*Corresponding Author: HongLiang Gao*

Abstract: The course "Principles and Applications of Microcontrollers" is highly practice-oriented and covers a broad range of knowledge domains. In program writing and debugging, students commonly face a steep learning curve, time-consuming debugging, and limited access to personalized instructor guidance. Taking generative artificial intelligence (Generative AI) as an entry point, this paper builds an AI-assisted teaching model for microcontroller programming practice based on three functional modules, namely code-skeleton generation, compilation/logic-error diagnosis, and personalized question answering, and conducts concrete teaching-practice exploration in terms of instructional design, laboratory cases, and implementation procedures. Preliminary teaching practice indicates that the model helps shorten students' debugging time and improve learning autonomy and classroom satisfaction. At the same time, it also carries risks such as the reliability of AI-generated code and students' over-reliance on AI, which require instructors to guide and regulate through instructional organization and assessment design. The teaching model and implementation pathway proposed in this paper can serve as a reference for teaching reform in similar practice-oriented courses.
Keywords: Principles and Applications of Microcontrollers; Generative artificial intelligence; Programming instruction; Debugging competence; Teaching-model reform

1 INTRODUCTION

"Principles and Applications of Microcontrollers" is a core professional foundation course for majors such as electronic information, automation, and mechatronics, and it is characterized by a deep integration of theory and practice. The course requires students not only to master the fundamental principles of microcontroller hardware architecture, instruction systems, interrupt mechanisms, timers/counters, serial communication, and analog-to-digital conversion, but also to develop strong C-language programming skills and hardware debugging ability, so that they can independently complete the full engineering-practice cycle from requirements analysis and code writing to on-board debugging and fault troubleshooting. Compared with purely theoretical courses, this course places higher demands on students' engineering-practice ability and capacity for autonomous learning.

However, long-term teaching practice has revealed an evident gap in the transition from "understanding principles" to "hands-on implementation" for most students. On one hand, students have a weak grasp of low-level programming details such as register configuration and peripheral initialization, and often do not know where to start when writing code. On the other hand, the debugging stage relies heavily on trial and error: when students encounter compilation errors or abnormal runtime behavior, they frequently lack a systematic troubleshooting approach and can only repeatedly consult the instructor. Given the limited classroom time and instructor bandwidth, it is difficult to provide timely, personalized guidance to every student, resulting in low learning efficiency and heightened frustration among learners.

In recent years, generative artificial intelligence technologies represented by large language models (LLMs) have advanced rapidly, demonstrating strong capabilities in code generation, code explanation, and error diagnosis, and have been widely applied in software-engineering education [1-2]. Introducing generative AI into microcontroller programming and debugging instruction is expected to alleviate the teaching pain points described above to some extent. Students can use AI to quickly obtain code-skeleton suggestions, explanations of error messages, and personalized answers to questions, thereby devoting more energy to understanding principles and engineering innovation. At the same time, however, AI-assisted teaching may also introduce new problems such as "valuing results over process" and "over-reliance on AI" [3-4]. How to strike a balance between efficiency gains and student development is therefore the central concern of this paper.

Existing research at home and abroad has already explored the application of generative AI in programming-course instruction to some extent. One line of work focuses on using large models to automatically generate example code and assist students in understanding programming syntax, confirming the positive role of AI in lowering entry barriers [5-7]. Another line of work focuses on the potential academic-integrity risks and the weakening of critical thinking that AI assistance may bring, arguing that instructional design should constrain the scope of AI use [8-9]. However, most

existing studies concentrate on general-purpose programming-language (e.g., Python, Java) teaching scenarios [10-12]; for practice-oriented courses such as microcontroller programming, which combine both software programming and hardware debugging, systematic discussion of the application model, implementation pathway, and effect-evaluation system of AI-assisted teaching remains scarce [13].

Building on the above background and combined with the actual teaching of the "Principles and Applications of Microcontrollers" course, this paper proposes a teaching model that uses generative AI as an assisting means, with the cultivation of students' programming and debugging ability as its core goal. The model is systematically elaborated from the perspectives of instructional-model design, laboratory-case organization, implementation-procedure planning, and structured-questioning training, and the preliminary effects and existing problems observed in teaching practice are analyzed and discussed, with the aim of providing a reference for teaching reform in similar practice-oriented courses.

The main contributions and features of this paper can be summarized in three points. First, an overall AI-assisted teaching framework spanning three levels, namely knowledge learning, programming practice, and debugging optimization, is constructed. Second, combined with the 51-series microcontroller and Keil C51 teaching platform, six laboratory projects of progressively increasing difficulty are designed together with corresponding AI-assistance intervention strategies, and a concrete compilation-error-diagnosis case is provided. Third, a structured AI-questioning training method for students is proposed to constrain the depth of AI-assistance intervention and guard against the risk of over-reliance.

