Jiang, B. et al. (Eds.) (2025). Proceedings of the 33rd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

CodeRunner Agent: Integrating AI Feedback and Self-Regulated Learning to Support Programming Education

Huiyong LI[a] & Boxuan MA[b*]
[a]*Research Institute for Information Technology, Kyushu University, Japan*
[b]*Faculty of Arts and Science, Kyushu University, Japan*
*boxuan@artsci.kyushu-u.ac.jp

Abstract: The emergence of Large Language Model (LLM) tools has revolutionized programming education, demonstrating considerable efficacy in providing instant, personalized feedback. However, most existing LLM-based tools focused on direct coding assistance, and neglected the cultivation of self-regulation skills. Moreover, many of these tools operate independently from institutional learning management systems, which creates a significant pedagogical disconnect that limits the ability to leverage contextual learning materials and exercises for generating tailored, context-aware feedback. To address these challenges, we developed CodeRunner Agent, an LLM-based programming assistant. CodeRunner Agent enhances students' self-regulated learning by providing strategy-based AI feedback and embedding within Moodle system. Additionally, it empowers educators to customize AI-generated feedback by incorporating detailed context from lecture materials, programming questions, student answers, and execution results. This integrated approach, emphasizing self-regulation skill development with contextual awareness, offers promising avenues for data-driven enhancements in programming education with AI.

Keywords: LLM-powered tools, programming education, self-regulated learning, personalized feedback, CodeRunner

1. Introduction

Innovation within higher education has been accelerated dramatically by digital transformation strategies and evolving pedagogical approaches. Programming education, in particular, has attracted an ever-growing number of students with wider recognition of the critical importance of computer science literacy. This growth has created significant pedagogical challenges in providing timely, individualized assistance to each student (Yilmaz & Yilmaz, 2023). Large Language Models (LLMs) have emerged as a promising technological intervention. A variety of LLM-powered programming assistants have been developed to provide timely coding guidance and suggestions, potentially transforming traditional instructional approaches and feedback mechanisms in programming education (Pankiewicz & Baker, 2023).

While these advancements present exciting opportunities for personalized learning, they also raise concerns about over-reliance and reduced development of self-regulated learning (SRL) and problem-solving skills (Humble et al., 2024; Prasad & Sane, 2024). Easy access to answers may encourage surface-level understanding, yet many LLM tools lack pedagogical guardrails to promote active engagement and SRL strategies (Sun et al., 2024). A further limitation is the lack of integration with institutional Learning Management Systems (LMS) like Moodle. Effective feedback is highly context-dependent, relying on alignment with lecture materials and assignment specifications (Kaur & Chahal, 2024; Kazemitabaar et al., 2024). Without LMS integration, tools cannot leverage rich contextual data or provide educators with insights into student interactions and learning outcomes (Ma et al., 2024a, 2024b). Therefore, it's essential to ensure that the AI-generated feedback is both relevant and aligned with course objectives by integrating LLM-based tools into the LMS environment.

To address these limitations, we developed CodeRunner Agent (Li & Ma, 2025), an LLM-powered programming assistant embedded within CodeRunner, an open-source Moodle

