A Study on AI-Driven Question Generation Using ChatGPT for Python Programming Education

Khaing Hsu Wai[1,3], Ye Kyaw Thu[2,3*], Thazin Myint Oo[3], Nobuo Funabiki[4], Lu Xiqin[5], and Akihiro Yamamura[1]
[1]Graduate School of Engineering Science, Akita University, Japan
[2]National Electronics and Computer Technology Center, Pathum Thani, Thailand
[3]Language Understanding Lab., Myanmar
[4]Department of Information and Communication Systems, Okayama University, Japan
[5]College of Information Science and Engineering, Ritsumeikan University, Japan

***Abstract*— Programming education is becoming more essential due to the increasing demands on coding skills and computer literacy. Consequently, many professional schools and universities are providing programming courses to train future programming engineers. However, making quizzes and practice questions for diverse programming topics can be time-consuming for the teachers, reducing the time they can spend on teaching, such as guiding students and providing personalized feedback. In this paper, we propose an AI-driven question generation using ChatGPT to support Python programming education. We developed an automated program that can generate three types of questions to cover basic programming concepts: *Code Understanding Questions*, *Output Prediction Questions*, and *Code Completion Questions* by using ChatGPT's language model (LM). For evaluations, we have prepared 28 python source codes and applied 140 generated questions to our *Language Understanding* lab members in online setting. The examination was conducted as an unsupervised, open-book assessment, allowing participants to reference materials while answering the questions. This setup aimed to simulate a realistic self-learning environment, reflecting how students typically engage with AI-generated questions outside of formal classroom settings. The application results confirm the validity of the proposal and our initial findings indicate that AI-generated questions are effective in supporting both educators and students for Python programming education.**

***Index Terms*—Programming education, Python, ChatGPT in Education, Question generation, Teaching Assistance Tools**

I. Introduction

PROGRAMMING education is crucial in fostering critical thinking, problem-solving abilities, and creativity of students. These skills empower them to explore a broad range of career paths after graduations. Consequently, *computer programming* has become a pivotal subject in universities and professional schools. As a result, many universities and professional schools are offering *programming courses* to train future programming engineers.

Python is an ideal starting language for students, who are new to programming due to its straightforward syntax and readability, making it easy for beginners to understand code and self-study. Its simplicity allows students to quickly grasp core programming concepts without being overwhelmed by complex syntax. Learning Python not only builds essential programming skills but also prepares students for real-world tech jobs, as Python is highly valued in the industry [1]. Python programming education plays a critical role in preparing students for advanced fields like *data science*, *machine learning*, and *artificial intelligence*. Python's powerful libraries and frameworks—such as Pandas, NumPy, and TensorFlow—allow students and researchers to conduct in-depth analyses, develop predictive models, and perform complex data processing. By using *Python* in both introductory and advanced courses, universities prepare for the students to have strong foundation in programming, readying them for data-focused careers in today's tech-driven world.

Teachers/educators use and prepare various materials to effectively engage students and enhance learning outcomes. Quizzes and questions play a vital role in student learning by reinforcing key concepts, testing comprehension, and encouraging active participation with the materials. Regular quizzes help students retain knowledge and identify areas where they may need additional support.

However, creating diverse quizzes and practice questions across programming topics can be time-consuming for teachers, limiting the time they can spend on interactive teaching activities, such as guiding students and providing personalized feedback.

In this paper, we implement an AI-driven program that uses *ChatGPT's language model* to automate *question generation* for *Python programming education*. The program reads pre-prepared Python source codes at three introductory courses: *Introduction to Python, Intermediate Python, and Intermediate Python for Data Science*. By analyzing each code snippet, the program can generate *three* types of questions:

1) Code Understanding Question: This question type assesses students' comprehension of Python code by focusing on the logic, structure, and purpose of code segments. Students are asked to interpret code snippets, explain what a given piece of code does, or identify errors, helping to reinforce fundamental programming concepts.
2) Output Prediction Question: This question type requires students to predict the output of a given Python code snippet. By analyzing code execution step-by-step, students

Manuscript received March 30, 2025; Revise: July 28, 2025; Accepted July 30, 2025; *Corresponding author: Ye Kyaw Thu (email: yekyaw.thu@nectec.or.th).

practice understanding program flow and debugging, developing skills to anticipate how code will behave when run.

