Development and Utilization of AI-Enabled Automatic Programming Problem Generator Using the CodeRunner Plugin of Moodle

CodeRunner+: AI-Enabled Programming Problem Generator

Jay Vince D. Serato
Faculty, College of Computer Studies, Cebu Institute of Technology – University
jayvince.serato@cit.edu

Cherry Lyn C. Sta Romana
Dean, College of Computer Studies, Cebu Institute of Technology – University
cstaromana@cit.edu

ABSTRACT

Programming instructors provide various kinds of problems that suit their current topics of programming. With the use of Learning Management Systems (LMS) such as Moodle, teachers can create and store their problems in problem banks using a question plugin *CodeRunner*. However creating and validating programming problems takes significant time and creative effort. With instructors dealing multiple programming languages, it would need mastery not only to solve the problem but also to use a specific language. This paper presents an automation of the programming problem generation using AI. This is effectively an improvement of the *CodeRunner* plugin of Moodle that allows programming instructors to generate problem descriptions, answers, and testcases of different programming topics in various programming languages with a few clicks. To address the issue of difficulty control, an additional feature is placed to generate an easier or harder problem than what is currently generated. The evaluation of the improved tool showed that the problems generated with AI are similar, correct, and practical that matches the human-generated problems.

CCS CONCEPTS

• **Social and professional topics**; • **Professional topics**; • **Computing education**; • **Computing education programs**; • **Computer science education**; • **CS1**; • **Computing methodologies**; • **Artificial intelligence**; • **Applied computing**; • **Education**; • **E-learning**;

KEYWORDS

Automatic question generation, Programming learning, Generative AI

ACM Reference Format:
Jay Vince D. Serato and Cherry Lyn C. Sta Romana. 2024. Development and Utilization of AI-Enabled Automatic Programming Problem Generator Using the CodeRunner Plugin of Moodle: CodeRunner+: AI-Enabled Programming Problem Generator. In *2024 10th International Conference on Education and Training Technologies (ICETT) (ICETT 2024), April 11–13, 2024, Macau, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3661904.3661921

ICETT 2024, April 11–13, 2024, Macau, China

ACM ISBN 979-8-4007-1789-5/24/04
https://doi.org/10.1145/3661904.3661921

1 INTRODUCTION

Programming problem generation has been one of the many tasks that programming educators have been doing in the facilitation of an effective, objective, and outcomes-based assessment of the skills of programming learners. Programming problems have been prominent in the assessment of problem-solving capabilities of computer studies students. It is also established that the comprehension of programs and its underlying paradigms is a necessity in learning programming [1]. The fundamentals of programming include the concepts of control structures that include sequential, selection, and repetitive statements. Mastery to these concepts are essential to the continuity of the study of programming towards the intermediate levels, especially that it is widely known that there is a steep learning curve [2]. With this, it is imperative that programming educators be able to develop and sustain the knowledge and skills of programming learners. If learners fail to grasp the fundamentals of programming, there is a major risk of student drop-out and a decline in the number of students taking the course.

Furthermore, the paper addresses the following items of concern: the problem generation being a time-demanding ordeal for programming educators, learners needing to hone their programming skills, the limitations of creativity of programming problem generation, and the setting of the difficulty of programming problems.

1.1 Time Necessitated in Problem Generation

The manual process of question construction is a complex task that requires instructor training and prior experience to question generation [3] and the generation of programming problems takes more effort into conceptualizing and identifying the target learning outcomes of programming. Notwithstanding, generating programming problems would have multiple stages rather than conventional question generation.

First, the problem description would have to be comprehensive enough for learners to be able to grasp what problems need to be solved. This will also incorporate the topics to be tested and the programming practices to be utilized. The problem description would need to explain the different set of inputs and their nature, along with the expected output.

Next, the instructor would have to present one's own solution to verify whether the problem description has covered all the necessary details that the solution requires. Despite being an optional step since the solution is not front facing to the learners, the solution is seen to be the target for the learners to achieve as they solve the problem. Hence it is imperative that through the solution that the description be clear to the intended outcomes that the instructor has identified.

Then the instructor would have to provide multiple test cases. Test cases are used to test the program wherein given set of inputs then it shall produce the desired outputs. Test cases are notorious for being shallow such that only a few test cases are added that do not fully encapsulate the breadth and width of the problem. On programming problems that require condition statements, it is important to be able to check corner cases. The efficiency of test cases would prove to be synonymous with generating the answer key, only that there are multiple possibilities of answers for different inputs. This makes test case generation one of the most important steps in programming problem generation, hence more effort is to be allocated in making sure the test cases are comprehensive.

