Edumate.AI: A Generative AI Tutor to Improve Programming Skills Using Large Language Model (LLM)

Nur Syafia Amira Zairosezary[1],Sharifalillah Nordin[2], Rozianawaty Osman[3], Azliza Mohd Ali[4]

Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Malaysia

DOI: https://doi.org/10.47772/IJRISS.2026.1026EDU0177

Received: 17 March 2026; Accepted: 22 March 2026; Published: 09 April 2026

ABSTRACT

Learning programming language is difficult for many beginners, not only because of syntax, logic deployment, and debugging issues, but also due to the lack of immediate personalized support. General-purpose LLM tools can help but tend to promote answer dependency by generating complete answers without enforcing reasoning. This paper presents EduMate.AI, a logic-based generative AI tutoring system for beginner C++ learners. The system is implemented using a Model-View-Controller (MVC) architecture, integrating a large language model via constrained prompt orchestration. EduMate.AI aims to improve learners' conceptual understanding, reasoning ability, and learning experience by shifting generative AI from answer-delivery to structured tutoring support.

Keywords: Generative AI; Intelligent Tutoring System; Programming Education; C++ Learning; Large Language Model; Socratic Hinting; Reasoning Control

INTRODUCTION

Learning to program is an essential part of computer science education. Yet many students have difficulty understanding these basic concepts, including syntax, statement completion, and the problem-solving process that involves debugging. These problems are associated with low self-confidence and school achievement. Consequently, it also increased failure rates in introductory programming. Cognitive load is typically generated by programming learning students who are unfamiliar with how to think and are unable to generate code from logical expressions [1], [2]. Research indicates that managing this load is critical for beginner programmers to successfully move from syntax comprehension to problem-solving [2].

The recent progress in Artificial Intelligence (AI), such as Large Language Models (LLMs), has drastically changed the field of intelligent tutoring systems. LLMs are capable of generating natural language explanations, examples, and feedback that are suitable for beginners. However, moving beyond simple answer-generation towards LLM-based Socratic conversational agents has been shown to enhance students' reflective thinking and academic performance [3].

Despite such potential, the current programming assistance tools are not sufficiently personalized and interactive. Developed platforms lack traditional learning skills, making students rely on external forums or search engines that can give partial or false information. Guided, structured AI-based learning makes learners more productive in problem-solving than unguided exploration [4].

This paper presents EduMate.AI, a reasoning-based generative AI tutoring system designed to support beginner C++ learners through guided, staged reasoning. EduMate.AI introduces a system-level Reasoning Control Framework inspired by recent advancements in structured chain-of-thought (SCoT) prompting, which breaks down code generation into intermediate logical structures [5]. Also, a Prompt Rule Set and a Learning Safety Constraint Policy to prevent direct solution disclosure and promote active learning. Unregulated use of LLMs in introductory courses often leads to an 'assistance dilemma' where over-reliance on automated code generation results in poorer long-term learning outcomes [6].

This research offers significant contributions to the domain of AI-assisted programming education. First, it suggests EduMate.AI, a new generative AI tutoring system that changes the role of large language models from generating answers to providing structured, reasoning-based learning support. EduMate.AI is different from general purpose LLM tools that make people dependent on answers. It has a system-level EduMate Reasoning Control Framework (ERCF) that is based on structured chain-of-thought prompting [5]. This framework enforces staged reasoning processes like problem interpretation, concept identification, logical decomposition, error diagnosis, and guided hinting.

This study also suggests a Prompt Rule Set (PRS) and a Learning Safety Constraint Policy (LSCP) to keep AI systems in check and stop solutions from leaking. This makes sure that the learner is mentally engaged, which solves the problem of how to help someone that was found in earlier research [6].

Third, the study makes a contribution to the field with a pedagogically aligned architecture, integrating the functionality of LLM within a Model-View-Controller (MVC) architecture [20]. This facilitates the controlled and modular deployment of the potential of AI in tutoring tools. This addresses the need to bridge the gap between AI productivity tools and educational tools through the direct integration of the principles of instructional scaffolding [21].

Finally, the research introduces a comparative conceptual framework that distinguishes EduMate.AI from other approaches, including general-purpose LLMs, automated feedback approaches, and AI-native IDEs, by emphasizing reasoning, learning integrity, and adaptability over task accomplishment. Thus, the research is a contribution to the advancement of trustworthy and educational-oriented generative AI for novice programmers.

LITERATURE REVIEW

Several strategies have been proposed to help novice programmers, including traditional collaborative methods, automatic feedback systems, general-purpose large language models (LLMs), and artificial intelligence-enhanced development environments. While all these methods have greatly reduced the threshold for programming through automation and ease of use, they vary in their ability to support structured reasoning and conceptual understanding.

Conventional and Rule-Based Approaches

Pair programming and other traditional methods encourage people to work together to solve problems and use different ways to do so. However, traditional methods have some problems when it comes to scalability and the lack of quick and personal feedback, which can make it harder to fix mistakes and understand concepts [7].

To address the shortcomings of traditional methods and enhance motivational and conceptual comprehension, automated formative feedback systems such as AsPin and AP-Coach employ strategies including unit testing, Abstract Syntax Tree (AST) comparison, and rule-based feedback templates to elevate motivational and conceptual understanding and to pinpoint prevalent errors [8], [9]. There are, however, limits to rule-based approaches when it comes to new problems and complicated mistakes.

LLM-Based Programming Support

General-purpose LLMs, such as the ChatGPT, are being increasingly used as programming assistants due to their ability to generate code, explanations, and debugging support, as described in [10]. The latest versions of the models show promising performance for tasks such as fault localization and code generation, as described in [11, 12].

However, the tools are mainly used for task completion. They are likely to provide complete solutions, promote over-reliance, and lack any structured learning support. Furthermore, the tools are also known for some limitations, such as hallucinations, inconsistencies, and sensitivity to the design of the prompt, as described in [11, 13].