3) Code Completion Question: In this question type, students are given incomplete code and asked to fill in the missing parts, such as keywords, function names, or parameters. This type of question strengthens students' understanding of syntax, function use, and logic, supporting them in writing correct and working code.

These questions are formatted as *multiple-choice (MCQ)* and *true/false* for *code understanding* and *output prediction* question types and *short-answer fill-in-the-blank* for *code completion* question type. This approach reduces the time and effort required for quiz preparation, allowing educators to focus more on interactive teaching and feedback, while providing students with effective, targeted practice in Python programming skills.

The rest of this paper is organized as follows: Section II discusses related works in literature. Section III presents our study on AI-driven question generating using chatGPT's API. Section IV evaluates the proposal. Finally, Section V concludes this paper with future works.

II. Related Works

In this section, we discuss related works in literature.

In [2], the authors presented a rule-based question generation system that uses linguistic templates and syntactic structures to generate factual questions from text. It highlighted the limitations of rule-based methods, especially in terms of flexibility and adaptability, compared to more recent machine learning approaches.

In [3], the authors explored a template-based approach to automatically generate multiple-choice questions, using predefined patterns to create questions that test specific knowledge areas. This paper pointed out limitations in adaptability and the difficulty of producing diverse question types.

In [4], the authors examined how Codex, as an AI-powered coding assistant, can support students with instant code generation, debugging, and comprehension, making it a valuable tool for beginners. The authors also highlighted both the advantages of Codex—such as providing immediate feedback and freeing educators to focus on higher-level concepts—and the risks, like the possibility of students becoming overly reliant on the tool without fully understanding fundamental coding principles.

In [5], the authors introduced GPT-3, a large language model capable of generating context-specific questions without extensive training data. The model demonstrated the ability to adapt to different cognitive tasks, including question generation, by leveraging few-shot learning techniques. This work represented a major advancement in AI-driven systems and their ability to generate questions that cater to diverse educational needs.

In [6], the authors explored the potential applications and implications of artificial general intelligence (AGI) within educational settings. This paper discussed how AGI could transform education by providing highly adaptive, personalized learning experiences and automating complex instructional tasks that go beyond current AI capabilities. The authors also addressed challenges, including ethical concerns, technical limitations, and the need for policies that ensure AGI's safe and effective integration into educational environments

In [7], the authors discussed potential issues such as privacy, data security, and the risk of over-reliance on AI-driven interactions, which may affect students' learning independence. Kooli also explores solutions to mitigate these risks, including implementing robust data protection measures, ensuring transparency in chatbot usage, and establishing guidelines to balance AI support with human oversight. The study highlights the need for thoughtful integration of chatbots to maximize educational benefits while addressing ethical challenges.

In [8], the authors introduced an AI-based system designed to support beginners in learning programming. The system functioned as a virtual coach, providing real-time feedback, hints, and tailored guidance as students work through coding tasks. This paper discussed how this AI system helps reduce common challenges novice programmers face, such as debugging and understanding complex syntax, by offering instant, adaptive assistance.

In [9], the authors explored the use of large language models (LLMs) to create educational content for computer science students. The authors examined how LLMs can generate diverse learning materials, such as explanations, practice questions, and feedback, to support student learning. Their findings indicated that while LLMs can produce valuable educational resources, challenges remain in ensuring accuracy and relevance. The paper highlighted both the potential and limitations of LLMs in automating content creation for CS education.

III. AI-Driven Question Generation Using ChatGPT

In this section, we present our study of an AI-driven program that uses ChatGPT to automate question generation for Python programming education.

A. Conceptual Workflow of Question Generation

The conceptual flow for how the AI-driven question generation program operates is as follows.

