Koh, E., et al. (Eds.) (2026). Proceedings of the 2nd International Conference on Learning Evidence and Analytics. Asia-Pacific Society for Computers in Education.

coding-exam: Enhancing Academic Integrity and Code Understanding through Personalized Assessments with Generative AI

Toshihiro KITA[a*], Rwitajit MAJUMDAR[a], Izumi HORIKOSHI[b] & Hiroaki OGATA[c]
[a]*Kumamoto University, Japan*
[b]*Uchidayoko Institute for Education Research, Japan*
[c]*Kyoto University, Japan*
*kita@rcis.kumamoto-u.ac.jp

Abstract: While Generative AI has made code generation effortless, it has also compromised academic integrity through "AI-assisted ghostwriting." This paper introduces "coding-exam," an LTI-compliant tool designed to analyze student-submitted source code and automatically generate personalized multiple-choice quizzes. By extracting and questioning critical logic within the students' own code, the system validates their genuine understanding. Evaluation results indicate significant improvements in student motivation and self-perceived understanding.

Keywords: AI cheating, LMS, Automatic question generation, Personalized learning

1. Introduction

Generative AI (GenAI) in programming education acts as a “double-edged sword" by assisting learners while simultaneously enabling "AI cheating." Regarding its role as a tutor, Shanto et al. (2025) found that GenAI effectively explains core principles to novices through dialogue, although its performance tends to decline when handling more complex logic. Addressing the need for structured evaluation, Kumar et al. (2024) utilized LLMs to automate quiz creation that aligns strictly with specific educational goals and pedagogical principles. Furthermore, systems like "Tutor Kai" offer unlimited personalized practice by generating custom tasks and solutions, which Jacobs et al. (2025) noted led to high student satisfaction and perceived learning benefits. To safeguard academic integrity against these same AI tools, this paper introduces "coding-exam," a system that requires students to pass AI-generated tests tailored specifically to their individual code submissions.

2. System Architecture

The system is built with Python, using FastAPI for the backend and Streamlit for the user interface. It integrates with LMSs like Moodle via the LTI 1.1 protocol. All files that make up this system are publicly available in the author's GitHub repository (Kita, 2026).

2.1 Core Components

- **Authentication**: Uses LTI to ensure secure access directly from the LMS. (by simplified implementation that performs Consumer Key matching without signature verification)
- **Content Generation**: Leverages Gemini API to analyze submitted Python files and generate multiple-choice questions.
- **Adaptive Assessment**: The system focuses on specific blocks of code to test the learner's comprehension of logic and syntax.

2.2 Context-Driven Quiz Generation for Logic Assessment

To ensure pedagogical validity, the system instructs the AI to identify lines critical to the program's logic, such as control structures and complex expressions. The prompt design emphasizes a specialized role and strict constraints: the AI must explain a line's functional purpose within the specific program context rather than merely describing its syntax (e.g., explaining a counter's role in a loop instead of just stating "increments a variable"). Furthermore, the AI generates "plausible distractors"—options that sound technically correct in a general sense but are logically false for that specific script (Kita, 2026). This approach ensures the quiz tests deep comprehension of the unique logic submitted by each student, effectively preventing guesswork based on general knowledge.

3. Mapping the Interface to Personalized Assessment

The user interface explicitly links student work to evaluation through three key views:

- **Learner Input** (Figure 1): The personalization is visible here, as the student must select the most appropriate description for specific lines (e.g., Line 1, Line 4, Line 5) of the code they provided. The dropdown options are generated dynamically to match the context of their specific variable names and logic.
- **Detailed Feedback** (Figure 2): In **Practice Mode**, the system provides an output results table that directly links the student's specific "Code" and "Line No." to the correct description. This allows students to see exactly where their understanding of their own code was incorrect. In contrast, in **standard mode**, the score is not displayed, and students are not shown whether their answers were correct or what the correct descriptions were.
- **Instructor Oversight** (Figure 3):Instructors assign Exam IDs to specific LTI links, track student high scores, and export data via CSV. They also control the "Practice Mode" toggle to facilitate immediate re-learning through personalized feedback.