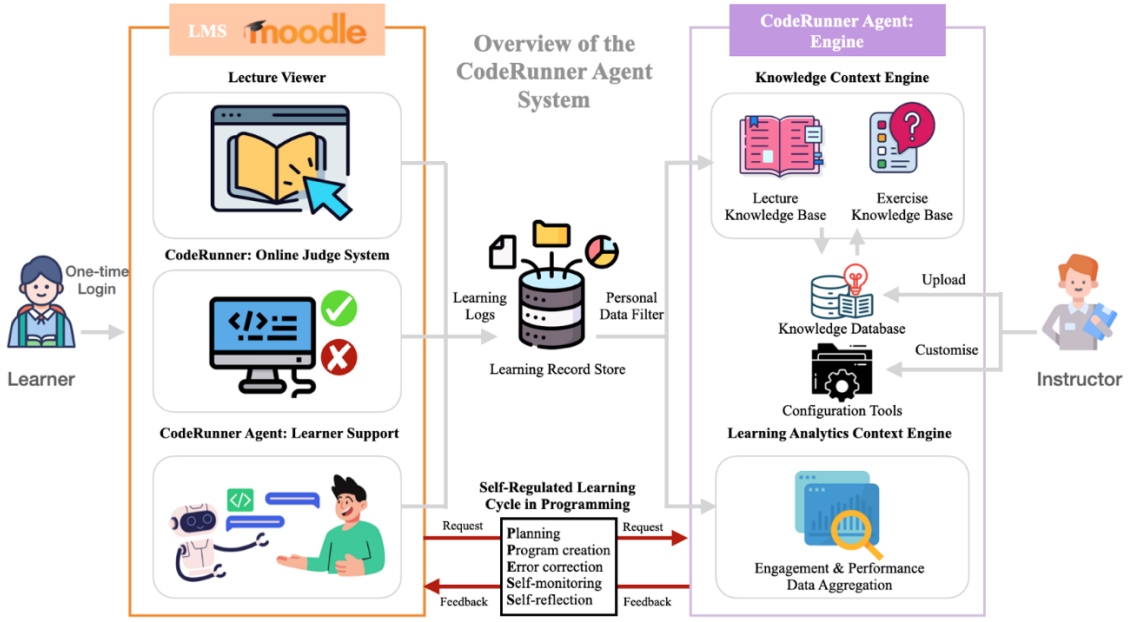

Figure 1. Overview of the CodeRunner Agent Programming Support Environment.

plugin for executing and assessing student-submitted code. CodeRunner Agent leverages the comprehensive context available within an LMS environment from learning logs and enhances students' self-regulated learning in programming education. Specifically, it enhances the delivery of individualized feedback by combining students' knowledge level, self-regulation skills, and strategy-focused LLM-based scaffoldings. It also tracks AI feedback and subsequent actions in students' coding behaviors. This provides valuable insights into learning behaviors and the effectiveness of AI-assisted feedback. By addressing the challenges of scalability, contextual feedback, and skill enhancement, our approach contributes to a deeper understanding of AI's role in modern education.

2. CodeRunner Agent

2.1 Overview of CodeRunner Agent

The framework of CodeRunner Agent is shown in Figure 1. It is developed and integrated to scaffold programming education with an LLM-based assistant in the Moodle LMS. The framework contains a Lecture Viewer, a CodeRunner plugin, and a CodeRunner Agent.

Lecture Viewer is provided to deliver the lecture slides by instructors and access the lecture slides by learners. The operations of the lecture viewer are recorded in the form of Experience API (or xAPI) statements. Then, the xAPI statements are stored in the Learning Record Store (LRS). *CodeRunner* is an open-source plugin that can be imported to Moodle (Lobb & Harlow, 2016). Learners can write programming codes and receive automated grades by running it in a series of tests. A logging plugin named "Logstore xAPI" is used to record the results and send them to the LRS. For example, the code for the test, got output, correct status, and mark reward will be logged in LRS if learners run the test code once. *CodeRunner Agent* is an AI-powered tool to support learners' SRL and teachers' customized designs for LLM-based feedback in programming education. It can be embedded into the Moodle LMS and executed with the CodeRunner plugin. Learners can receive general-purpose and programming-specific regulatory strategy feedback based on OpenAI API capabilities and strategy-focused prompt design. Specifically, the agent provides scaffolding hints while not directly generating code solutions by code guardrails design. A specific LLM prompt is used to rewrite the response by giving salient features of the code when code blocks are detected in the scaffolding hints. The interactions with the agent are automatically tracked as xAPI statements and stored in the LRS (e.g., the request type, request time, exercise ID).