All these steps in generating a programming problem have been manually made by instructors to facilitate learning however performing these steps in each of the problems for an activity proves to be demanding in terms of time and effort to make sure each problem conforms to quality.

1.2 Learners Requiring Programming Practice

Being learners of programming, it is important to be able to practice solving programming questions to hasten their knowledge base and problem-solving capabilities outside of the classroom. It has been established that student initiative to solve programming problems is linked to excellence in programming classes [4]. However, the learner's initiative would mostly come from other sources since instructors typically only reserve their generated questions for their laboratory exercises and examinations. It would be highly beneficial for the learners to practice programming with the types of questions that their instructor would give for the learner to familiarize the concepts and apply the best practices taught by the instructor. The creation of multiple programming problems, or questions in general, is found to positively affect the long-term memory of course knowledge when applied to short answer questions [5] and this could also apply to programming questions.

1.3 Limitations in Creative Efforts

It is already established that manual preparation of programming questions be a challenging task that requires not only time but also extensive effort to ensure a quality assessment [6]. This potentially limits the educator in generating problems that seeks the same learning outcome.

1.4 Setting Programming Problem Difficulty

One of the primary issues of question generation is to generate questions with controlled difficulty [3]. This allows the instructor to generate questions that differ in difficulty since one of the effective techniques of teaching programming is to adapt adaptive learning [7]. With some problems easier and some harder, this would allow the educator to choose in one activity the easier problems and harder problems on the other to facilitate a progressive learning pattern.

2 RELATED WORK

2.1 Automatic Problem Generator

The use of generative artificial intelligence, specifically using Large Language Models (LLMs) for the implementation of an Automatic Question Generation (AQG) is nowadays common with the advent of OpenAI technologies. In fact, this type of studies has been performed in different areas of concern including English education [8], computer education [9], [10] using Generative Pre-trained Transformer (GPT). However, most of the use of AQG is mostly used for multiple-choice questions (MCQ) [9], [10] and short answer questions [11] which argues the level of sophistication of the use of GPTs for complex problems including the generation of programming problems.

While in regard to programming problems and the integration of artificial intelligence to facilitate learning, several studies have utilized existing open-source repositories and generate a template from said repositories to be used in problem solving [12]. Although this technique validates the problem being applied to the real-world scenario, this also limits the nature of questions generated depending on what is available in the dataset.

It has also been recognized that the use of Large Language Models including ChatGPT could potentially assist learners in programming, including understanding the concepts and generate solution codes [13]. With this, it would most definitely assist educators as well in preparing for their programming problems.

Although there already exists some automatic programming problem generator, one study utilizes Context-Free Grammars (CFGs) to create programs based on Python's syntax and semantics [14]. Although this study perfectly takes advantage of Python's unique syntax, it limits the problem generation to a single programming language and the said system could only generate Python code templates and solutions though not problem descriptions that encapsulates the requirements into solving the problem.

Another study provides a software to automatically generate and grade program tracing quizzes [1]. This is an automatic problem generator; it may allow students to understand the code in a sequential manner however, it only limits the system's capabilities that only reaches to the realm of output tracing. This does not qualify as a tool for facilitating problem solving with programming. It may be useful to reflect on this paper to also provide insight into the students' code submissions and not only end at problem generation.

2.2 Learning Management Systems

With the use of Learning Management Systems (LMS) teachers can create and store their programming problems in problem banks. Moodle is one of the most used LMS and is recognized as the best open-source platform [13] as of 2023. With over 160,000 sites utilizing Moodle used by 238 countries, it is noteworthy that Moodle is a major virtual learning environment platform that is currently being used by major institutions [14]. Being able to utilize such a platform as grounds for the implementation and integration of

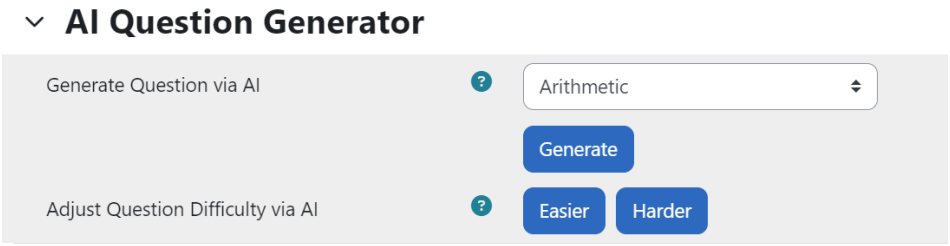

Figure 1: User Interface added into the CodeRunner question page.

advanced educational tools would promote sustainability, and there would be no need for transition towards a new system.