AI-Native Development Environments

In addition, AI-native development environments such as Cursor and similar tools integrate LLMs directly into the coding workflow, allowing for natural language interaction, context-aware suggestions, and even seamless debugging. While this increases productivity and decreases workflow friction, it also creates new pedagogical concerns related to reduced cognitive engagement, over-reliance on automation, and lack of support for structured reasoning, especially for new learners.

Research Gap and Motivation

However, a common drawback of these approaches is that the available tools either focus on productivity, i.e., LLM-based assistants and AI-native IDEs, or feedback, i.e., rule-based systems, but rarely both. Furthermore, these tools do not always provide explicit support for the construction of reasoning strategies necessary for problem solving, debugging, and algorithmic thinking. This drawback is further complicated by factors like overdependence on AI-generated solutions, lack of personalization, insufficient adaptation to the needs of the learner, and bias in automated feedback [15]–[18].

The need for a system that meets these requirements and provides a pedagogically appropriate approach to problem solving by ensuring structured reasoning, flexibility, and adaptation has led to the creation of EduMate.AI. EduMate.AI provides a reasoning-controlled approach to problem solving through structured, stage-based guidance.

METHODOLOGY

This section describes the methodology for research and the development system of EduMate.AI. Describing the architecture and development details in a structured and systematic way.

Software Development Methodology

This project was developed using the System Development Life Cycle (SDLC) approach [19]. This paper specifically focuses on the Requirement Analysis and System Design phases.

The first phase, requirement analysis, focuses on identifying the learning challenges faced by beginner C++ students and analysing the limitations of existing generative AI tools. The system design phase then translates the identified requirements into a detailed system architecture, including user interface design and interaction flow. Next, the implementation phase involves developing the EduMate.AI by integrating reasoning-based generative AI features, guided hints, and step-by-step explanations. Finally, the testing phase evaluates the usability and effectiveness of the system. In addition, this study incorporates a comparative evaluation against general-purpose LLM tools to validate the pedagogical effectiveness of the proposed approach.”

Phase 1: Requirement Analysis

Target Population and Sampling

The target population for requirement validation and evaluation is a group of first-year undergraduate students from the Faculty of Computer and Mathematical Sciences at Universiti Teknologi MARA (UITM). These participants are purposively selected as they fall into the category of beginner learners disproportionately presented in the literature review as most susceptible to cognitive overload.

Phase 2: System Design

System Architecture (MVC)

EduMate.AI uses MVC architecture to maintain strict separation between the system’s tutoring logic, the data persistence layer, and the student interface. This architectural choice is particularly effective as it allows for scalable development and clear organization of pedagogical components [20].

The View represents the web-based interface used by students. It includes pages such as login/registration, learning dashboard, code or questions input panel, step-by-step explanation output, guided hints display, and progress history. The Controller acts as the system's control layer that handles user requests and directs them to the appropriate services. It manages user sessions, validates inputs, controls the learning flow, and sends requests to the generative AI module. Controllers are organised by function, such as authentication, learning navigation, tutoring interaction, and progress tracking. The model manages all system data and business logic. It includes domain services and communicates with both the database layer and the generative AI module. The Model also contains a Prompt Engine to format the user inputs into structured prompts to produce reasoning-based responses, such as step-by-step explanations and guided hints.

Figure I shows the illustration of the MVC architecture of EduMate.AI, including database integration and the AI module.

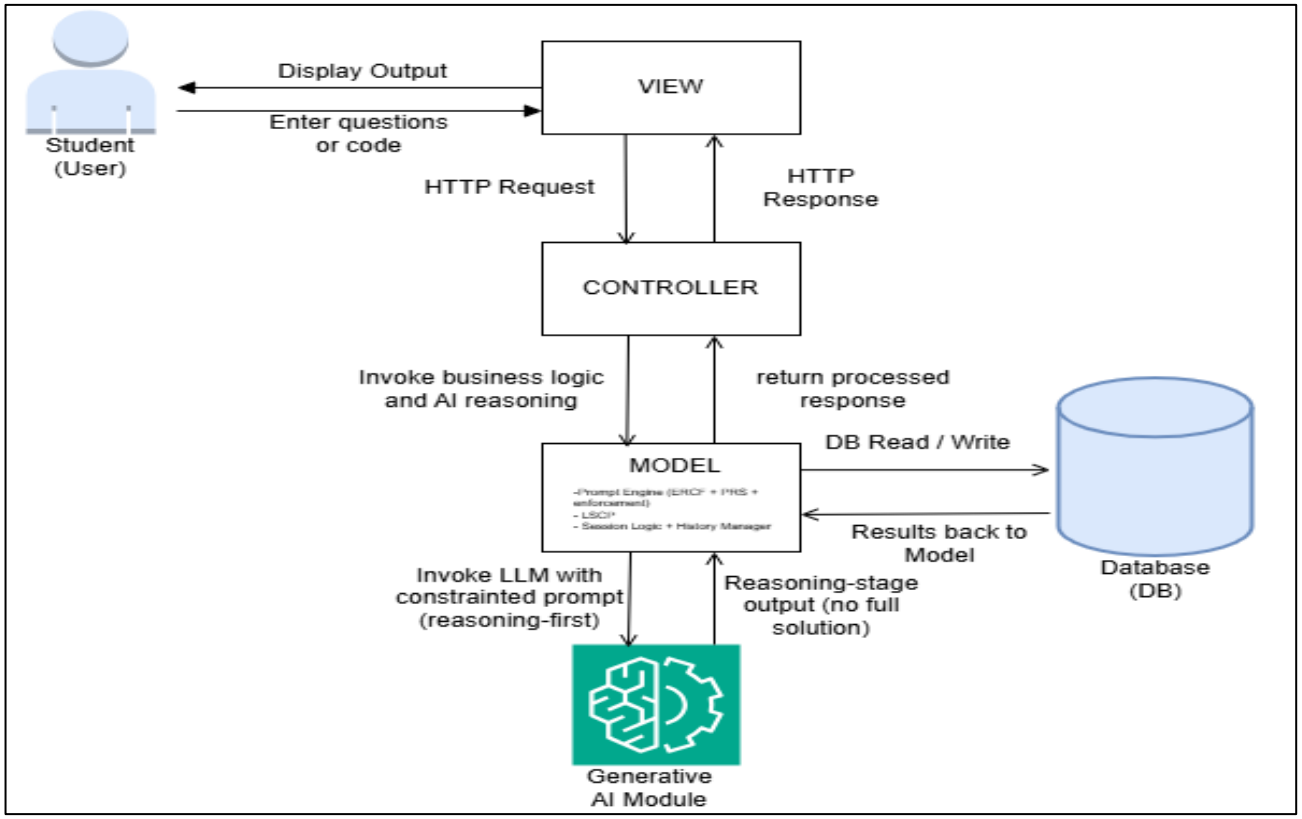

Figure I MVC Architecture

Reasoning Control and Safety Mechanism

EduMate Reasoning Control Framework (ERCF)