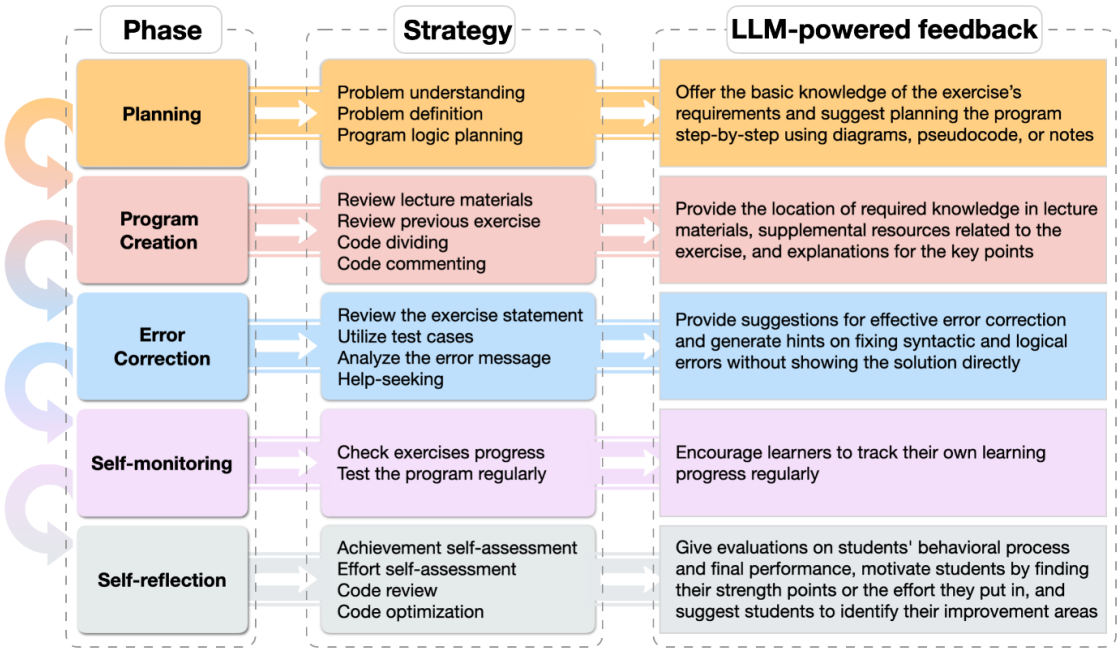

Figure 2. Phase, Target SRL strategy, and LLMs Feedback in the CodeRunner Agent.

The CodeRunner Agent's core intelligence contains two context engines: Learning Analytics Context Engine (LACE) and Knowledge Context Engine (KCE). LACE calculates and aggregates learners' engagement metrics (i.e., time spent, attempt frequency) and performance metrics (i.e., success rates, error patterns) from LRS data after filtering personal data (i.e., name, gender, and email). LACE is integrated as contextual embedding of self-regulation behavioral strategies with the feedback of the agent. KCE handles both lecture and exercise knowledge bases. The lecture knowledge base manages the key programming concepts and the dependencies between the concepts from lecture materials. The exercise knowledge base categorizes exercises by concept, difficulty, solution, and typical mistakes. KCE is used as learner knowledge embedding in the agent.

Instructors can upload lecture materials (i.e., concept definition, concept illustration, and annotated examples) and exercises (i.e., problem statement, solution, and test cases) to the context engine. They will be converted to textual knowledge and stored in the knowledge database for future retrieval by the Retrieval Augmented Generation (RAG) technique. More importantly, instructors can update the knowledge bases and customize the parameters of the context engine by using the configuration tools.

2.2 SRL Support Model in CodeRunner Agent

To enhance strategy-based programming learning for novice learners, LLM-powered feedback is integrated into a five-phase model grounded in the Zimmerman's SRL theory (Zimmerman, 2008) and adopts a conceptual framework for regulating learning in programming (Silva et al., 2024). The model adds two CS domain-specific elements to the general three-phase SRL cycle (forethought, performance, and self-reflection). The model contains five phases, named PPESS: *Planning*, *Program creation, Error correction, Self-monitoring,* and *Self-reflection*.