2 CURRENT STATE AND PROBLEMS IN MICROCONTROLLER COURSE TEACHING

2.1 Key Problems in Traditional Teaching Practice

Microcontroller programming involves extensive low-level details such as register configuration, timing control, and interrupt vectors, which differ substantially from the high-level-language programming students have previously learned. Taking the 51-series microcontroller as an example, students must not only understand C-language syntax itself but also master the physical meaning of special function registers (SFRs), bit-manipulation methods, and the correspondence of the interrupt vector table, all of which are hardware-related knowledge. Beginners often need to spend a great deal of time consulting datasheets and understanding peripheral operating mechanisms before they can write a runnable basic program. The learning curve is steep and prone to inducing anxiety, and some students even develop resistance to the course after feeling "lost" in the very first laboratory session.

Beyond this high entry barrier, debugging microcontroller programs is itself time-consuming and characterized by delayed feedback. Debugging involves not only code-logic inspection but also multiple factors such as hardware wiring, clock configuration, and register settings, making fault localization considerably difficult. When students encounter issues such as compilation errors or on-board behavior not matching expectations, they often cannot tell whether the problem lies in software logic or hardware connections, and easily fall into a state of blind trial and error. Under the traditional teaching model, students rely mainly on in-person instructor guidance when problems arise, but limited classroom time and student-to-teacher ratios lead to clearly delayed feedback: during a single 45-minute laboratory session, the instructor often must circulate among dozens of students, leaving very limited time for targeted guidance to any individual student. Some students, stuck on a single fault point for a long time, interrupt their learning progress and even lose confidence to keep trying [14].

Compounding these difficulties, personalized guidance is hard to extend to all students, and engineering-practice ability together with autonomous-learning awareness remain insufficiently cultivated. Constrained by class hours and instructor bandwidth, traditional teaching mostly organizes laboratory instruction through uniform pacing and centralized lectures, making it difficult to provide personalized guidance for students with different foundations and learning paces. This commonly results in a polarization phenomenon in which weaker students cannot keep up while stronger students are underfed. Some students, worried about taking up too much of the instructor's time or afraid of exposing their weak foundations, are reluctant to ask questions proactively, so accumulated knowledge gaps go unaddressed, further widening the gap in ability among students. At the same time, some students, during experiments, are accustomed to copying reference code or directly reusing programs that others have already debugged, lacking the awareness to independently analyze problems and proactively explore solutions, which is detrimental to the long-term cultivation of engineering-practice ability and autonomous-learning ability. This phenomenon is especially prominent in more difficult sessions such as comprehensive project design: when facing complex tasks, some students tend to search for a ready-made universal code rather than decomposing the task and implementing it step by step, resulting in a large gap between the depth of code understanding reflected in lab reports and their actual coding ability, which also creates hidden risks for later, higher-level practice sessions such as course design and capstone projects [13].

2.2 Necessity of Introducing AI-Assisted Teaching

The analysis above shows that the problems present in the traditional microcontroller teaching model, spanning entry barrier, debugging feedback, personalized guidance, and autonomous-learning cultivation, are in essence all closely related to the limited nature of teaching resources, particularly instructor bandwidth and classroom time. Generative AI technology, with its round-the-clock instant responsiveness, capacity to serve many students simultaneously, and ability to explain specific error messages, can effectively supplement precisely the areas that instructor bandwidth struggles to

cover, becoming an increment to, rather than a replacement for, traditional teaching resources. This provides the practical basis and entry point for constructing the AI-assisted teaching model discussed in the following section.

3 DESIGN OF THE AI-ASSISTED TEACHING MODEL

3.1 Design Principles

The teaching model proposed in this paper follows the basic principle of "AI as an aid, competence as the foundation," meaning that generative AI serves as a tool to assist student learning and to support the cultivation of students' programming and debugging ability, rather than replacing students' independent thinking process. Specifically, the instructional-model design balances three considerations. First, students are required to complete independent thinking and an initial attempt before turning to AI, thereby avoiding the tendency to seek answers directly without prior effort. Second, students are guided to analyze, verify, and critically evaluate the suggestions given by AI rather than accepting them wholesale. Third, over-reliance on AI is constrained through the design of instructional sessions and corresponding adjustments to assessment methods.

As shown in Figure 1, the AI-assisted teaching model constructed in this paper comprises three instructional layers, namely knowledge learning, programming practice, and debugging optimization, with a generative-AI assistance engine running through all three layers. Through three functional modules, namely code-skeleton generation, compilation/logic-error diagnosis and explanation, and personalized question answering with learning-path recommendation, the model supports students' knowledge internalization, programming practice, and debugging optimization, ultimately driving a spiral improvement in students' programming ability and autonomous-learning ability.