1) Source Code Preparation: The system begins by reading a collection of pre-prepared Python source codes by the teachers, organized into three main categories: Introduction to Python, Intermediate Python, and Intermediate Python for Data Science. These categories ensure that questions are tailored to the appropriate skill level and cover relevant basic Python programming concepts.
2) Source Code Analysis: The source code is loaded into the program within *Google Colab*, where each code snippet is processed to extract key features and concepts, such as specific functions, loops, data types, and common programming patterns. This step allows the system to determine which elements will be the focus of each question.
3) Question Generation with ChatGPT: The question generation process also continues in *Google Colab*, where the

ChatGPT API is called to generate questions based on the analyzed source code. Using the ChatGPT language model, the system automatically generates questions based on the analyzed source code. ChatGPT designs each question type to test different parts of programming knowledge:

- For code understanding questions, the model asks students to interpret code structure or logic.
- For output prediction questions, the model generates questions that require students to predict code outputs.
- For code completion questions, the model creates questions with missing code parts, prompting students to recall specific syntax or functions.

4) Question Formatting: The generated questions are formatted appropriately for the learning environment. Code understanding and output prediction questions are set up as *multiple-choice (MCQ)* or *true/false questions*, while code completion questions are formatted as *short-answer fill-in-the-blanks* responses, where students must directly input missing code elements.
5) Output and Review: Finally, the system compiles the questions into a usable format for educators, who can review and adjust them as needed. This organized output enables teachers to incorporate the questions into quizzes, assignments, or practice materials for Python programming education.

To provide a clear understanding of how our program operates, we present the overall architecture of the AI-driven question generation process in Figure 1.

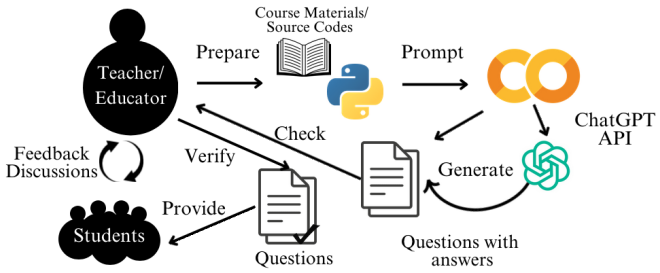

Fig. 1: Overall architecture of AI-driven question generation process.

B. Prompt Design for Effective Question Generation

As effective prompt design is crucial for obtaining high-quality, context-specific questions that align with the educational goals of each topic, the prompt design strategies used to guide ChatGPT in generating relevant and structured questions from Python code snippets will be discussed in this section.

- Contextual Introduction in the Prompt: This sets the scenario for ChatGPT, positioning it as an educational tool and guiding it to produce questions aligned with a teaching goal, thus keeping the model focused on creating educationally relevant questions. The prompt example for contextual introduction can be seen in Figure 2.

```
I am a teacher and want to generate exactly
one {question_type} question for my students
based on the following Python code.
```

Fig. 2: Prompt example for contextual introduction.

- Detailed Specification for Each Question Type: Specific instructions for each question type have been outlined, which is crucial in guiding ChatGPT to produce distinct question formats. By instructing ChatGPT to "Ensure the answers are concise and without any repetition," the prompt aims to maintain the quality and uniqueness of each generated question. The prompt example for detailed specification for each question type can be seen in Figure 3.

```
If {question_type} is **Code Understanding** :
   - Provide exactly one MCQ with 3 choices
   and the correct answer.
   - Provide exactly one True/False question
   with the correct answer.

If {question_type} is **Output Prediction**:
   - Provide exactly one MCQ predicting the
   output with 3 choices and the correct
   answer.
   - Provide exactly one True/False question
   based on a part of the output.

If {question_type} is **Code Completion**:
   - Provide exactly one fill-in-the-blank
   question by removing a key part of the code
   - Provide the correct answer for the blank.

Please generate only the requested question
type: {question_type}. Ensure the answers are
concise and without any repetition.
```

Fig. 3: Prompt example for detailed specification.

C. Automated Question Generation Process and Its Educational Value

This section describes our step-by-step AI-driven approach to generate structured questions from Python code using *ChatGPT's API*, emphasizing how this automation can improve Python education by saving educators time and ensuring a consistent question quality.