The phase, target SRL strategy, and LLMs support in the CodeRunner Agent are shown in Figure 2. Both general-purpose and programming-specific regulatory strategies are included for SRL scaffoldings in the five-phase PPESS model. *Planning* supports problem analysis and decomposition. *Program creation* assists the implementation of coding solutions using both declarative and procedural knowledge. *Error correction* helps cognitive diagnosis and metacognitive regulation for addressing coding errors. *Self-monitoring* supports ongoing evaluation and adjustment toward task completion during coding and debugging. Finally, *Self-reflection* helps learners assess their performance and improve future learning strategies.

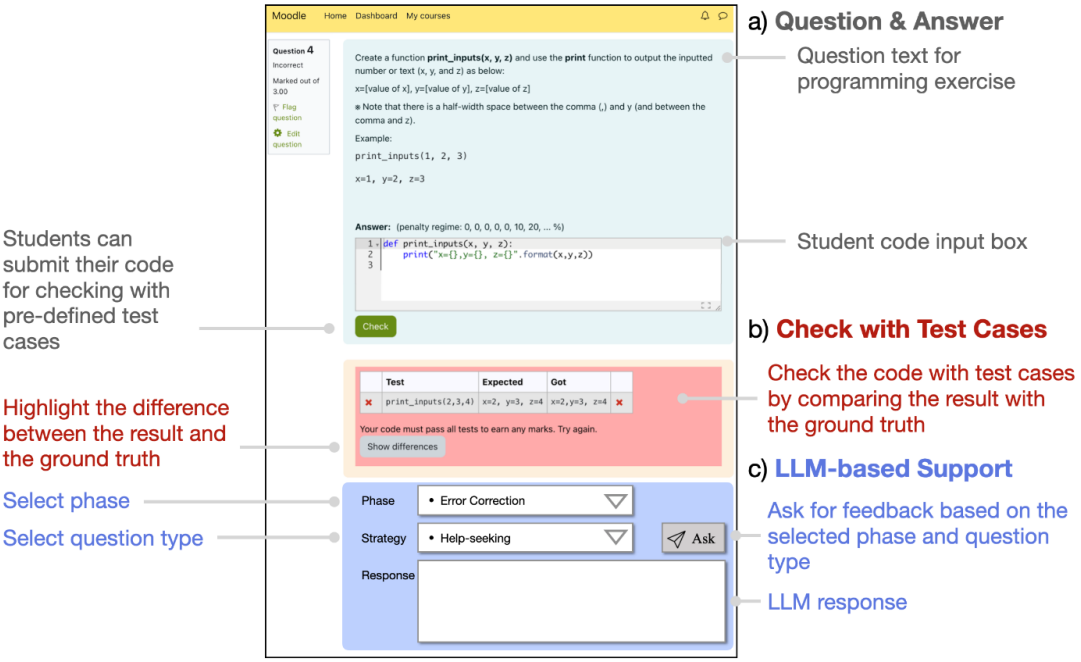

Figure 3. User Interface of Integrated CodeRunner Agent for Learners.

Figure 3 illustrates the user interface of the integrated CodeRunner Agent, which consists of three main components: (a) Question & Answer, (b) Check with Test Cases, and (c) LLM-based Support. The LLM-based support box for the CodeRunner Agent appears below the test results provided by CodeRunner. Learners have the option to select one of five SRL phases like error correction. Additionally, they can select either the question type in general level as default or the question type from one of the specific strategies in the phase like help-seeking. Upon submitting a request, the LLM-based strategy feedback is displayed within the response box.