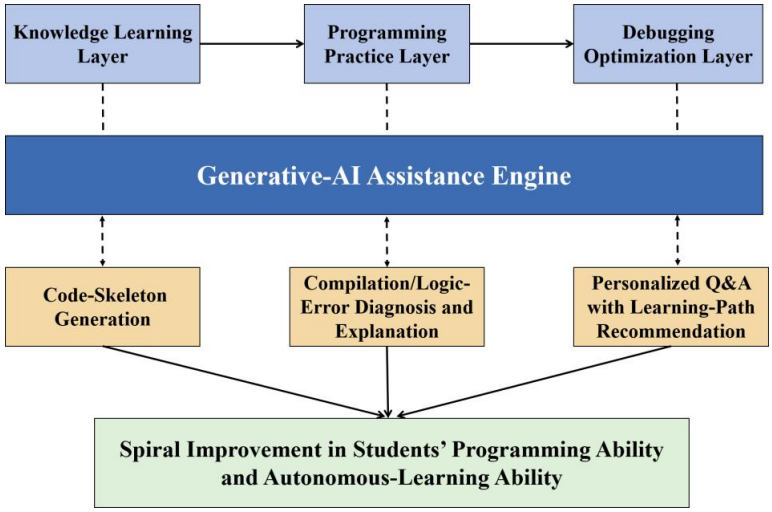

Figure 1 Framework of the AI-Assisted Microcontroller Programming Teaching Model

3.2 Key Functional Modules of AI-Assisted Teaching

(1) Code-skeleton generation. Once students have clarified the laboratory task and technical requirements, they can use AI to generate a code skeleton based on 51-series-microcontroller register operations within the Keil C51 development environment, including port initialization and timer/interrupt-service-function templates. This helps students quickly build the overall program structure and shifts the learning focus from memorizing fixed syntax to understanding configuration logic. It should be emphasized that the AI-generated framework typically contains only the skeleton of key configurations together with necessary comments; the core business logic, such as specific duty-cycle calculations or state-transition conditions, must still be completed by students according to the experimental requirements, so that students are prevented from obtaining a complete, directly runnable program.

(2) Compilation and logic-error diagnosis and explanation. When a student's program produces a compilation error or abnormal runtime behavior, the error message or a description of the phenomenon can be submitted to AI, which returns a possible-cause analysis and troubleshooting suggestions. Rather than directly telling students how to fix the problem, the teaching approach emphasizes guiding students to use AI's explanation to understand the underlying cause of the error, such as an unenabled clock, interrupt-priority conflicts, improper pointer use, or a mismatch between the interrupt number and the timer, thereby strengthening students' ability to independently troubleshoot similar problems in the future. Teaching observation shows that this module achieves relatively high diagnostic accuracy for compile-time errors, such as syntax errors or undefined symbols, whereas for runtime logic errors, such as intermittent anomalies caused by improper timing configuration, it often requires repeated clarification based on the phenomena described by the student. This places greater demands on students' ability to accurately describe fault symptoms, which is also an important motivation for the structured-questioning training discussed in Section 4.5.

(3) Personalized question answering and learning-path recommendation. For students with different knowledge weaknesses, AI can provide one-on-one question answering and, combined with a student's questioning history and experiment-completion status, recommend corresponding knowledge-review content or extension exercises. This partially compensates for the difficulty that traditional one-to-many classroom teaching has in personalizing instruction. For example, a student who repeatedly asks questions about interrupt-priority issues can be guided to review the priority-arbitration rules and IP/IE register configuration covered in the interrupt-system chapter, while students with a strong foundation who complete assigned tasks early can be recommended extension-oriented comprehensive-project directions, achieving differentiated instruction that ensures a baseline for all students while leaving room for advanced learners to progress further.

The three modules do not operate in isolation but are dynamically coupled with students' actual programming progress: code-skeleton generation dominates in the early stage of an experiment, helping students cross the entry barrier; once the debugging stage begins, error diagnosis and explanation becomes the most frequently used module; and personalized question answering runs throughout the experiment, addressing students' need for the immediate identification and filling of gaps in principle-level knowledge. The main differences between the three modules and the traditional teaching model are summarized in Table 1.