EduMate.AI proposes EduMate Reasoning Control Framework (ERCF) to produce tutoring-style guidance rather than direct solutions. ERCF is a system-level reasoning pipeline that governs how the AI module generates output for beginner programmers. The ERCF is grounded in instructional Scaffolding theory [21], which posits that learners require temporary support that is gradually removed to foster independence. Table I shows the ERCF Framework.

Table I EduMate Reasoning Control Framework (ERCF)

Stage	Purpose	Output Constraint
R1: Problem Interpretation	Identify task, inputs, outputs, and expected behavior.	No code outputs.
R2: Concept identification	Identify relevant C++ concepts	Definition and concept

		explanation only.
R3: Logical Decomposition	Break the solution into structured reasoning steps	Pseudocode/ Logic steps.
R4: Error Diagnosis	Identify the misconception or error type and location	Full fix /final code
R5: Guided Hinting	Provide incremental hints to move the learner forward	Partial hints only

Unlike general-purpose chatbots, which rely on user prompting and may still output complete solutions, ERCF is enforced at the system level through the Model layer and Prompt Engine. This makes EduMate.AI a controlled tutoring system where the reasoning sequence, allowed output type, and solution disclosure are governed by design rather than user request.

Each stage of the EduMate Reasoning Control Framework (ERCF) is operationalized into measurable components. For example, R1 (Problem Interpretation) is assessed based on the learner's ability to correctly identify inputs and outputs, while R3 (Logical Decomposition) is evaluated based on the completeness and correctness of step-by-step solution planning. Each component is assessed using a rubric-based scoring scheme, enabling quantitative evaluation of learners' reasoning ability.

Prompt Rule Set (PRS)

EduMate.AI implements a Prompt Rule Set (PRS) within the Prompt Engine to operationalize ERCF and ensure consistent tutoring behavior. PRS acts as a hard constraint layer that regulates the AI output format and prevents the model from producing final answers, full programs, or copy-ready solutions.

The PRS includes the following enforced rules:

The AI must not generate complete C++ programs or full assignment-ready solutions.

The AI must follow the ERCF stage order and respond in a stage format.

The AI must stop after delivering the current stage output and encourage learner reflection before proceeding.

If the user request final solution, the AI must refuse and redirect to the reasoning steps and guided hints.

The AI must provide incremental hints and avoid

revealing full code blocks that solve the entire task.

Learning Safety Constraint Policy (LSCP)

In order to ensure EduMate.AI remains a learning support tool rather than an answer generator, the system enforces a Learning Safety Constraint Policy (LSCP). This policy restricts the disclosure of complete solutions and copy-ready code. Instead requires that assistance be delivered through reasoning-first explanations and incremental hints. When users request final answers or complete programs, the system responds by redirecting them to guided reasoning steps, ensuring learners remain cognitively engaged with the problem-solving process.

System Flowchart

The process flow of the system explains the steps that are done by EduMAte.AI during user interaction. It starts from a student logging into the system and submitting a programming question. The input is processed by the controller and sent to the generative AI module.

The AI module generates a reasoning-based response with a step-by-step explanation and guided hints. The

output is then presented to the user through the user interface. The structured flow makes sure that students are getting clear and guided exploration support rather than straight answers. The overall process flow of the system is shown in Figure II.