3. User Experience Interview with Students and Educators

To gain deeper insights into students' and educators' perspectives on the CodeRunner Agent, we conducted semi-structured interviews with four students and two educators after introducing the agent and discussing its capabilities and potential use cases. The student participants included an undergraduate with one month of programming experience (S1) and three graduate students with programming experiences of two years (S2), three years (S3) and six years (S4), respectively. Both educators involved were actively teaching undergraduate programming courses, with teaching experiences of four years (T1) and two years (T2), respectively. Each interview, conducted via Zoom with informed consent, lasted approximately one hour.

3.1 Students Perceptions

- **General Impressions** Students generally held favorable impressions of the agent. S2 and S4 expected the agent to support computational thinking, problem requirement understanding, and code quality improvement more than just code validators. S3 was positive about the multi-phase support provided by the agent, stating that it "quickly helps me understand problems and errors."
- **Over-Reliance Risk Control** As a novice programming learner, S1 appreciated the design of the hint-like feedback and expressed that "hiding the direct answers will help my own thinking". S1 also wanted detailed explanations and visual feedback for error messages. S2 liked the way it "helps develop my thinking skills rather than override them." As advanced programming learners, S4 emphasized the "assistant role" of the agent and preferred the agent "helps me understand the target problem more concisely and why my

understanding was wrong." However, S2 raised concerns that some novice students may struggle even with hints and suggested providing pseudo-code or partial code to ease the learning curve.

- **Contextual Assistance** S4 appreciated the design of the contextual assistant feature since it can provide "more relevant knowledge and suggestions" for the specific learning tasks. Also, the seamless integration of the agent within LMS was highlighted as a key advantage. S3 valued the ability to support the entire learning workflow, from reading materials to coding and checking codes and receiving LLM-assisted feedback.
- **Self-regulated Learning Support** S4 highlighted the critical role of planning support in the agent and wanted to use it regularly. S4 also mentioned that "self-reflection support is highly related to the learning tasks" and expected to use it for code optimization. S2 highlighted that "students can use the agent in different SRL stages, sometimes in a cycle or non-linear manner." S3 expressed that "self-reflection on programming tasks can be widely practiced."
- **Trust and Reliability** Although many students frequently use AI tools, they do not fully trust them. S1 used them regularly for both learning and everyday tasks but viewed them more as virtual freshman peers rather than trustworthy sources, citing issues with complex questions. S4 expressed a more critical perspective, stating that ChatGPT had "about 60%-70% accuracy during my advanced learning tasks" and exhibited weaknesses in logical reasoning. S3 warned that "LLM feedback may not always be accurate or applicable. It may even be misleading in some cases." Across responses, they raised concerns about the low accuracy in advanced tasks, lack of transparency, and the potential for misleading outputs.

3.2 Educators Opinions

Educators provided valuable feedback on the CodeRunner Agent, emphasizing the importance of customization, instructor control over response types, and its pedagogical benefits. T2 noted that CodeRunner Agent is "more beneficial than ChatGPT," explaining that "many students struggle with crafting effective prompts when using ChatGPT" and that "open-ended interactions may raise privacy concerns." In contrast, "the customized feedback from CodeRunner Agent, integrated with SRL, offers contextually relevant support, facilitating actual knowledge acquisition rather than merely solving tasks." T1 further emphasized that "helping students understand the question and concept is more effective for student learning compared to directly offering answers." Additionally, T2 highlighted CodeRunner Agent's potential in a real classroom setting, envisioning its role in facilitating student self-reflection on lecture material and supporting assignment completion more effectively.

Educators expressed concerns about students becoming overly dependent on repeated LLM feedback requests. T2 suggested "setting limits on how frequently students can access such feedback." He further recommended "structuring feedback progressively, beginning with simpler guidance and providing more detailed assistance only when students continue to struggle." Lastly, both educators highlighted the need for a monitoring dashboard to analyze student interactions and LLM responses, aiding instructors in identifying instructional gaps and enhancing teaching strategies.