Table 1 Comparison between the Traditional Teaching Model and the AI-Assisted Teaching Model

Dimension	Traditional Teaching Model	AI-Assisted Teaching Model
Code entry	Instructor explanation + imitation of reference code	AI-assisted generation of a code skeleton; students understand and complete it independently
Error troubleshooting	Relies on in-person instructor Q&A; feedback is delayed	AI provides instant explanation of error causes; instructor focuses on difficult questions
Guidance approach	Uniform pace, centralized lecturing	AI provides personalized Q&A; instructor balances the whole class and individuals
Student role	Relatively passive knowledge recipient	Active learner who questions, verifies, and reflects
Teacher role	Lecturer + full-process question-answerer	Instructional designer + AI-use guide + gatekeeper for difficult problems

4 TEACHING IMPLEMENTATION AND CASE DESIGN

4.1 Teaching Subjects and Course Arrangement

This teaching model was applied to undergraduate students majoring in Electrical Engineering and Automation. The teaching class comprised 60 students, randomly and evenly divided into two groups of 30 each: one group adopted the traditional teaching model (hereafter the "traditional-teaching group"), and the other adopted the AI-assisted teaching model proposed in this paper (hereafter the "AI-assisted-teaching group"). Both groups were taught by the same instructor using the same syllabus, the same laboratory task sheets, and the same acceptance criteria, differing only in whether AI assistance was introduced during the laboratory sessions, so as to control as far as possible for confounding variables such as instructional quality and task difficulty. The course used the 51-series microcontroller, mainly the STC89C52/STC15 series, as the hardware teaching platform, paired with Keil C51 as the integrated development and compile/debug environment. Laboratory sessions were scheduled twice a week for a total of six sessions, advancing in parallel with the theoretical instruction.

4.2 Laboratory Projects and AI-Assistance Design

Table 2 Correspondence between Laboratory Projects and AI-Assistance Sessions

No.	Laboratory Project	Core Knowledge Points	Main AI-Assistance Sessions
1	Circuit schematic drawing	Minimum-system circuit design for microcontrollers; schematic-drawing conventions (Proteus/Altium)	Circuit-design consultation; schematic-convention checking
2	Seven-segment-display interrupt control	External/timer interrupt configuration; display-driving principle	Error diagnosis (interrupt-configuration troubleshooting)
3	Digital stopwatch display	Timer T0/T1 interrupt timing; start/pause/reset via keypress; display refresh	Code-skeleton generation + compilation-error diagnosis
4	Two-machine serial communication	UART protocol; synchronized two-device data transmission/reception; SBUF/SCON configuration	Error diagnosis + personalized Q&A
5	8-digit digital display	Dynamic-scanning principle; multiplexed driving; scan frequency and blanking handling	Personalized Q&A + solution discussion (timing optimization)
6	Comprehensive test (final assessment)	Integrated application of knowledge and skills across projects	No AI use (serves as a check of learning outcomes)

The six laboratory sessions were, in order, circuit-schematic drawing, seven-segment-display interrupt control, digital stopwatch display, two-machine serial communication, 8-digit digital display, and a comprehensive test serving as the final assessment, following a progressively advancing instructional logic that moves from hardware-design awareness,

to single-point function implementation, to multi-point function integration, to multi-device coordination, to complex timing control, and finally to a comprehensive-ability examination. This paper further specifies the main points at which AI assistance intervenes in each project, as shown in Table 2, to ensure that AI assistance is used with clear purpose, avoiding inappropriate use of AI in simple tasks or in the assessment session. It should be specially noted that the sixth session, the comprehensive test, served as the final assessment; students in both groups were required to complete it independently on-site without AI assistance. The purpose was not to compare the two groups' ability to use AI on the spot, but to examine whether the programming and debugging ability that the AI-assisted-teaching group had built up with AI's help in the preceding five sessions had genuinely been internalized as their own competence, an important design consideration for the effectiveness evaluation in Section 5.

4.3 Teaching Case Example: Digital Stopwatch Display Experiment

Taking the digital stopwatch display experiment as an example, this section illustrates the concrete instructional organization of AI-assisted code generation and debugging. The experiment requires students to use Timer T0 interrupts to generate a precise timing reference, for instance one interrupt every 50 ms, accumulate a count within the interrupt-service function, complete one second of timing after every 20 counts, and then drive the seven-segment display to refresh the minutes and seconds, while also implementing start/pause/reset functions via keypresses. It is an important experiment in which students first encounter the integrated programming mindset of combining an interrupt-service function, global variables, and software timing, and it also forms the ability foundation for later, more complex projects such as the 8-digit digital display and the comprehensive test.

When independently writing their initial version of the code, students commonly make errors such as non-standard interrupt-service-function declarations, mismatches between the interrupt number and the timer used, and improper scope or type usage for timing variables. Below is a real teaching scenario recorded during instruction: the interrupt-service function written by a student was as follows (excerpted for illustrative teaching purposes only, not a standard example):

```
Void Timer0_ISR() interrupt 3
{
    TH0 = 0x3C;    TL0 = 0xB0;     // 50 ms timing preset
    tick_cnt++;
    if (tick_cnt >= 20)
    {
        tick_cnt = 0;
        sec++;
        if (sec >= 60) { sec = 0; min++; }
    }
}
```

When the student compiled the code in Keil µVision, the following error messages were obtained (example):

```
** ERROR C231 IN LINE 21 OF main.c: 'Timer0_ISR': redefinition
main.c(27): warning C206: 'sec': missing function-prototype
main.c(27): warning C206: 'min': missing function-prototype
```

The student then submitted the error message, the relevant code snippet, and a description of the phenomenon (the program fails to compile) to AI for help. In the teaching process, students were required to ensure their questions included the hardware platform (STC89C52/Keil C51), the original error text, the relevant code snippet, and any operations already attempted, rather than submitting only a vague query such as "why does my program have an error."