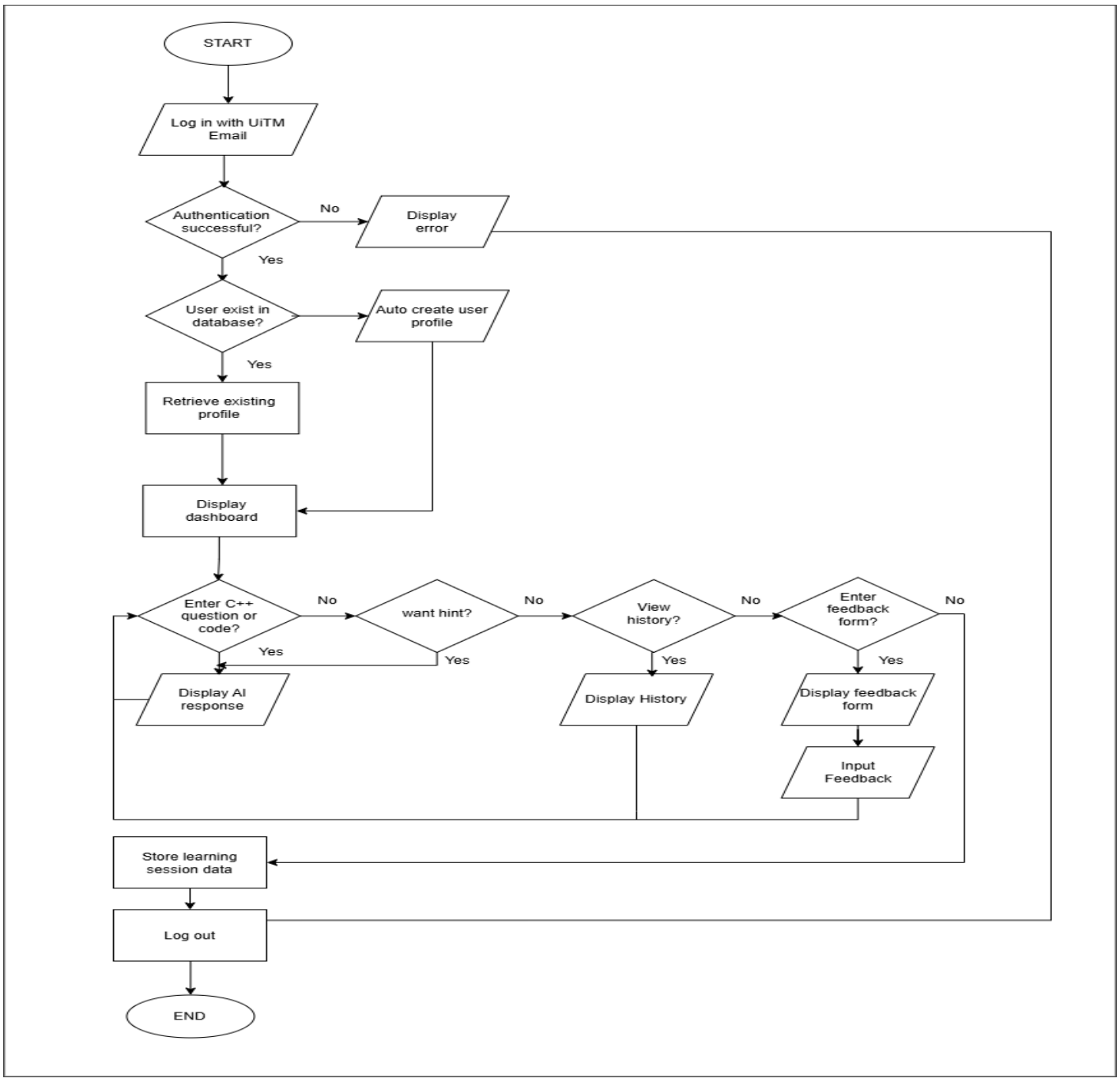

Figure II System Flowchart

Phase 3: Implementation

This section describes how EduMate.AI will be implemented. The software requirements for developing and running EduMate.AI are Visual Studio Code for the primary integrated development environment (IDE). Python 3.10 for core logic and backend processing. Flask for web framework to handle routing and API requests. Google Generative AI SDK as an interface for the LLM generate reasoning. MySQL 8.0 is a relational database for persistent storage of users and history. Bootstrap 5/CSS3 for creating a responsive and beginner-friendly UI. Lastly, JavaScript (ES6+) to handle asynchronous (AJAX) calls to the AI backend.

The system was developed using a laptop with intel i5, AMD Ryzen 5 processor, 8 GB RAM, and 256 GB SSD. Deployment environment: localhost and a minimum of 5 Mbps internet connectivity.

Phase 4: Testing

In this phase, testing strategy and test cases are conducted to evaluate EduMate.AI after full implementation. Testing is performed to verify that each module functions correctly and meets the functional and non-functional requirements.

Usability Test

Usability evaluation will include the use of a set of questions that have to be answered by novice C++ learners who use EduMate.AI. The goal of this assessment is to quantify the users' satisfaction, clarity of interface, and perceived usefulness of the system's Socratic-style guided hints.

The questionnaire adopts a 5-point Likert scale, ranging from Strongly disagree (1) to Strongly agree (5). Data collected from the questionnaire will be analyzed using descriptive statistics, including frequency, mean scores, and standard deviation. In addition, open-ended responses will be reviewed qualitatively to capture users' subjective feedback and suggestions for improvement. Table II presents the usability evaluation criteria used in this study.

Table II: Questionnaire evaluation criteria

Evaluation Criteria	Description
Ease of navigation	Measures how easily students can move between system features such as the dashboard, history, and feedback pages.
Interface Clarity	Evaluates the visual layout, readability, and organization of the user interface components.
Clarity of Explanation	Evaluates how clear and understandable the step-by-step explanations generated by EduMate.AI.
Helpfulness of Guided Hints	Measure the effectiveness of guided hints in supporting reasoning without directly revealing full solutions.
Overall User Satisfaction	Measures overall satisfaction with EduMate.AI as a learning support tool for C++ programming.

Effectiveness Evaluation

For the evaluation of the effectiveness of EduMate.AI, this study proposes to use a quasi-experimental design with comparative baseline conditions. It evaluates the effectiveness of EduMate.AI through a controlled comparison with general-purpose LLM tools such as ChatGPT. Specifically, participants in this study will be divided into two groups: one that uses a general-purpose LLM for learning and another that uses EduMate.AI to assist in learning. Participants in this study will be presented with similar programming tasks to ensure that there is consistency in learning objectives and task difficulty. The evaluation involves one group of participants using a general-purpose LLM for learning programming, and another group of participants will use EduMate.AI that is based on ERCF to assist in learning programming.

Evaluation Metrics

To ensure a comprehensive assessment of system effectiveness, multiple quantitative and qualitative metrics are defined:

Learning Gain - measured as the difference between post-test and pre-test scores:

Learning Gain = Post-test Score − Pre-test Score

This reflects improvement in conceptual understanding and problem-solving ability.

Task Completion Accuracy - defined as the percentage of correctly solved programming tasks to measures correctness of solutions.

Accuracy = (Correct Solutions / Total Tasks) × 100

Task Completion Time - measured as the total time (in minutes) required to complete each task, representing learning efficiency.

Hint Dependency Ratio - defined as the ratio of hints requested to total tasks:

Hint Dependency = Number of Hints Requested / Total Tasks

Lower dependency indicates stronger independent reasoning ability.

Reasoning Quality Score - the EduMate Reasoning Control Framework (ERCF) is operationalized into measurable components by evaluating learner performance across:

- Problem interpretation (R1)
- Concept identification (R2)
- Logical decomposition (R3)
- Error diagnosis (R4)

Each component is assessed using a rubric-based scoring scheme to produce a composite reasoning score.

User Experience and Perceived Learning - measured using the Likert-scale usability questionnaire, including clarity of explanation, helpfulness of guided hints, and overall satisfaction.