1) API Initialization: The program begins by configuring the OpenAI API key, which provides authorized access to ChatGPT's question generation capabilities.
2) Duplicate Filtering: A filtering function in our program removes redundant or repeated content in the generated questions, ensuring each question remains clear and unique. This step enhances readability and ensures the content is concise.
3) Targeted Question Generation: For each Python code snippet, the system uses ChatGPT's LM to generate questions across three main types:

- Code understanding questions are crafted as MCQs or true/false questions that assess students' comprehension of code logic and structure.
- Output prediction questions involve MCQs or true/false questions that ask students to predict code outcomes, helping them develop strong analytical and debugging skills.
- Code completion questions omit critical code elements, prompting students to fill in missing parts. This encourages accurate syntax recall and procedural memory in coding.

4) Batch Processing of Code Files: The program processes each Python source code files in a designated folder iteratively, reading the code snippet and running the question generation function to create questions in all three types. These questions are grouped by file and type for easy review.
5) Output Formatting: The generated questions are saved in an organized output file, segmented by Python file and question type, providing educators with a structured, ready-to-use resource for student assessments.

D. Examples of Question Types

To illustrate the types of questions generated, we will examine a sample Python code snippet as shown in Figure 4 that introduces basic variables and data types. This snippet includes a string, integer, float, and boolean, along with print statements that display these values in simple sentences. By analyzing this code, the AI-driven question generation system creates various questions aimed at testing different aspects of Python knowledge, such as code understanding, output prediction, and code completion.

```
name = "Alice"  # String data type
age = 25  # Integer data type
height = 5.6  # Float data type
is_student = True  # Boolean data type

print(H"ello, my name is", name)
print(I" am", age, "years old.")
print(M"y height is", height, "feet.")
print(A"m I a student?", is_student)
```

Fig. 4: Python source code for *variables and data types*.

1) Code Understanding Questions: assess students' ability to interpret a given Python code snippet, focusing on logic, structure, and purpose. These questions help students gain foundational understanding of programming concepts.

Listing 1: **Generated questions with answers for code understanding.**

```
**Code understanding Questions:**

**Multiple Choice Question (MCQ):**
What data type is used to represent
whether someone is a student or not in the
code?
A) String
B) Integer
C) Boolean
**Correct Answer:** C) Boolean

**True/False Question:**
The variable `height` represents the
height in meters.
**Correct Answer:** False
(It represents the height in feet.)
```

2) Output Prediction Questions: ask students to predict the output of a given code snippet. These questions help develop students' ability to trace code execution, understand variable states, and anticipate program behavior.

Listing 2: **Generated questions with answers for output prediction.**

```
**Output prediction Questions:**

**Multiple Choice Question (MCQ):**
What will be the output of the following
code?
**Choices:**
A.
Hello, my name is Alice
I am 25 years old.
My height is 5.6 feet.
Am I a student? True
B.
I am 25
My height is 5.6
C.
I am years old.
My height is feet.
Am I a student?
**Correct Answer:** A

**True/False Question:**
The output "Am I a student? True" will
appear in the console.
**Correct Answer:** True
```

3) Code Completion Questions: ask students to complete missing parts of a code snippet, such as keywords, functions, or parameters. This type of question emphasizes knowledge of syntax, function usage, and logical structure, helping students to reinforce accurate coding practices.

Listing 3: **Generated questions with answers for code completion.**

```
**Code Completion Question:**

Fill in the blank in the following line of
code to correctly print the statement
about the age.
```python
print(I" am", ___ , "years old.")
```
Correct Answer:
`age`
```

IV. Evaluation

A. Assignments

In this paper, 28 Python source codes were carefully collected from university course materials, online Python tuto-

rials, and real-world coding exercises, covering fundamental and intermediate programming concepts in three introductory courses: Introduction to Python, Intermediate Python, and Intermediate Python for Data Science. Each code snippet was reviewed by Python educators to ensure its clarity, correctness, and alignment with learning objectives. Table I shows courses and the key topics covered for each course for evaluations.

TABLE I: Assignments for evaluations.