The diagnostic suggestions given by AI generally included the following points, as summarized by the instructor rather than quoted verbatim from the AI's original output. First, interrupt number 3 corresponds to Timer 1, not Timer 0; on the 51-series microcontroller, the interrupt number for Timer 0 should be 1, and this mismatch between the interrupt number and the timer used is the direct cause of the redefinition and interrupt-vector conflict. Second, the timing variables sec and min are used within the interrupt-service function but are not declared and initialized globally at the top of the file, so the compiler issues a warning about a missing function prototype; it was suggested that the declaration "unsigned char sec = 0, min = 0, tick_cnt = 0;" be added at the top of the file.

Based on the cause analysis given by AI, the student cross-checked the correspondence in the 51 interrupt-vector table against the classroom material and the datasheet, confirmed that the AI's analysis was correct, and then revised the code, recording the complete process from error phenomenon, to AI's analysis points, to the datasheet-verification process, to the final revision, in the lab report. The purpose of this recording requirement is to prevent students from skipping the verification step and adopting AI's suggestion directly, thereby ensuring that students still master the core knowledge point of interrupt-vector configuration rather than merely getting the program to run.

It is worth noting that timing accuracy in this experiment is also closely related to the crystal-oscillator frequency: if a student's development board does not use a 12 MHz crystal, directly reusing the timer initial values given by AI, such as 0x3C and 0xB0, will cause the stopwatch to run inaccurately, and students need to be guided to recalculate the timer initial values based on the actual crystal frequency. This is another representative detail for training students to verify rather than copy AI's suggestions.

4.4 AI Prompt Design and Structured-Questioning Training

The way students phrase their questions to AI largely determines the effectiveness of the AI-assisted feedback. Teaching practice has found that, without guidance, students' initial questions are often vague, lack context, and directly ask for a complete answer, which tends to lead AI to give generic suggestions that are poorly matched to the specific hardware platform, and can even induce students to directly copy and paste an entire block of AI-generated code. For this reason, the course includes a dedicated approximately 20-minute structured-questioning training session, explaining the basic elements of structured questioning to students and holding a classroom discussion around the comparative examples shown in Table 5.

A structured question should generally include four elements. First, the hardware platform and development environment, for example the STC89C52 microcontroller, Keil C51, and the crystal frequency. Second, the specific requirement or problem phenomenon, such as the original error text or the abnormal behavior observed. Third, the code snippet already completed and the troubleshooting methods already attempted. Fourth, the expected role of the AI's response, for instance whether it should only provide a hint on the approach rather than directly provide complete code.

The teaching also explicitly requires that AI serve only as an auxiliary analysis and hint tool, and that students must not ask AI to directly output a complete block of experiment code that can be submitted as-is. For suggestions given by AI, students must first write out or verbally restate their understanding of the underlying principle in a draft notebook before modifying their code, so as to reinforce the independent-thinking step. The questioning training and usage norms described above constrain, to some extent, the depth of AI-assistance intervention, and constitute a key design element that distinguishes the teaching model proposed here from a laissez-faire use of AI.

4.5 Teaching Case Example: 8-Digit Digital Display Experiment

The 8-digit digital display experiment is technically the most demanding of the six sessions and best exemplifies the solution-discussion characteristic of AI-assisted teaching. Building on the two preceding experiments, digital stopwatch display and two-machine serial communication, this experiment requires students to expand the number of display digits from one to eight, driving an 8-digit common-cathode/common-anode display via dynamic scanning, or multiplexing. Students must comprehensively weigh factors such as scan frequency, the timing of digit-select and segment-select switching, and display blanking, making it a concentrated test of students' systematic timing-design ability.

In this session, the instructional organization differs from the previous ones: the instructor no longer provides a concrete code skeleton but only gives the functional requirement, for example achieving stable 8-digit display with no obvious flicker or ghosting, requiring students first to independently design an overall approach, such as the scan frequency, the digit-select switching order, and how blanking timing should be arranged, and then submit their design idea to AI for a solution-level discussion. For example, students may have AI evaluate whether the chosen scan frequency is reasonable, or whether there is a risk of flicker or ghosting, rather than directly asking AI for the complete project code. This design aims to further shift AI's role from code provider to solution-discussion partner, training students' systems-design ability at a higher level of abstraction. Teaching observation found that most students encounter phenomena such as ghosting and afterglow crosstalk between adjacent digits in this session, which are usually related to the timing coordination between digit-select switching and segment-select data updates. The diagnostic suggestions given by AI, such as blanking the display before switching digit-select to avoid old and new segment codes overlapping, need to be verified by students against digit-select and segment-select waveforms measured with an oscilloscope or logic analyzer, further reinforcing the closed loop of AI suggestion, empirical verification, and principle understanding.