3 METHODOLOGY

3.1 CodeRunner

CodeRunner is a question-type plugin of Moodle that allows teachers to create programming questions and students answer in code and being a question-type plugin, it will also be saved into Moodle's problem bank. CodeRunner verifies the correctness of the answer based on testcases that the teacher also provides. As of September 2023, there are over 3,000 Moodle sites utilizing CodeRunner.

For CodeRunner, the problem generation would be five-fold. First, the teacher selects a programming language. The languages supported by CodeRunner include but are not limited to the following: C, C++, Java, NodeJS, Pascal, PHP, and Python. This step allows CodeRunner to identify and provide the necessary compiler and/or interpreter for checking the students' submissions.

The second step includes the teacher inputting the question name and the question text. The question name is for the instructor to give a title to the question for documentation and will not be shown to the students. The question text shall be the description of the problem where the teacher provides in detail the nature of the problem along with its inputs and expected outputs.

The third step is an optional step to include a code answer to the given problem. Providing the answer allows the CodeRunner system to validate the answer through the testcases to be provided in the next step. This also allows the teacher to guide oneself in clarifying one's description based on the code answer.

The last step is to provide the testcases that shall serve as the point generator of the question. These testcases may be used as an example and each test case might be visible to the students or hidden to avoid brute-force answers. This allows teachers to have control over the availability and scoring of each test case.

Since CodeRunner is an open-source plugin of Moodle, an additional header element is inserted into the CodeRunner's question page, as seen in Figure 1.

This simple interface encapsulates the process of automatically generating a programming problem. It consists of a combo box containing the fundamental and intermediate topics of programming as mentioned in section 3.3. When having chosen, the Generate button is to be clicked. When there already exists a problem description, the problem creator may opt to make the existing question easier or harder by clicking on the respective buttons.

3.2 OpenAI

The use of OpenAI's ChatGPT has become a standard for generative AI, specifically a Large Language Model. This paper utilizes ChatGPT's GPT-3.5 turbo model. Although there exists a more reliable GPT-4 model that utilizes a Large Multimodal Model, the paper has considered the pricing for which GPT-4 would be 30 times more expensive than GPT-3.5. It is also justified to use GPT-3.5 for this program for it does not require accuracy in solving problems. The program only needs to describe the problem, come up with a working program, and generate testcases out of the program. This software also would utilize multiple tokens for the tasks, greatly impacting cost if GPT-4 is to be chosen.

3.3 Programming Problem Concepts

The following topics of programming problems have been identified for programming fundamental course and intermediate course: Arithmetic, Input/Output, Selection, Iterative, Arrays, and Functions. Moreover, subtopics have also been added to ensure that the specific concepts of the topics can be selected by problem creators. The Selection topic would have subtopics on if-statements, if-else statements, if-elseif-else statements, and switch statements. Iterative topic would have while statements, do-while statements, and for statements. Arrays topic would have one-dimensional arrays and two-dimensional arrays. Functions would also have a subtopic option on recursive functions. This allows the problem creator to specify the problem generated into their choice to cater to their student learning targets.

3.4 Prompt Engineering

Tokens are used for language models to provide inputs and generate adequate outputs. The prompt used to generate the results for the programming problem generation must also be precise to only fit the needs of the problem.

It starts with specifying the role of the model. The provision "you are a straightforward assistant" has proven to be able to only provide the direct answer to all queries of the conversation. Then the prompt "Give one programming problem involving " appended by the choice of the problem creator will be sent to OpenAI. The result of this query is used as the problem description in CodeRunner's question text.

If the difficulty buttons are pressed, the problem description will be used to assume as a response, and another prompt is given to provide an easier or harder programming problem.

Table 1: Time Required to Complete Programming Problem Generation

	Selection	Iterative	Arrays	Language Average
C Program	13.8 seconds	14.5 seconds	17.6 seconds	15.3 seconds
C++ Program	12.4 seconds	14.1 seconds	15.7 seconds	14.1 seconds
Java Program	14.6 seconds	13.2 seconds	21.4 seconds	16.4 seconds
NodeJS	17.6 seconds	16.1 seconds	17.2 seconds	17.0 seconds
Pascal	17.5 seconds	14.3 seconds	20.5 seconds	17.4 seconds
PHP	12.8 seconds	14.0 seconds	25.2 seconds	17.3 seconds
Python[a]	15.2 seconds	15.6 seconds	15.6 seconds	15.5 seconds
Average	14.8 seconds	14.5 seconds	19.0 seconds	16.1 seconds

[a]Python 2 and Python 3 are available in CodeRunner, however Python 3 is used for testing.