Data Analysis Plan

Pre-test, post-test, and usability questionnaire data will be analyzed using descriptive statistics (frequency distribution, mean scores, and standard deviations). Learning efficiency will be measured by pretest-posttest score differences with a paired t-test. A non-parametric alternative (Wilcoxon signed-rank test) will be conducted if the number of observations is limited or assumptions for normality are not satisfied.

In this stage, the focus of this research is on the design and development of EduMate.AI. The empirical evaluation, including experiments and comparison to general-purpose LLM tools such as ChatGPT and statistical validation, will be performed in the future after the completion and deployment of the proposed system.

Comparative Logic Trace

Table III: Logic Trace (Standard AI vs. Edumate.AI)

Scenarios	Standard LLM Response	EduMate.AI (PRS Rule 1-5)
Input: "Fix my cin >> x error."	"You didn't declare x. Add int x;	"Before taking input x, does the compiler know what data type x is?"
Input: "Write a for loop for me."	Provide complete lines of code	"I can't write the code for you, but let's do it together. What are the three parts of a loop?"

Feature-Set Comparison

Table IV: Comparative Analysis of Support Tools

Features	General AI	EduMate.AI (Proposed)
Primary Strategy	Information Delivery	Instructional Scaffolding
Constraint Level	User-prompt dependent	System-level Enforcement (PRS)
Output Goal	Completion	Reasoning
Integrity Risk	High	Low

User Interface

This proposed user interface (Figure III-VI) of EduMate.AI is meant to have a low barrier to entry and be friendly towards new programming. The interface is designed for simplicity and findability to minimize cognitive load during learning.