5 PRELIMINARY ANALYSIS OF TEACHING EFFECTIVENESS

5.1 Comparison of Student Debugging Efficiency

As shown in Figure 2 across the six laboratory projects, the average debugging time of the AI-assisted-teaching group was shorter than that of the traditional-teaching group to varying degrees, and the gap tended to widen overall as experimental complexity increased, offering an initial indication of the efficiency advantage AI assistance provides in troubleshooting more complex problems. In entry-level experiments such as circuit-schematic drawing and seven-segment-display interrupt control, the time gap between the two groups was relatively limited, presumably because such experiments have relatively single fault points and instructors can also respond fairly quickly under the traditional model. In experiments involving more coordination among multiple steps and more difficult fault localization, such as two-machine serial communication and 8-digit digital display, the time gap widened noticeably, consistent with the theoretical expectation that AI can quickly pinpoint clues in multi-factor cross-cutting faults and shorten students' blind trial-and-error time.

The sixth comprehensive-test session is particularly noteworthy because both groups were required to complete this session independently on-site without AI assistance, the fact that the AI-assisted-teaching group's completion time remained shorter than the traditional-teaching group's even under this AI-free condition indicates that the time gap does not simply come from AI's on-the-spot assistance, but more likely reflects that the debugging approach and knowledge structure the AI-assisted-teaching group built up with AI's help in the preceding five sessions has genuinely been internalized as their own competence. This constitutes key evidence supporting the effectiveness of the teaching model

proposed in this paper. Future research will conduct further attribution analysis combined with specific fault types to further clarify the relative contributions of real-time AI assistance versus internalized competence.

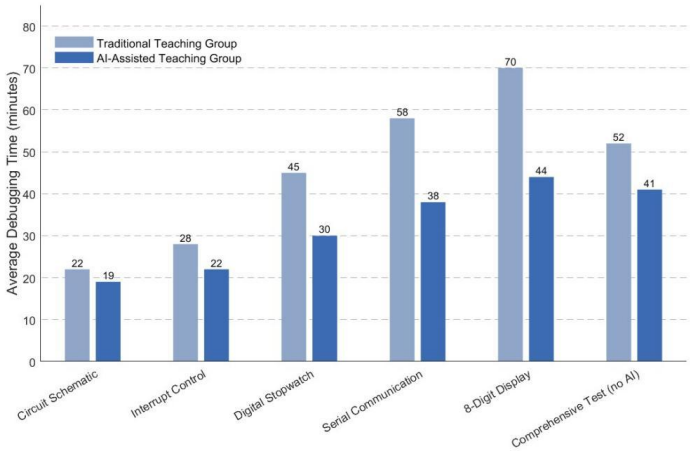

Figure 2 Comparison of Average Debugging Time across Laboratory Projects

5.2 Comparison of Student Experiment Scores

Table 3 presents a comparison of the score distributions of the traditional-teaching group and the AI-assisted-teaching group in the sixth comprehensive test, the final assessment in which neither group used AI. Because this assessment session itself does not permit AI use, score differences are more appropriately interpreted as differences in the two groups' genuine ability level rather than differences in on-the-spot AI-assistance performance, which is also the core rationale for designating the comprehensive test as an AI-free assessment session. Based on the distribution shown in Table 3, the proportion of students in the excellent and good score ranges was higher in the AI-assisted-teaching group, while the proportion in the passing-or-below range was lower, showing an overall shift toward the higher-score end. An independent-samples t-test was conducted on the two groups' comprehensive-test scores $t(58) = 2.23$, $p = 0.03$, confirming that the difference reached statistical significance.

Table 3 Comparison of Final Comprehensive-Project Score Distributions

Score Range	Traditional-Teaching Group (n=30)	AI-Assisted-Teaching Group (n=30)	Proportion Trend
90–100 (Excellent)	3	7	Increase
80–89 (Good)	9	12	Increase
70–79 (Fair)	11	8	Decrease
60–69 (Pass)	5	3	Decrease
Below 60 (Fail)	2	0	Decrease