Course	ID	Topic
Introduction to Python	1	Variables and Data Types
	2	Arithmetic Operations
	3	Conditional Statements
	4	Loops
	5	Functions
	6	Lists
	7	Dictionaries
	8	Loops with Lists
	9	Error Handling
Intermediate Python	10	File Handling
	11	List Comprehensions
	12	Lambda Functions and Higher-Order Functions
	13	Classes and Objects
	14	Exception Handling
	15	Working with External Libraries
	16	Generators
	17	Decorators
	18	Working with JSON Data
	19	List Sorting with Custom Keys
Intermediate Python for Data Science	20	Matplotlib
	21	Dictionaries and Pandas
	22	Logic, Control Flow, and Filtering
	23	Loops with Matrix Data
	24	Mean, Median, and Standard Deviation
	25	Histogram
	26	Logic Filtering with Pandas
	27	Nested Loops
	28	Correlation

B. Performance Analysis of Question Generation Methods

This subsection examines the effectiveness of different prompting methods: 1) Zero-shot, 2) One-shot, and 3) Few-shot—in generating quality questions for each question type (Code Understanding, Output Prediction, and Code Completion). Table II summarizes the results across each question type and prompt method.

TABLE II: Accuracy of generated questions by prompt method and question type.

Prompt	Code Understanding		Output Prediction		Code Completion
Method	MCQ	T/F	MCQ	T/F	Fill-in-Blank
Zero-shot	96.3%	21.1%	96.3%	17.8%	53.7%
One-shot	100%	100%	100%	100%	100%
Few-shot	100%	100%	100%	100%	100%

In the zero-shot approach, where no example is provided in the prompt, results were mixed. The system showed high accuracy in generating MCQs for both code understanding and output prediction questions, with accuracies of 96.30%. However, true/false question accuracy was significantly lower, at 21.11% for code understanding and 17.78% for output prediction. The code completion questions achieved moderate accuracy (53.70%), indicating that zero-shot prompts might lack the contextual grounding needed for more precise question formats.

While the one-shot method (providing a single example in the prompt) resulted in 100% accuracy across all question types, it exhibited some practical limitations. We observed that one-shot prompts often generated duplicate questions and, in some cases, omitted answers for certain questions. This inconsistency suggests that while one-shot prompting can technically produce the required questions, it may lack the robustness needed for consistent, high-quality outputs in a real-world educational setting.

The few-shot approach, which includes multiple examples (such as giving few instructions for three question types) in the prompt, achieved 100% accuracy across all question types without the issues observed in one-shot prompting. Few-shot prompts provided a more stable generation of unique and fully answered questions, indicating that this approach is more reliable for educational applications where clarity and completeness are essential.

These findings highlight the advantages of using examples in prompts to improve question quality. While zero-shot prompting is effective for generating simpler question types (like MCQs), one-shot prompting may introduce inconsistencies such as duplicate questions and missing answers. Few-shot prompting, on the other hand, provides both accuracy and reliability, making it the preferred choice for generating diverse, complete, and well-structured questions.

Given the observed limitations of zero-shot and one-shot prompting, we ultimately selected questions generated using the few-shot prompting method for application with students. The few-shot approach provided consistent, high-quality questions that were free from duplicates and included complete answer information, ensuring that each question met our educational standards. By using few-shot generated questions, we aimed to deliver a reliable and comprehensive assessment experience that effectively supported students' learning and evaluation.

C. Student Solution Results

This section presents an analysis of student solution results on three types of assignments.

Figure 5 shows the average correct rate (%) for each student across three types of assignments. Most of the students achieved over 80% in *code understanding* and *output prediction* assignments. This indicates that the generated questions for these assignments are effective in helping students understand and predict code behavior, which is essential for foundational programming knowledge.