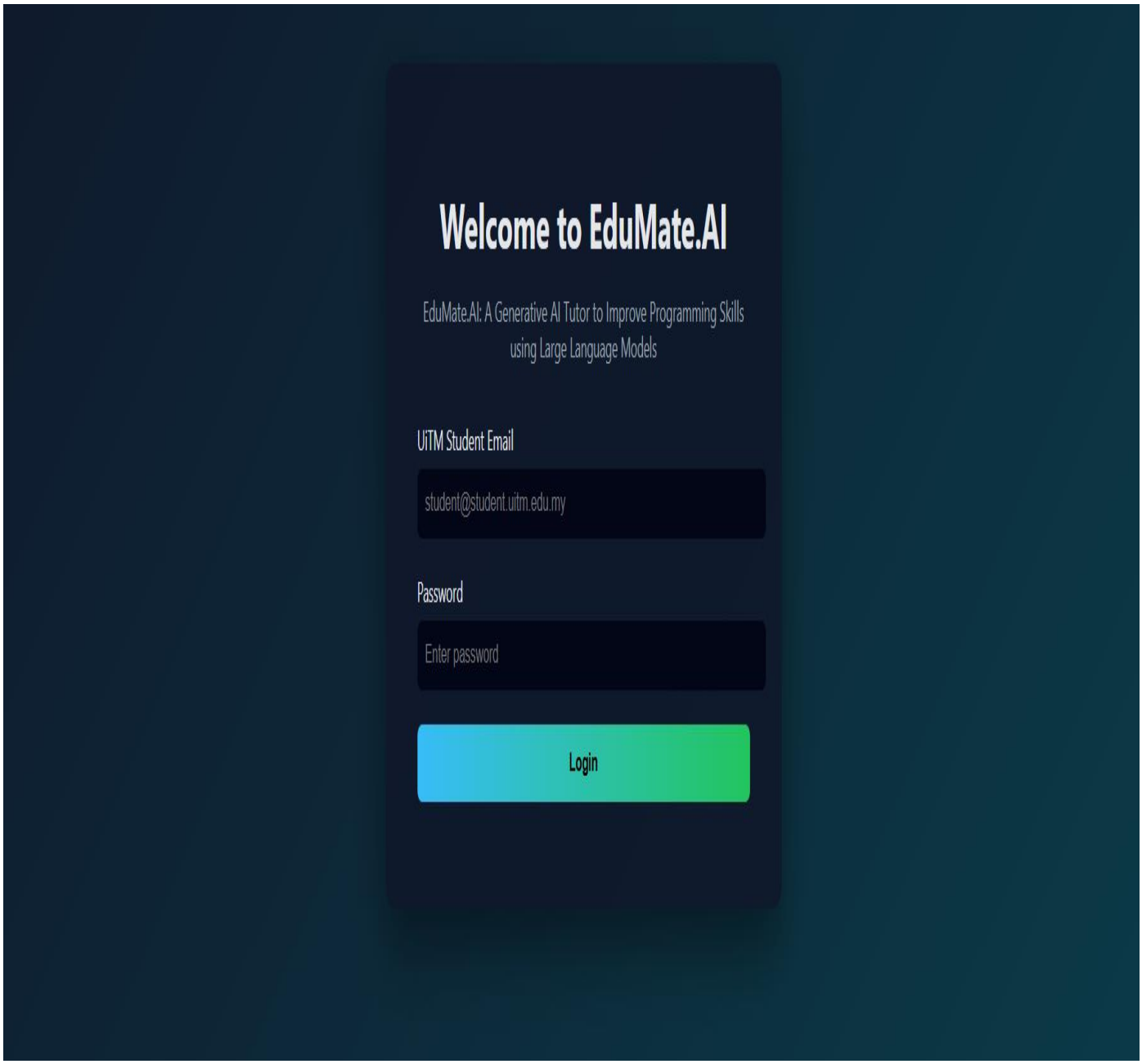

Figure III Student Log-in Page

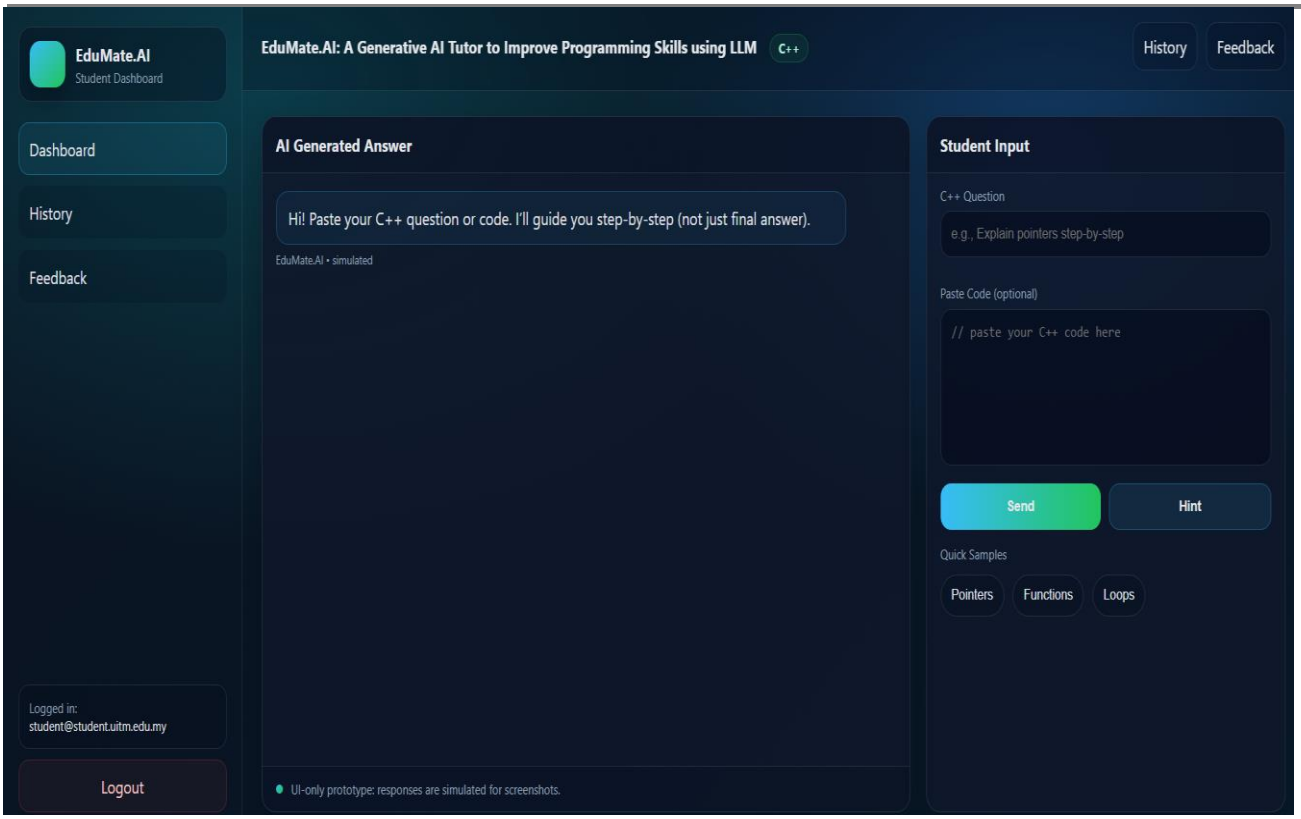

Figure IV Student Dashboard Page

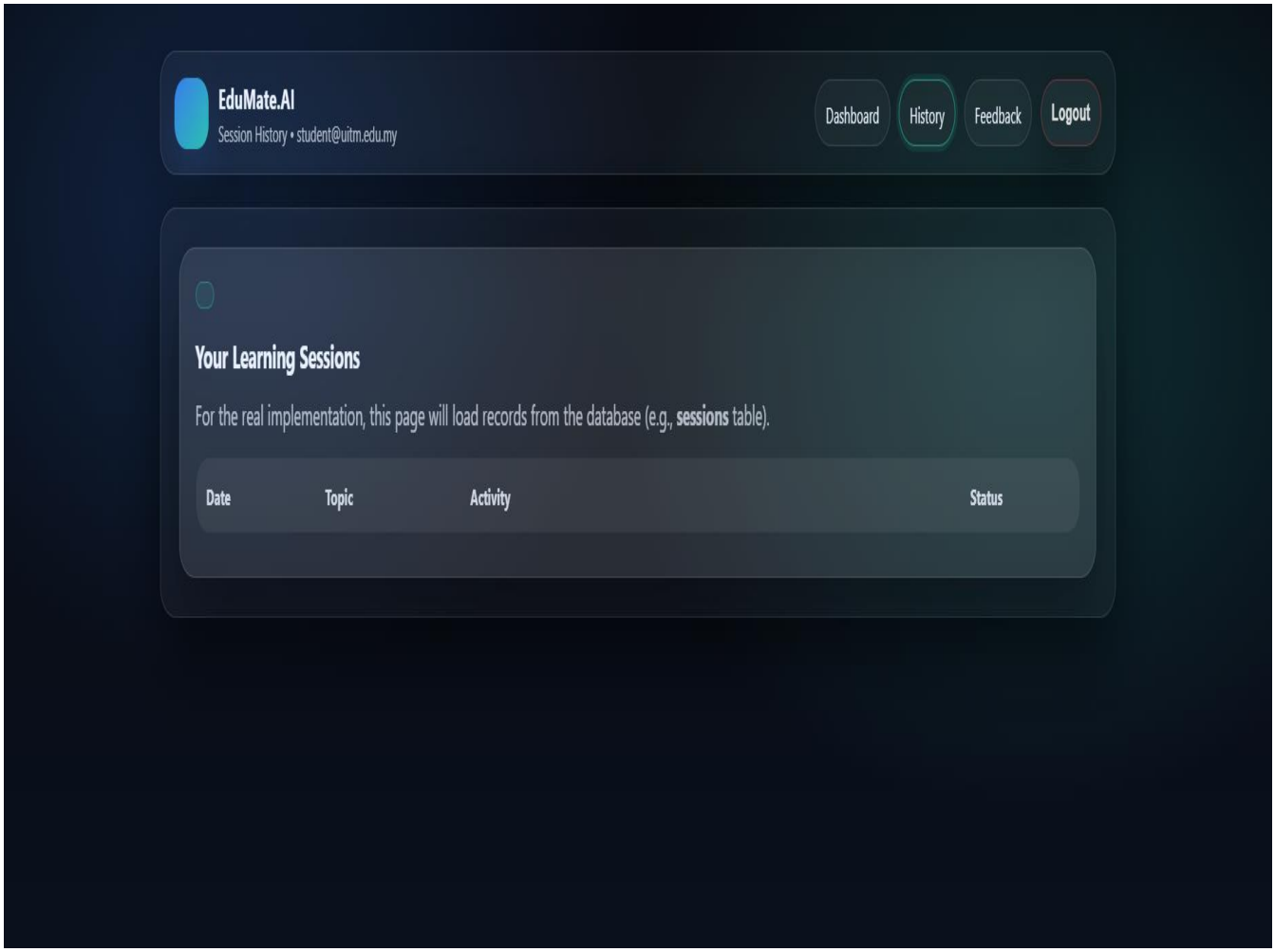

Figure V Learning History Page

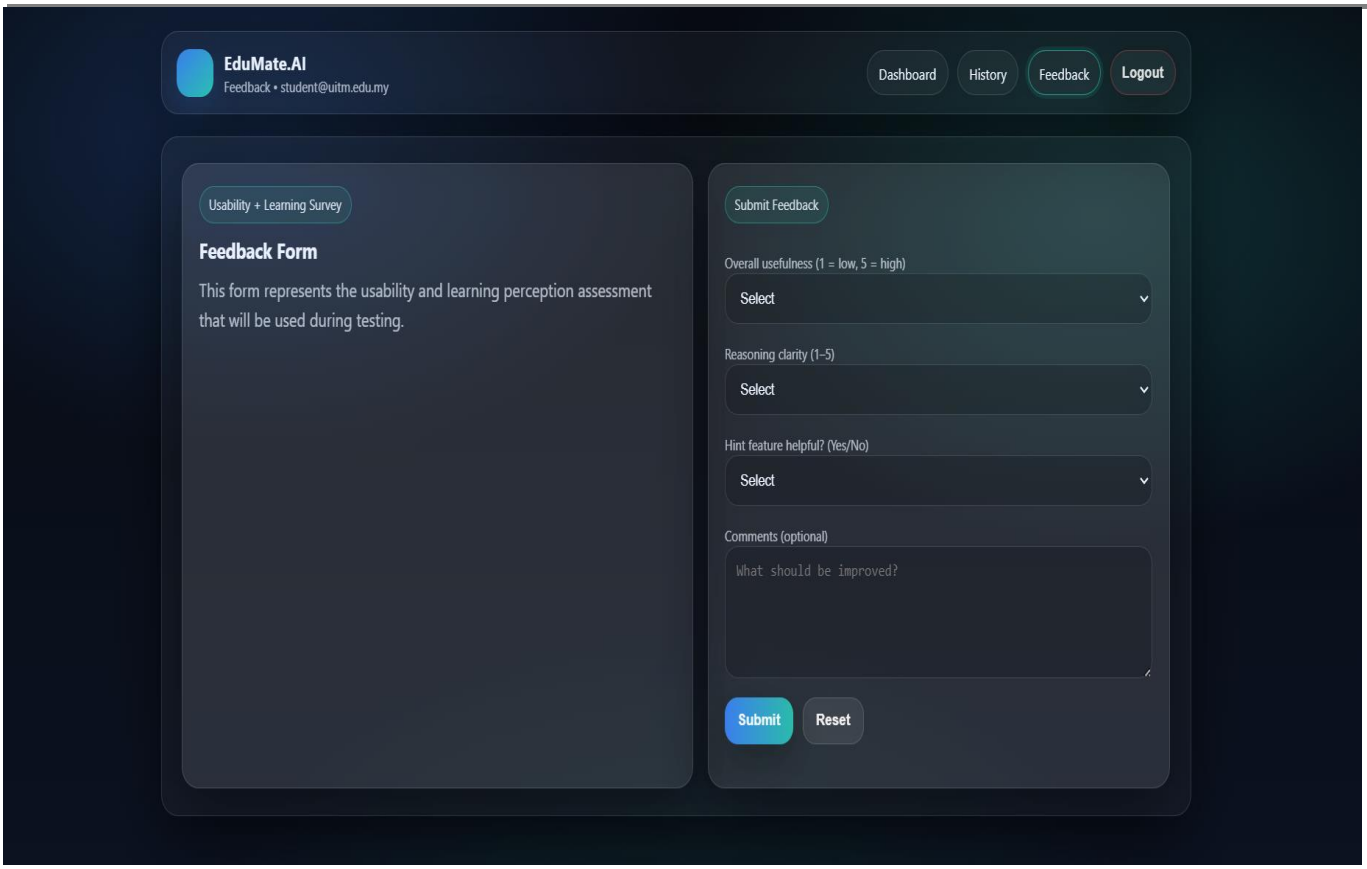

Figure VI Feedback Page

Unlike the multi-purpose interface of general chatbots, EduMate.AI's dashboard prioritizes 'Hint' visibility over 'Solution' visibility. Align with LSCP, ensuring students' primary interaction is with problem-solving rather than solution generation.

FUTURE WORK RECOMMENDATION AND CONCLUSION

In this paper, the researchers propose the development of an AI tutoring tool called EduMate.AI, which is based on the concept of generative AI for supporting novice programmers. The proposed AI tutoring tool has the ability to transform the concept of generative AI from an answer generation tool to an instructional support tool with the integration of the EduMate Reasoning Control Framework (ERCF) based on the MVC (Model-View-Controller) architecture.

The framework addresses the major limitations of existing AI tools, which include the promotion of answer dependencies and a lack of cognitive engagement. EduMate.AI, therefore, offers a model for the effective and safe use of generative AI in programming education, while at the same time promoting the development of independent logical thinking skills without compromising academic integrity. This study contributes to the advancement of trustworthy and educationally aligned AI systems by focusing on the significance of aligning AI capabilities with existing instructional principles. The study also explores the potential of generative AI as an educational learning aid.