5.3 Student Satisfaction Survey

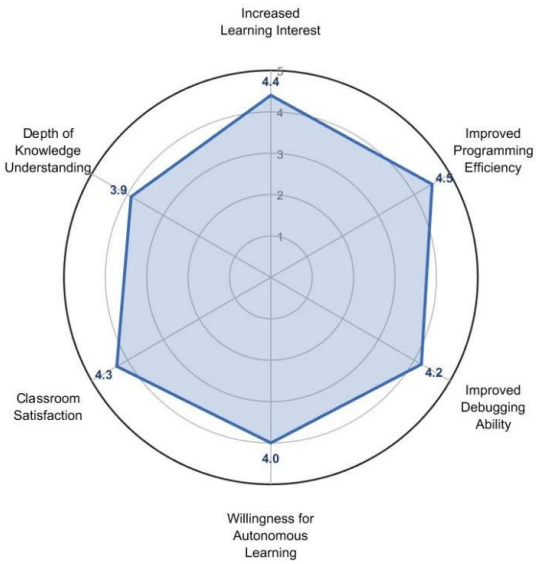

Figure 3 Radar Chart of the Student Satisfaction Survey

After the course, a questionnaire survey was administered to gauge students' subjective evaluation of the AI-assisted teaching model. A five-point Likert scale was used (1 = very dissatisfied, 5 = very satisfied), with all 30 students in the AI-assisted-teaching group completing the survey (response rate: 100%). Figure 3 shows the radar chart of the satisfaction-survey results, and Table 4 shows the corresponding questionnaire statistics. Students rated the dimensions of increased learning interest and improved programming efficiency relatively highly, while the depth-of-knowledge-understanding dimension was rated relatively lower. This pattern suggests that AI assistance has a more direct effect on improving efficiency and interest, while its effect on deepening understanding of principles depends more on whether instructional-session designs, such as the verification-record requirement described in Section 4.4, are properly implemented.

Table 4 Statistical Table of the Student Satisfaction Survey

Survey Dimension	Mean Score (5-point scale)	Standard Deviation	Sample Size
Increased learning interest	4.4	0.52	30
Improved programming efficiency	4.5	0.48	30
Improved debugging ability	4.2	0.61	30
Willingness for autonomous learning	4.0	0.65	30
Classroom satisfaction	4.3	0.55	30
Depth of knowledge understanding	3.9	0.70	30

6 DISCUSSION, CONCLUSION, AND OUTLOOK

6.1 Risks of AI-Generated Content and Over-Reliance

Generative AI is not always accurate in code generation and error diagnosis; it may produce incorrect register addresses or logic suggestions that do not match the specific hardware platform. In the case discussed in Section 4.4, had the student not cross-checked the interrupt-number correspondence against the datasheet, they could equally have been misled by an incorrect suggestion from AI. Such plausible-looking but incorrect suggestions are particularly risky in a hardware-oriented course, since a register value or timing parameter that appears syntactically reasonable may still be functionally wrong for the specific microcontroller model or crystal frequency in use, and this cannot be verified by inspection alone. Teaching must therefore explicitly inform students that AI's suggestions are for reference only, and that the official datasheet and actual on-board verification results should be treated as authoritative, thereby cultivating students' critical-verification awareness toward AI output. Specific measures include mandating a three-part "AI suggestion, datasheet/empirical verification, final conclusion" record section in the lab-report template, as described in Section 4.4, and setting up on-site, closed-book, AI-free debugging assessments at the midterm and final to check whether students' genuine troubleshooting ability matches their usual performance.

Closely related to this reliability concern is the risk of student over-reliance on AI. Without reasonable instructional organization, some students may tend to seek AI's help the moment a problem arises, or even submit a full requirement directly so that AI generates an entire block of code, preventing independent thinking and debugging ability from being adequately exercised. In such cases, students risk merely reproducing AI-generated solutions without going through the reasoning process needed to actually understand them — a pattern observed repeatedly during classroom circulation, where some students could run their code successfully but were unable to explain why a particular register configuration or timing value was chosen. For this reason, the model proposed in this paper emphasizes that students must first independently complete an initial version of the code and an initial round of troubleshooting; the timing and depth of AI-assistance intervention are constrained by instructional-session design, such as requiring students to independently write the initial version of the code as a prerequisite step before AI intervention in the workflow shown in Figure 2, and the structured-questioning training in Section 4.5 explicitly requires that AI only provide a hint on the approach rather than directly provide complete code. During circulating guidance, instructors should also watch for cases where a student's code is highly similar in style to typical AI output but the student is unable to explain the code's underlying logic, and intervene promptly when this occurs.

6.2 Adaptation of Assessment Methods and the Teacher's Role

The promotion of AI-assisted teaching places new demands on the management of academic integrity in assessment sessions such as lab reports and course design. It is recommended that assessments include a written-record requirement covering students' debugging approach and their process of adopting and verifying AI's suggestions, and that on-site question-and-answer sessions and random spot-checks be appropriately increased, for example by requiring students to explain, on the spot at the acceptance stage, the function of a section of code or to make an ad hoc modification to some functional parameter. This would allow a more comprehensive assessment of students' genuine ability level and avoid grading based solely on the final submitted code and report.