Then, the prompt to "Write the code of the above problem with standard input and output using [the given programming language] without input prompts, comments, and explanation" and this clear prompt would provide such. Since CodeRunner does not support input prompts e.g., "Enter input:" for the test cases, it is important for the generated code answer to not have input prompts.

Then, the prompt "Give one testcase of the above code using [code answer] with the format "Input:" and "Output:". Do not include explanation" is given. This and its result are then utilized four more times to ensure that there are five different generated testcases each time with the context of the previously generated testcases.

4 RESULTS

The results of this paper are evaluated based on its performance via the time required to generate a programming problem, its cost per problem generation, and the perception of programming educators on the generated problem.

4.1 Performance

Generally, the problem generation in the system requires less than half a minute to complete. This will depend on the tokens required by each of the programming languages. The more syntax necessary, the more tokens needed to complete the problem generation. For example, Java's necessity of its lengthy class declaration and its main function would require more time to generate than its Python equivalent. The summary of the average time to complete a problem generation is as follows:

It is noteworthy that all the problems only require from 12 seconds to 25 seconds to complete, as compared to the traditional way of generating questions which takes significantly longer time and effort to come up with the idea. Using generative AI spares the educator from the said effort and only needs to check manually whether the test cases are sufficient and whether the problem generated suits the concept to be tested. If the latter fails, then they only must generate again.

With the inclusion of difficulty selection, the problem creator needs only generate the problem once and whey they find it too easy or difficult, they can always opt to change the problem's difficulty by generating an easier or harder problem.

4.2 Tokenization and Pricing

Each problem generation has an average of 2,292 tokens (1891 for prompt, 401 for completion). The principles above wherein Java have more constructs than Python, having Java as the selected programming language would also require more tokens, as seen in Table 2.

Pricing for the use of language model is the following: for the prompt tokens, it is USD0.0015 per 1000 tokens and for the completion tokens, USD0.002 per 1000 tokens. Therefore, for each problem generation, it costs USD 0.0035.

4.3 Instructor Assessment and Evaluation

A survey was given to programming educators of Cebu Institute of Technology – University (CIT-U), the university proponent of the paper. This includes instructors from two colleges, the College of Computer Studies that includes the Computer Science (CS) Department, Information Technology (IT) Department, and the College of Engineering's Computer Engineering (CpE) Department. A total of ten educators of programming and programming-related courses have been introduced to the system as part of system validation.

Only five out of ten educators have used CodeRunner in their classes however among these respondents, three of whom still utilize CodeRunner today. When asked for the convenience of the use of the traditional CodeRunner, some respondents disagree on the part of the ease of coming up with a question and some respondents also disagree or are neutral on the part of ease of writing the answer in code and its testcases, as viewed in Figure 1.

Moreover, to assess and validate the system, all the ten educator respondents have been asked to evaluate the improved system and the results are as follows:

Since the problem creator has no need to think about a question, the item has been changed to ease in understanding the generated question, to which all respondents agree and strongly agree. On the time and effort of checking the generated description, answers, and testcases, most respondents agree.

When educator respondents evaluated the rating of the improved system, the program generated an average rating of 4.8 out of 5. This sharply validates the system given that some of the respondents have used the existing system and have positively evaluated the improved built-upon system.

Table 2: Prompt and Completion Tokens Required to Complete Programming Problem Generation

	Selection		Iterative		Arrays		Language Average	
	Prompt	Completion	Prompt	Completion	Prompt	Completion	Prompt	Comple-tion
C Program	1673	316	1866	452	2159	469	1899	412
C++ Program	1687	302	1740	363	2183	469	1870	378
Java Program	1553	319	2299	559	2225	522	2026	467
NodeJS	1912	374	1903	446	1843	386	1886	402
Pascal	1232	186	2234	454	2245	435	1904	358
PHP	1393	219	2319	605	2042	416	1918	413
Python[a]	1683	371	1749	374	1763	390	1732	378
Average	1590	298	2016	465	2066	441	1891	401

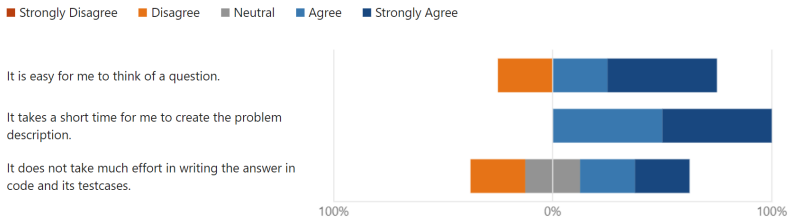

Figure 2: Response results on convenience of the use of traditional CodeRunner.