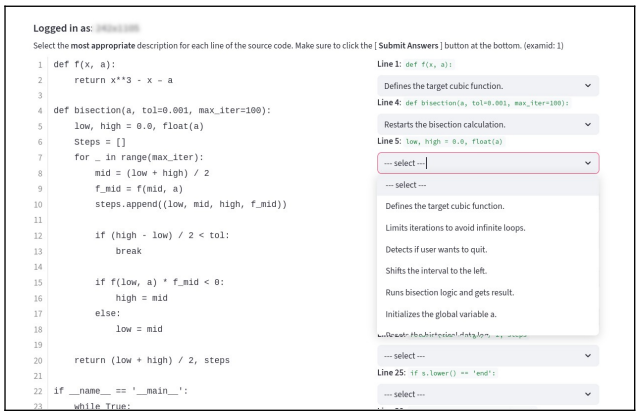

Figure 1. Learner Input

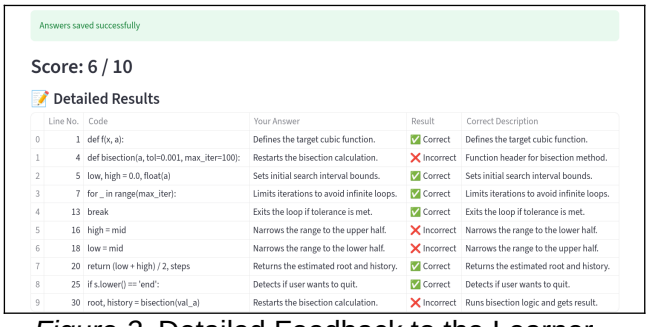

Figure 2. Detailed Feedback to the Learner

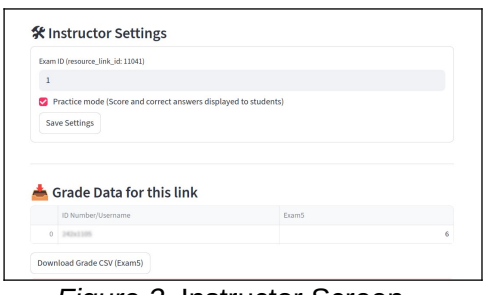

Figure 3. Instructor Screen

4. Methodology and Implementation

4.1 Deployment Context

The "coding-exam" system was integrated into a final proctored, on-site web examination to evaluate student comprehension of their submitted work. The examination included **Test 4** and **Test 5**, which required students to answer AI-generated, personalized quizzes based on their specific code submissions.

4.2 Assignment Specifications

The quizzes were derived from two distinct programming tasks:

- **Assignment 1 (Test 4)**: Implementation of the bisection method to solve the cubic equation $x^3 - x - a = 0$ until the error was 0.001 or less.
- **Assignment 2 (Test 5)**: Development of a Python script to retrieve and decode JSON-formatted data from a public Web API.

For both tasks, students were permitted to use Generative AI (GenAI) but were explicitly required to demonstrate a line-by-line understanding of the resulting code.

4.3 Assessment Design and Grading Policy

The instructional design distinguished between a formative learning phase and a summative evaluation phase:

- **Formative Phase (Test 4)**: To familiarize students with the personalized testing interface, a "Practice of Test 4" was provided. This session operated in **Practice Mode**, providing immediate feedback and correct answers to facilitate re-learning. Crucially, practice scores were excluded from the final grade to encourage exploration.
- **Summative Phase (Test 5)**: In contrast, a "Practice of Test 5" was intentionally omitted. This decision was made to ensure a rigorous summative assessment for grading purposes. By removing the practice scaffolding for Test 5, the instructor could accurately measure each student's independent mastery of Assignment 2 without the influence of prior exposure to the quiz distractors. This approach ensured that the final grades reflected genuine code understanding rather than the mere recollection of practice feedback.