The next step in the immediate future will be to implement the system technically using the Google Generative AI SDK and Python Flask. The next step will be to conduct experiments to empirically validate the system to check the learning efficiency and self-efficacy of the students. The future work will include extending the system to support more programming languages like Java and Python. In addition, the system will be integrated with LMS and mobile applications. The future work will also include conducting experiments to empirically validate the system to check the learning efficiency and self-efficacy of the students.

REFERENCES

1. Wang, X., Liu, J., & Zhang, Z. (2020). Sustainable C++ education in general high school: From teaching programming skills to developing computational thinking. Proceedings of the 15th International Conference on Computer Science & Education (ICCSE), 35–40.
2. Duran, R., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive load theory in computing education research: A review. ACM Transactions on Computing Education, 22(4), Article 40, 1–27. https://doi.org/10.1145/3483843
3. Xi, L., Zhang, Y., & Wang, Q. (2026). Investigating the effects of an LLM-based Socratic conversational agent on students' academic performance and reflective thinking in higher education. Computers & Education, 241, Article 105494. https://doi.org/10.1016/j.compedu.2025.105494
4. Tanvir, S. H., & Kim, G. J. (2024). WIP: Generative and custom chatbots in computer programming education and their effectiveness: A systematic literature review. Proceedings of the IEEE Frontiers in Education Conference (FIE). https://doi.org/10.1109/FIE61694.2024.10893425
5. Li, J., Li, G., Li, Y., & Jin, Z. (2025). Structured chain-of-thought prompting for code generation. ACM Transactions on Software Engineering and Methodology, 34(2), Article 37, 1–23. https://doi.org/10.1145/3690635
6. Poitras, E., Crane, B., & Siegel, A. (2024). Generative AI in introductory programming instruction: Examining the assistance dilemma with LLM-based code generators. Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual), 186–192. https://doi.org/10.1145/3649165.3690111
7. He, H., & Li, X. (2025). A comparative study on the effectiveness of generative AI tools and pair programming in college students' programming learning. Proceedings of the International Conference on Distance Education and Learning (ICDEL), 224–228. https://doi.org/10.1109/ICDEL65868.2025.11193454
8. Holanda, M., et al. (2023). Automatic formative and motivational feedback personalized for introductory programming course. Proceedings of the IEEE Frontiers in Education Conference (FIE). https://doi.org/10.1109/FIE58773.2023.10343336
9. Duong, T. N. B., Shar, L. K., & Shankararaman, V. (2022). AP-Coach: Formative feedback generation for learning introductory programming concepts. Proceedings of the IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), 323–330. https://doi.org/10.1109/TALE54877.2022.00060
10. Scholl, A., & Kiesler, N. (2024). How novice programmers use and experience ChatGPT when solving programming exercises in an introductory course. Proceedings of the IEEE Frontiers in Education Conference (FIE). https://doi.org/10.1109/FIE61694.2024.10893442
11. Torek, A., Sorensen, E., Hahle, N., & Kennington, C. (2024). A systematic evaluation of code-generating chatbots for use in undergraduate computer science education. Proceedings of the IEEE Frontiers in Education Conference (FIE). https://doi.org/10.1109/FIE61694.2024.10893165
12. Liu, Y., et al. (2024). Empirical evaluation of large language models for novice program fault localization. Proceedings of the IEEE International Conference on Software Quality, Reliability, and Security (QRS), 180–191. https://doi.org/10.1109/QRS62785.2024.00027
13. Santos, E. A., Salmon, A., & Hammer, K. (2025). It's dangerous to prompt alone! Exploring how fine-tuning GPT-4o affects novices' programming error resolution. IEEE Access, 13, 166943–166958. https://doi.org/10.1109/ACCESS.2025.3613500
14. Kumar, Y., Akinwunmi, I., & Kruger, D. (2025). Evaluating the advantage of an AI-native IDE cursor on programmer performance. Proceedings of the IEEE Integrated STEM Education Conference (ISEC). https://doi.org/10.1109/ISEC64801.2025.11147402
15. Kaufmann, C., Pavão, J., & Wahl, H. (2022). Is there a need for automated code review to be used in teaching? From the perspective of students. Proceedings of the 17th Iberian Conference on Information Systems and Technologies (CISTI).
16. Salvador, L. R., Llerena, C. A., & Dai, N. H. P. (2025). ChatGPT in education: Challenges and best practices. Proceedings of the IEEE International Conference on Intelligent Engineering Systems (INES), 291–296. https://doi.org/10.1109/INES67149.2025.11078207
17. Rachh, R., & Kavatagi, S. (2024). Study on students' perceptions of generative AI in learning

programming courses. Proceedings of the International Conference on Advanced Technologies (ICONAT). https://doi.org/10.1109/ICONAT61936.2024.10774681

18. Kostopoulos, G., Gkamas, V., Rigou, M., & Kotsiantis, S. (2025). Agentic AI in education: State of the art and future directions. IEEE Access, 13, 177467–177491. https://doi.org/10.1109/ACCESS.2025.3620473
19. Olorunshola, O. E., & Ogwueleka, F. N. (2022). Review of System Development Life Cycle (SDLC) models for effective application delivery. In A. Joshi et al. (Eds.), Information and Communication Technology for Competitive Strategies (ICTCS 2020) (pp. 281–289). Springer. https://doi.org/10.1007/978-981-16-0739-4_28
20. Magfira, F., Matulatan, T., Fahmitra, N. F., Irawan, F., & Herikson, R. (2024). Implementation of Model View Controller architecture in designing Outcome-Based Education (OBE) curriculum management information system. BIO Web of Conferences, 134, 05002. https://doi.org/10.1051/bioconf/202413405002
21. Abd El Bar, D. A., Mohaseb, M. M., & El Bassuony, J. M. (2022). Using instructional scaffolding in hybrid learning environment: A critical review. Port Said Journal of Educational Research, 1(1), 132–154. https://doi.org/10.21608/psjer.2022.160500.1004