4. Discussion and Future work

This study aims to develop an LLM-based programming assistant, CodeRunner Agent, that seamlessly integrates with a lecture viewer and CodeRunner plugin in the Moodle LMS. The design is grounded within Zimmerman's SRL theory and targets the freshmen students' programming-specific knowledge acquisition as well as general self-regulation skill development. This study can fill the gap between AI, SRL, and LA by combining LLM-based scaffolding, self-regulation in programming, and learning log-based contextual feedback.

Future work will involve updating and enhancing the CodeRunner Agent by focusing on four key design considerations for LLM-based tools (Kazemitabaar et al., 2024): *D1. Exploiting unique advantages of CodeRunner Agent* (e.g., for deciding when to use the tool,

what are the role and unique advantages of CodeRunner Agent compared to other available resources within the learning ecosystem?); *D2. Refining the CodeRunner Agent interface* (e.g., what are the considerations for CodeRunner Agent to allow users to formulate requests in a way that balances user-friendliness and understandingness with meta-cognitive engagement and SRL skills?); *D3. Balancing the directness of CodeRunner Agent responses* (e.g., how direct should CodeRunner Agent's responses be so that it balances directness and learning engagement, and who should control this balance?); *D4. Supporting trust, transparency, and control* (e.g., once a response from CodeRunner Agent is received, what considerations are needed to ensure accuracy, trust, transparency, and control?).

Moreover, we will enhance educator-agent collaboration by enhancing usability and streamlining the integration of instructor-facing tools like the configuration tool and monitoring dashboard. In addition, we will evaluate the effectiveness of the CodeRunner Agent through short-term pilot studies in actual classroom settings and semester-long experiments conducted across multiple university classes.

Acknowledgements

This work is supported by JSPS KAKENHI Grant Numbers JP24K20903 and JP25K17078.

References

Humble, N., Boustedt, J., Holmgren, H., Milutinovic, G., Seipel, S., & Östberg, A. S. (2024). Cheaters or AI-enhanced learners: Consequences of ChatGPT for programming education. *Electronic journal of e-Learning*, *22*(2), 16-29.

Kaur, A., & Chahal, K. K. (2024). A learning analytics dashboard for data-driven recommendations on influences of non-cognitive factors in introductory programming. *Education and Information Technologies*, *29*(8), 9221-9256.

Kazemitabaar, M., Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., & Grossman, T. (2024). Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems, 650,* 1-20.

Li, H., & Ma, B. (2025). Design of AI-Powered Tool for Self-Regulation Support in Programming Education. *arXiv preprint arXiv:2504.03068*.

Lobb, R., & Harlow, J. (2016). Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, *7*(1), 47-51.

Ma, B., Chen, L., & Konomi, S. I. (2024a). Enhancing programming education with ChatGPT: a case study on student perceptions and interactions in a Python course. In *International Conference on Artificial Intelligence in Education* (pp. 113-126). Cham: Springer Nature Switzerland.

Ma, B., Chen, L., & Konomi, S. I. (2024b). Exploring Student Perception and Interaction Using ChatGPT in Programming Education. In *21st International Conference on Cognition and Exploratory Learning in the Digital Age, CELDA 2024* (pp. 35-42). IADIS Press.

Pankiewicz, M., & Baker, R. S. (2023). Large Language Models (GPT) for automating feedback on programming assignments. In *Proceedings of the 31st International Conference on Computers in Education* (pp. 68-77).

Prasad, P., & Sane, A. (2024). A self-regulated learning framework using generative AI and its application in CS educational intervention design. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1070-1076).

Silva, L., Mendes, A., Gomes, A., & Fortes, G. (2024). What Learning Strategies are Used by Programming Students? A Qualitative Study Grounded on the Self-regulation of Learning Theory. *ACM Transactions on Computing Education, 24*(1), 1-26.

Sun, D., Boudouaia, A., Zhu, C., & Li, Y. (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*(1), 14.

Yilmaz, R., & Yilmaz, F. G. K. (2023). The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, *4*, 100147.

Zimmerman, B. J. (2008). Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects. *American educational research journal, 45*(1), 166-183.