5. Evaluation and Results

The tool's effectiveness was captured through a post-exam survey of 16 recruited volunteers out of 59 enrolled students. Using a 5-point scale (1 = "I don't think so," 5 = "I think so"), the results are presented in Table 1, Figure 4, and Figure 5.

Table 1. *Student Perception of Code Understanding and Test Effectiveness*

Survey Question	**Mean Score (1–5)**
Q1. Did the source code tests (Test 4 and 5) encourage you to re-examine the meaning of your code?	4.69
Q2. Before the Practice of Test 4, did you feel you understood the meaning of your code?	3.13
Q3. Did you understand your code better after doing the Practice of Test 4?	4.69

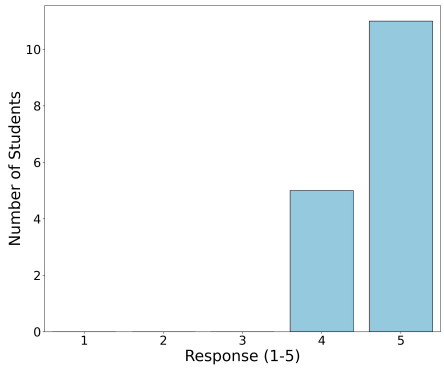

Figure 4. Motivation (Q1)

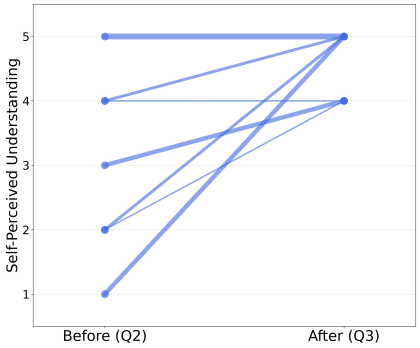

Figure 5. Understanding Change (Q2 to Q3)

5.1 Distribution of Motivation (Q1)

The histogram of responses for Q1 (Figure 4) shows a strong peak at 5.0 (Mean: 4.69). This suggests that the "unique test for a unique code" approach successfully triggered a "need to know". By linking the assessment directly to their own submissions for grading purposes, the system motivated students to shift from simply generating code with AI to truly understanding it.

5.2 Transition of Understanding (Q2 to Q3)

A slope graph (Figure 5) was used to visualize the change in each individual learner's self-perceived understanding.

- Q2 (Before the Practice of Test 4): Students reported a moderate understanding (Mean: 3.13), indicating they likely relied on GenAI to "write" code they couldn't fully explain.
- Q3 (After the Practice of Test 4): Self-perceived understanding rose significantly (Mean: 4.69).

The visualization shows that almost every student moved from a lower score in Q2 to a higher score in Q3, confirming that the personalized assessment acted as an effective learning intervention.

6. Conclusion

"coding-exam" effectively addresses the challenge of GenAI in education by forcing a transition from passive output to active comprehension. The system safeguards academic integrity while enhancing students' sense of achievement. Future work will expand question variety and support multiple programming languages.

References

Jacobs, S., Peters, H., Jaschke, S., & Kiesler, N. (2025). Unlimited practice opportunities: Automated generation of comprehensive, personalized programming tasks. *Proceedings of the 30th ItiCSE,* 319–325.

Kita, T. (2026). *Coding-exam: Programming Quiz System (LTI Tool)* [Source code]. GitHub. https://github.com/kita-toshihiro/coding-exam/blob/main/README.en.md

Kumar, Y., Manikandan, A., Li, J. J., & Morreale, P. (2024). Optimizing Large Language Models for auto-generation of programming quizzes. *2024 IEEE ISEC,* 1–5.

Shanto, S. S., Ahmed, Z., & Jony, A. I. (2025). Generative AI for programming education: Can ChatGPT facilitate the acquisition of fundamental programming skills for novices? *Proceedings of the 3rd ICCA,* 685–692.