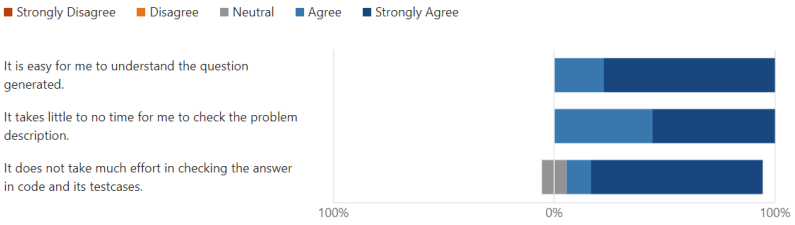

Figure 3: Response results on convenience of the use of the improved CodeRunner.

5 CONCLUSION

Due to the advancement of the use of generative artificial intelligence, automatically generating programming problem questions has become an important tool in facilitating the teaching of introductory and intermediate programming. This allows educators to create programming exercises within seconds with minimal effort built in the comfort of an existing system.

The limitations of this study include the specific language constructs that may be present in specific programming language that are not present among the set of selections of programming problem concepts. The same goes for the inclusion of concepts which are not present in some programming language. The CodeRunner system also has options for a function-type of programming questions to which this paper only considers generation of program-type questions.

REFERENCES

[1] S. Russell, "Automatically Generated and Graded Program Tracing Quizzes with Feedback," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2*, in ITiCSE '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, p. 652. doi: 10.1145/3456565.3460054.

[2] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming: 12 years later," *ACM Inroads*, vol. 10, no. 2, pp. 30–36, Apr. 2019, doi: 10.1145/3324888.

[3] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, "A Systematic Review of Automatic Question Generation for Educational Purposes," *Int. J. Artif. Intell. Educ.*, vol. 30, no. 1, pp. 121–204, Mar. 2020, doi: 10.1007/s40593-019-00186-y.

[4] M. Rahmat, S. Shahrani, R. Latih, N. F. M. Yatim, N. F. A. Zainal, and R. A. Rahman, "Major Problems in Basic Programming that Influence Student Performance," *Procedia - Soc. Behav. Sci.*, vol. 59, pp. 287–296, Oct. 2012, doi: 10.1016/j.sbspro.2012.09.277.

[5] F. Constantinou, "How novel can examination questions really be? Exploring the boundaries of creativity in examination question writing," *Res. Pap. Educ.*, vol. 38, no. 2, pp. 208–226, Mar. 2023, doi: 10.1080/02671522.2021.1961297.

[6] C. Troussas, A. Krouska, and C. Sgouropoulou, "A Novel Teaching Strategy Through Adaptive Learning Activities for Computer Programming," *IEEE Trans. Educ.*, vol. 64, no. 2, pp. 103–109, May 2021, doi: 10.1109/TE.2020.3012744.

[7] U. Lee *et al.*, "Few-shot is enough: exploring ChatGPT prompt engineering method for automatic question generation in english education," *Educ. Inf. Technol.*, Oct. 2023, doi: 10.1007/s10639-023-12249-8.

[8] A. Tran, K. Angelikas, E. Rama, C. Okechukwu, D. Smith, and S. Macneil, *Generating Multiple Choice Questions for Computing Courses using Large Language Models*. 2023.

[9] C. Grévisse, "Comparative Quality Analysis of GPT-Based Multiple Choice Question Generation," in *Applied Informatics*, H. Florez and M. Leon, Eds., in Communications in Computer and Information Science. Cham: Springer Nature Switzerland, 2024, pp. 435–447. doi: 10.1007/978-3-031-46813-1_29.

[10] D. C. L. Tsai, W. J. W. Chang, and S. J. H. Yang, "Short Answer Questions Generation by Fine-Tuning BERT and GPT-2".
[11] A. Prokudin, O. Sychev, and M. Denisov, "Learning problem generator for introductory programming courses," *Softw. Impacts*, vol. 17, p. 100519, Sep. 2023, doi: 10.1016/j.simpa.2023.100519.
[12] M. Liu, Y. Ren, L. M. Nyagoga, F. Stonier, Z. Wu, and L. Yu, "Future of education in the era of generative artificial intelligence: Consensus among Chinese scholars on applications of ChatGPT in schools," *Future Educ. Res.*, vol. 1, no. 1, pp. 72–101, 2023, doi: 10.1002/fer3.10.
[13] A. Ade-Ibijola, "Syntactic Generation of Practice Novice Programs in Python | SpringerLink," presented at the Annual Conference of the Southern African Computer Lecturers' Association, in Communications in Computer and Information Science, vol. 963. 2018, pp. 158–172. doi: https://doi.org/10.1007/978-3-030-05813-5_11.