Alongside these assessment adjustments, the instructor's role itself gradually shifts under the AI-assisted teaching model, from lecturer and full-process question-answerer to instructional designer, AI-use guide, and gatekeeper for difficult problems. This places new demands on instructors' own ability to apply AI tools and to design instructional sessions, requiring continuous improvement of the teaching team's corresponding capabilities through means such as teaching-research activities and collective lesson preparation. Instructors also need to invest effort in advance to define

the reasonable boundaries of AI-assistance intervention in each laboratory project, as shown in the "main AI-assistance sessions" column in Table 2, and to dynamically adjust these boundaries based on actual teaching feedback, which itself constitutes a new test of instructors' teaching-research ability.

6.3 Conclusion and Outlook

Focusing on the practical pain points in programming and debugging instruction in the "Principles and Applications of Microcontrollers" course, this paper proposes a teaching model that uses generative AI as an assisting means, systematically constructed from the perspectives of instructional-design principles, overall framework, key functional modules, laboratory cases, and implementation workflow, and provides a preliminary analysis and discussion of the possible teaching effects and potential problems identified above. Teaching practice indicates that the reasonable introduction of AI assistance helps lower students' entry barrier to programming and improve debugging efficiency and learning autonomy, but instructional-session design and assessment-method adjustment must simultaneously guard against students' over-reliance on AI, so as to ensure that the goal of cultivating engineering-practice ability is achieved. Future research will systematically collect quantitative data, including debugging time, experiment scores, and satisfaction surveys, in real teaching classes, and combine statistical-testing methods for a more rigorous empirical analysis of the actual effectiveness of the AI-assisted teaching model. It will further explore pathways for integrating AI-assisted teaching with course-based ideological-and-political education and engineering-ethics education, as well as differentiated AI-assistance strategies for students of different foundation levels, so as to continuously refine the teaching model proposed in this paper.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

FUNDING

This work was supported by the Hubei Provincial Teaching Research Project for Higher Education Institutions (Grant No. 2020602), and the Ministry of Education Industry-Academia Cooperation Collaborative Education Project: Embedded Systems Faculty Development (Grant No. 240804422131149).

REFERENCES

[1] Frankford E, Sauerwein C, Bassner P, et al. AI-tutoring in software engineering education. Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training, 2024: 309-319.
[2] Aljedaani W, Eler M M, Parthasarathy P D. Enhancing accessibility in software engineering projects with large language models (LLMs). Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1, 2025: 25-31.
[3] Zhai C, Wibowo S, Li L D. The effects of over-reliance on AI dialogue systems on students' cognitive abilities: a systematic review. Smart Learning Environments, 2024, 11(1): 28.
[4] Alhur A A, Khlaif Z N, Hamamra B, et al. Paradox of AI in higher education: qualitative inquiry into AI dependency among educators in Palestine. JMIR Medical Education, 2025, 11: e74947.
[5] Kohen-Vacs D, Usher M, Jansen M. Integrating generative AI into programming education: student perceptions and the challenge of correcting AI errors. International Journal of Artificial Intelligence in Education, 2025, 35(5): 3166-3184.
[6] Annuš N. Investigation of generative AI adoption in IT-focused vocational secondary school programming education. Education Sciences, 2025, 15(9): 1152.
[7] Keuning H, Alpizar-Chacon I, Lykourentzou I, et al. Students' perceptions and use of generative AI tools for programming across different computing courses. Proceedings of the 24th Koli Calling International Conference on Computing Education Research, 2024: 1-12.
[8] Maleki A. Rethinking ethical academic integrity ecosystems in higher education in the age of artificial intelligence. Discover Artificial Intelligence, 2026, 6(1): 145.
[9] Alkam R S, Alsawalqa R O, Alreesi R. The moderating role of academic integrity in the relationship between artificial intelligence and critical thinking among graduate students in Jordan. Frontiers in Education, 2026, 11: 1776308.
[10] Alanazi M, Soh B, Samra H, et al. PyChatAI: enhancing python programming skills—an empirical study of a smart learning system. Computers, 2025, 14(5): 158.
[11] Zhu F, Zeng Z. A novel teaching method based on AI automated programming: a case study of Python programming. 5th International Conference on Internet, Education and Information Technology (IEIT 2025), 2025: 475-489.
[12] Sharma P. Java-powered AI agents implementing LLM-based intelligent systems for scalable and efficient applications. Journal of Computational Analysis & Applications, 2025, 34(3): 32.

[13] Li Z, Ye Q, Jin X. Reform and practice of teaching mode for "Principles and Applications of Microcontrollers" in the perspective of digital and intelligent transformation. Proceedings of the 2nd International Conference on Intelligent Education and Computer Technology, 2025: 769-774.
[14] Bolanakis D E. A survey of research in microcontroller education. IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, 2019, 14(2): 50-57.