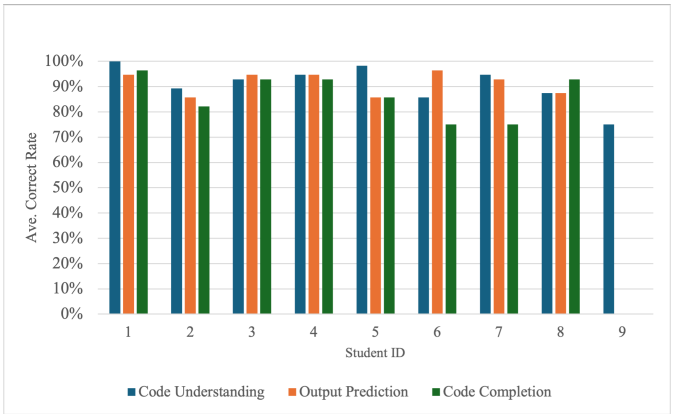

Fig. 5: Solution results for individual students.

In contrast, some students could not reach above 80% in *code completion* assignment. This indicates that students generally find this type of assignment more challenging, as they are requested to answer missing code parts. They may need more practice or guidance in coding tasks that require practical application.

To better interpret these results, we collected information on students' programming knowledge levels. The participants had diverse learning backgrounds in Python, including self-study, where they independently explored Python concepts using online resources and coding platforms; online courses, where they followed structured learning paths with guided instruction and assignments; and classroom-based learning, where they were formally taught Python as part of a university curriculum.

This variation in learning methods may have influenced individual performance, particularly in Code Completion tasks, which require a deeper understanding of syntax and logical structuring. Students with classroom-based instruction may have had more exposure to structured problem-solving techniques, while self-taught learners might have developed a more hands-on, trial-and-error approach. Additionally, those who completed online courses may have benefited from structured content but lacked direct instructor feedback. Understanding these differences is crucial for refining AI-generated questions to better accommodate learners with varying levels of experience and support their progression in Python programming.

TABLE III: Number of students and results in each assignment.

assignment group	# of students	# of questions	ave. score	ave. correct rate (%)
Code Understanding	9	56	50.89	90.87
Output Prediction	8	56	51.25	91.52
Code Completion	8	28	24.25	86.61

Table III provides an overview of student performance across three assignment types, showing average scores and correct rates. These results not only show the students' comprehension and skill levels but also provide evidence for the effectiveness of generated questions in classroom setting.

While *code understanding* and *output prediction* show high average correct rates (90.87% and 91.52%, respectively), indicating the student's proficiency in analyzing and predicting code behavior, *code completion* presents more of a challenge, with a lower average correct rate of 86.61%. This suggests that while the generated questions are effective in reinforcing comprehension and analysis skills, students may benefit from additional practice and support in practical coding tasks.

V. Conclusion

This paper present an AI-driven program that uses *ChatGPT* to automate *question generation* for *Python programming education*. The program reads pre-prepared Python source code at three introductory courses: *Introduction to Python, Intermediate Python, and Intermediate Python for Data Science*. By analyzing each code snippet, the program can generate *three* types of questions with multiple choice, true/false and fill-in-blanks with answers for code understanding, output prediction and code completion questions, covering key programming concepts tailored to the students' skill levels. We evaluated the system using three prompting methods: zero-shot, one-shot, and few-shot. Among these, only questions generated using the few-shot prompting method consistently met the quality standards for accuracy and clarity, making them suitable for use in Python education. Student solution results also showed that the questions generated using few-shot prompting effectively assessed students' understanding, with high accuracy in correct responses across all question types. Future improvements include expanding to more advanced topics, incorporating additional programming languages, and exploring adaptive learning features. We also would like to apply to university classrooms for real-world evaluation.

References

[1] Top Programming Languages 2024. IEEE Spectrum. Available online: https://spectrum.ieee.org/top-programming-languages-2024

[2] M. Heilman, and N. A. Smith, "Good question! statistical ranking for question generation," in Proc. Human language technologies: The 2010 annual conference of the North American Chapter of the Association for Computational Linguistics, pp. 609-617, 2010.

[3] R. Mitkov, H. Le An, and N. Karamanis, "A computer-aided environment for generating multiple-choice test items," Natural language engineering, vol. 12(2), pp. 177-194, 2006.

[4] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J Prather. "The robots are coming: Exploring the implications of openai codex on introductory programming," in Proc. 24th Aus. Comp. Edu. Conf., pp. 10-19, 2022.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, "Language models are few-shot learners," Advances in neural information processing systems, vol. 33, pp. 1877–1901, 2020.

[6] E. Latif, G. Mai, M. Nyaaba, X. Wu, N. Liu, G. Lu, S. Li, T. Liu, and X. Zhai, "Artificial general intelligence (agi) for education," arXiv preprint arXiv:2304.12479, 2023.

[7] C. Kooli, "Chatbots in education and research: A critical examination of ethical implications and solutions," Sustainability vol. 15, no. 7, pp. 5614, 2023.

[8] G. Cruz, J. Jones, M. Morrow, A. Gonzalez, B. Gooch, "An AI System for Coaching Novice Programmers," Lecture Notes in Computer Science, vol. 10296, pp. 12-21, 2017.

[9] Stephen MacNeil, Andrew Tran, Juho Leinonen, Paul Denny, Joanne Kim, Arto Hellas, Seth Bernstein, and Sami Sarsa. "Automatically Generating CS Learning Materials with Large Language Models". in Proc. ACM. Tech. Sym. on Compt. Sci. Edu., pp. 1176, 2023.

Khaing Hsu Wai received the B.E. and M.E. degrees in information science and technology from University of Technology (Yatanarpon Cyber City), Myanmar, in 2016 and 2020, respectively. In 2024, she earned her Ph.D in Graduate School of Natural Science and Technology, Okayama University, Japan. She is currently working as an assistant professor at Graduate School of Engineering Science, Center for Crossover Education, Akita Univesity, Japan. Her research interests include educational technology, learning support systems, AI-driven approaches for data science and programming education, and natural language processing. She is also a member of Language Understanding Lab, Myanmar.

Ye Kyaw Thu is a Visiting Professor at NECTEC, Thailand, where he has been working since January 2019, and Founder of the Language Understanding Lab in Myanmar. He holds a Doctor of Science (2011) and a Master of Science (2006) from Waseda University, Japan, and a Bachelor of Science in Physics from Dagon University, Myanmar (2000). He also earned diplomas in Computer Studies (UK). His research focuses on Artificial Intelligence (AI), Natural Language Processing (NLP), and Human–Computer Interaction (HCI). He supervises students at various institutions, including Assumption University (AU), Kasetsart University, King Mongkut's Institute of Technology Ladkrabang (KMITL), Sirindhorn International Institute of Technology (SIIT), and the Japan Advanced Institute of Science and Technology (JAIST).

Thazin Myint Oo received her Bachelor of Computer Science (B.C.Sc. Hons.) and Master of Computer Science (M.C.Sc.) degrees from the University of Computer Studies, Yangon, in 2005 and 2008, respectively. From 2008 to 2014, she served as an Assistant Lecturer at the University of Computer Studies, Yangon, and from 2014 to 2019, as a Lecturer at Computer University, Kyaing Tong, Myanmar. She was an Associate Professor at the University of Computer Studies, Yangon, from 2019 to 2021. In 2024, she earned her Ph.D. in Computer Science from Assumption University of Thailand. She is currently working as a Senior Researcher at the Language Understanding Lab (LU Lab) in Myanmar.

Nobuo Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

Lu Xiqin received the B.S. degree in electronic information engineering from Hubei University of Economics, China, in 2017, and received the M.S. and Ph.D. degrees in Graduate School of Natural Science and Technology at Okayama University, Japan, in 2021 and 2024, respectively. She is currently an assistant professor in Ritsumeikan University. Her research interests include educational technology and software engineering.

Akihiro Yamamura received the B.Sc. degree in mathematics from Shimane University, Japan, in 1986, and the Ph.D. degree in mathematics from the University of Nebraska–Lincoln, USA, in 1996. He is currently a Professor in the Course of Mathematical Science, Department of Mathematical and Electrical-Electronic-Computer Engineering, Graduate School of Engineering Science, Akita University, Japan. His research interests include discrete mathematics, algebra, cryptography, information security, and theoretical informatics. He is a member of the American Mathematical Society (AMS), the Association for Computing Machinery (ACM), and the Information Processing Society of Japan (IPSJ).